TECHNISCHE UNIVERSITÄT DRESDEN

Vorlesung
“Service and Cloud Computing"

6. SaaS – verteilte Transaktionen für Microservices

Dr.-Ing. Iris Braun

Gliederung

- Motivation und Beispiel
- Saga Pattern
  - Orchestration Saga
  - Event Choreography Saga
  - Backward Recovery
  - Forward Recovery
- Weitere Lösungen
  - Two-Phase-Commit-Protokoll
  - Spanner
  - Grit
- Frameworks

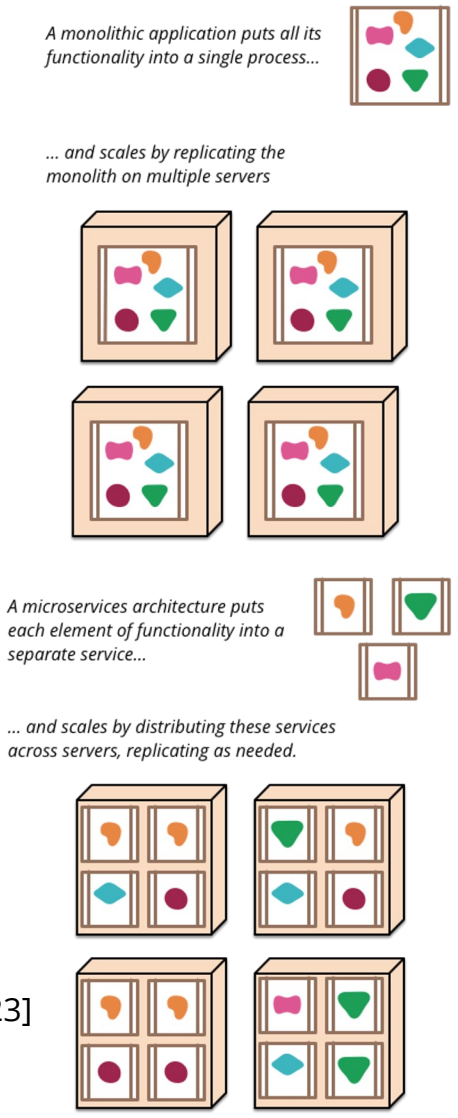
Motivation

Monolithe Software

- Eine Code-Basis
- Als ganzes deployed
- Simple Strukturen
- ACID Transaktionen innerhalb einer Datenbank

Microservice-Architekturen

- **Microservices**: kleine autonome Systeme, welche zusammen arbeiten
- Abhängigkeitsmanagement wird verbessert (Entwicklerteams, Code)
- Versionierung einzelner Services
- Deployment wird verbessert (weniger Downtime & Overhead, Flexibilität)



Vergleich Monolith/Microservices [23]

Motivation

Vorteile durch Microservices:

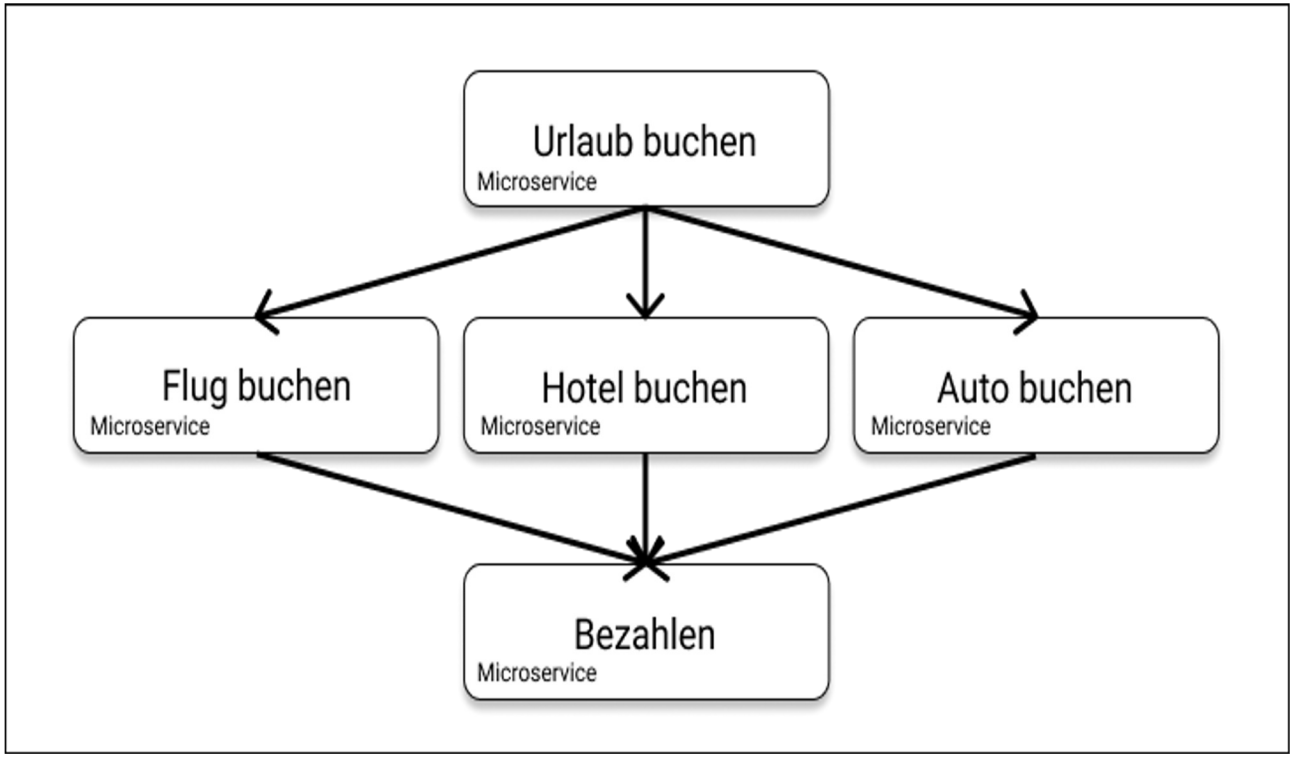
- Ausfallsicherheit
- Effiziente Verteilung der Serverlast durch Load Balancer
- Problemlösung in verschiedenen Sprachen unterschiedlich schwer

Worin liegt die Challenge?

- Jeder Microservice besitzt Daten exklusiv
- Transaktionen in verteilten Systemen
- Transaktionen können aus mehreren Schritten bestehen
- Einzelne Schritte können scheitern
- Long Lived Transactions
- Parallelität der Ausführung

Beispiel

Use Case / Scenario:



Beispielszenario [7]

Saga Pattern

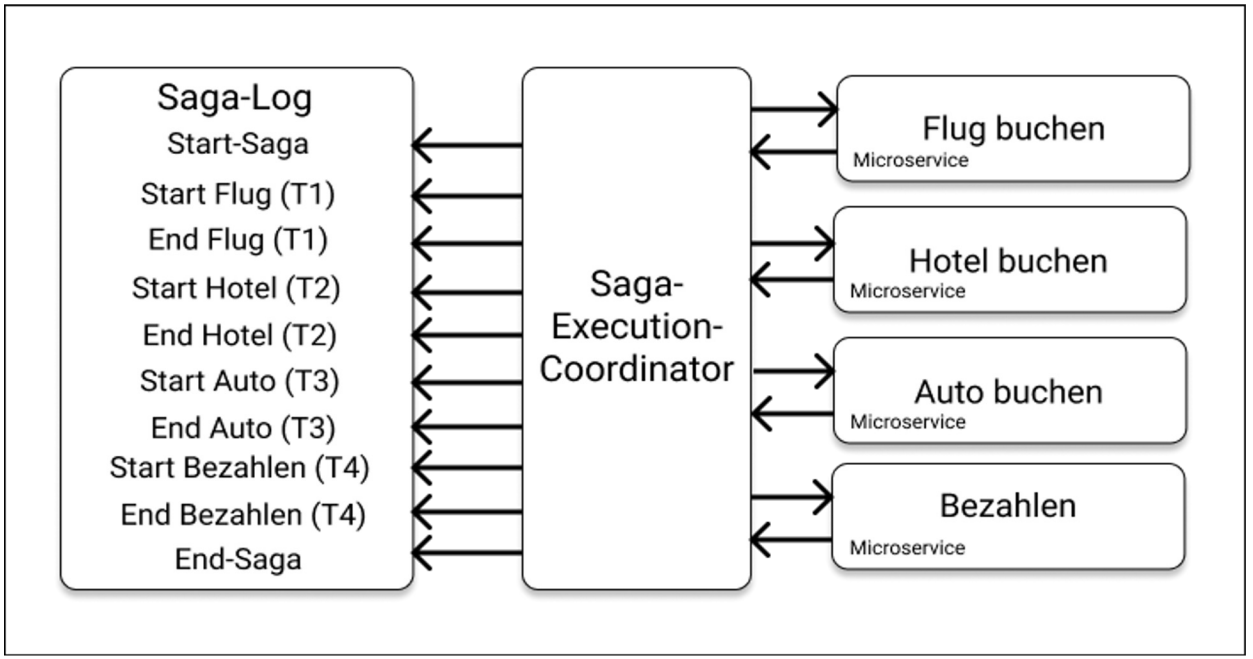
- Ursprung des Saga Patterns von Molina und Salem 1987 [6]
- Wurde für zentrale Datenbanksysteme entwickelt
- Problem damals: Long Lived Transactions, welche Ressourcen blockierten
- Aufsplitten der Transaktion in Teiltransaktionen
- Ausführung als nicht-atomare Einheit
- Jede Teiltransaktion verfügt über Kompensationstransaktion (semantisch)
- Ausführung der Transaktionen idempotent
- **Entweder alle Transaktionen erfolgreich oder alle kompensiert**

Saga Pattern

- **Saga Kommandos:**
  begin-saga, end-saga, begin-transaction, end-transaction, abort-transaction, abort-saga

- **Saga Log:** alle Aktivitäten werden geloggt

- **Saga Execution Coordinator:**
  zentraler Koordinator



Ablauf einer Orchestration Saga

Orchestration Saga

- Zentraler Koordinator

Vorteile

- Komplexer Code leichter verwaltbar
- Gute Reaktion auf viele Anfragen

Nachteile

- Generell langsamer
- Zentraler Angriffspunkt

Event Choreography Saga

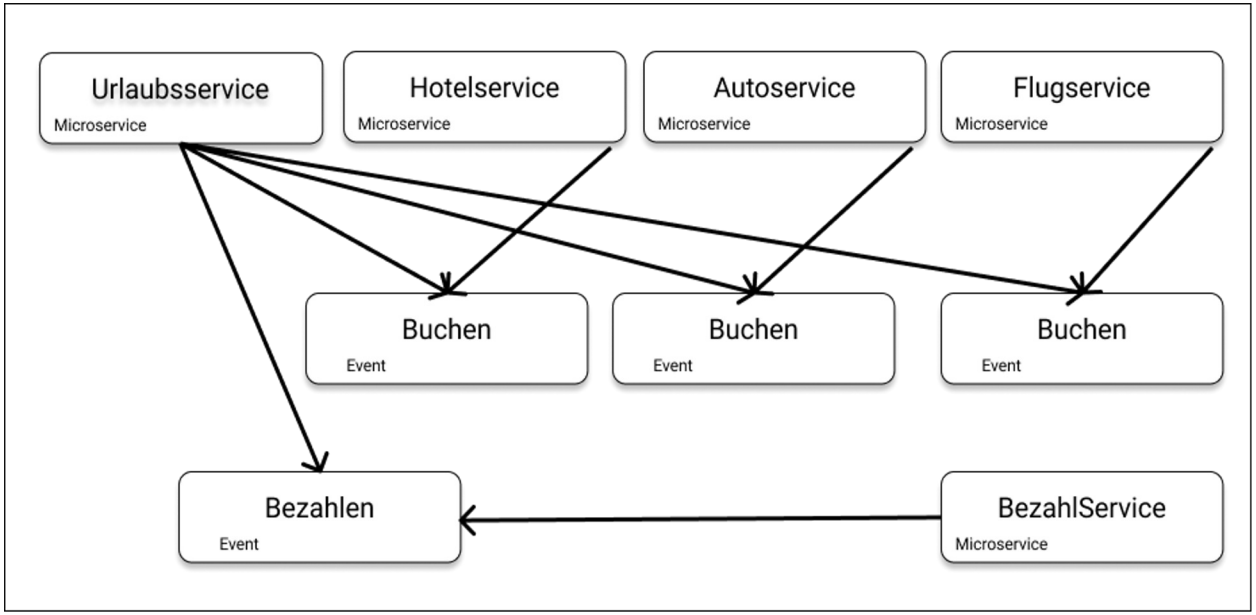
- Kein zentraler Koordinator
- Ablauf über Events

Vorteile

- Schneller als Orchestration

Nachteile

- Code-Komplexität steigt mit Zahl der Microservices
- Abhängigkeiten und Erweiterbarkeit



Ablauf einer Event Choreography Saga

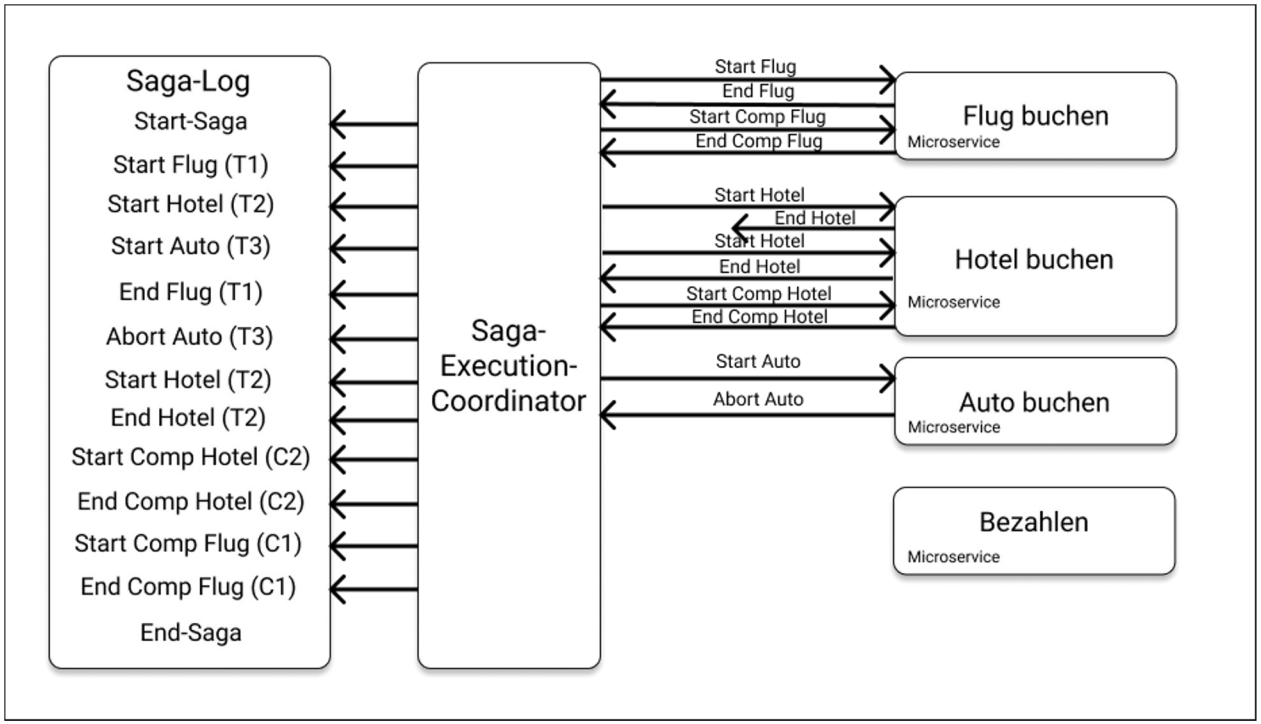
Verhalten bei Fehlern

Wie werden Failures in Saga behandelt?

- Backward Recovery
- Forward Recovery

Backward Recovery

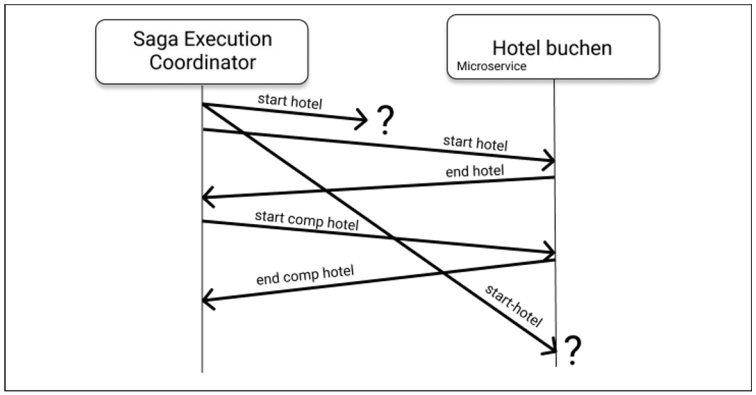
- Bringt System wieder in einen semantischen Ausgangszustand
- Ausgelöst durch abort
- Auswertung des Zustands und Ausführung der Kompensationen
- Wenn alle benötigten Kompensationen ausgeführt wurden, erfolgt *end-saga*



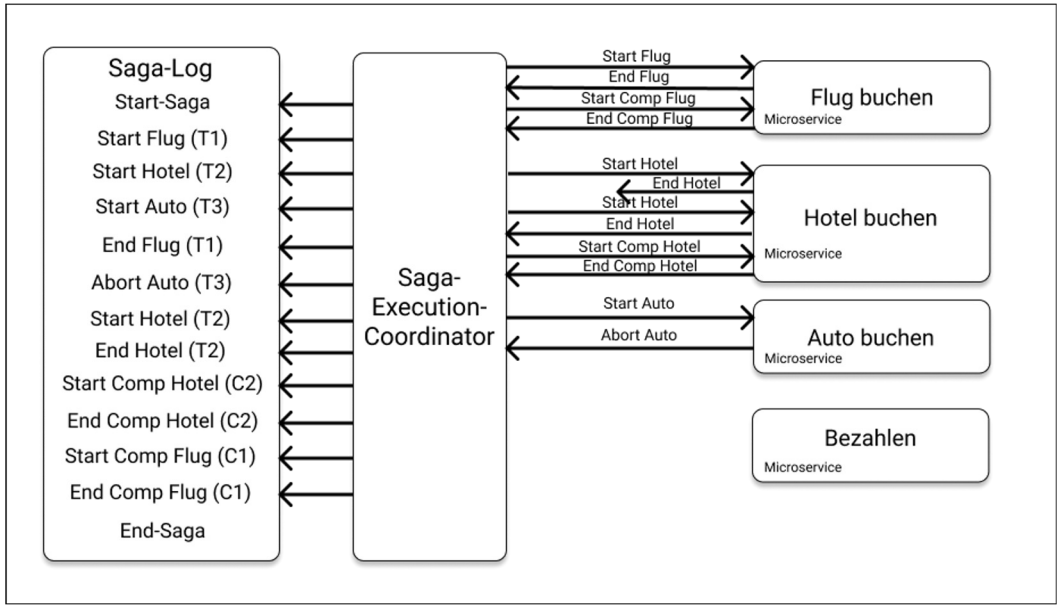
Ablauf einer Backward Recovery

Backward Recovery

Sonderfall start-transaction aber kein *end-transaction*



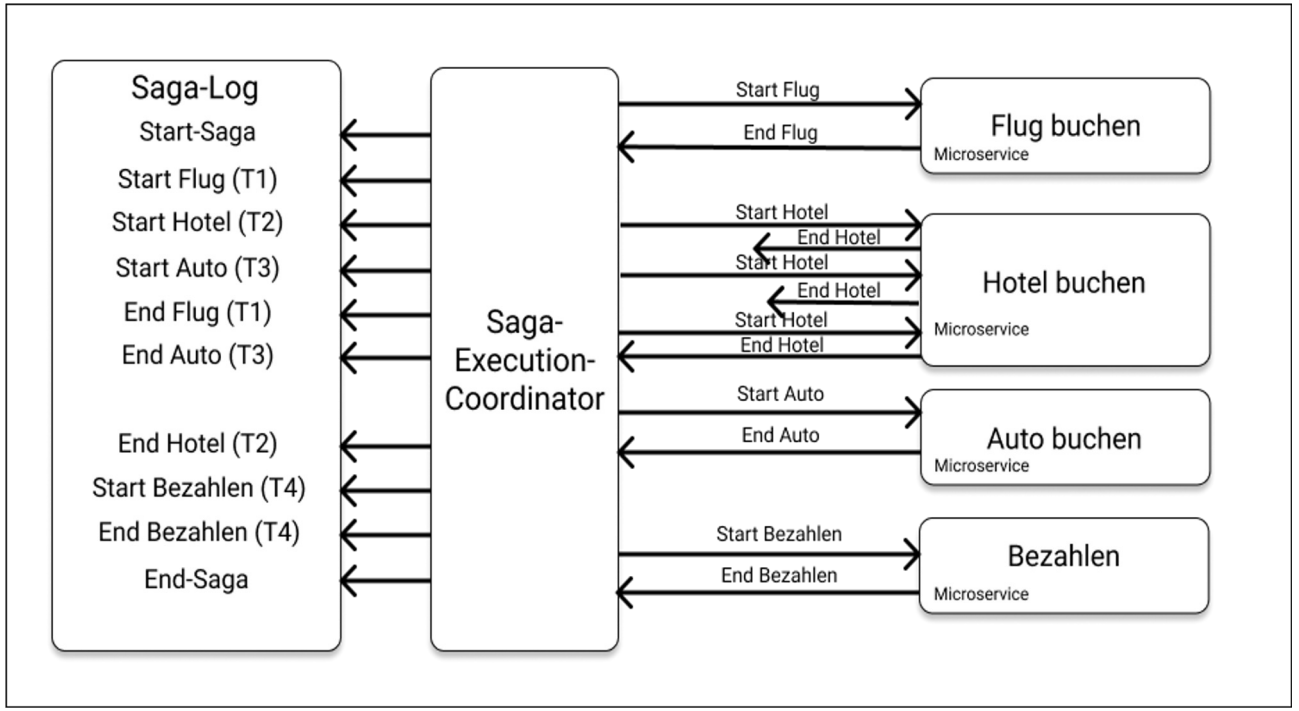
Ablauf keine Bestätigung



Ablauf einer Backward Recovery

Forward Recovery

- Forciert System die Transaktion auf jeden Fall durchzuführen
- Savepoints (Saga Log)
- Verbot von *abort-saga* Kommando
- Nicht immer einsetzbar (Abhängig von Anwendungsfall)



Ablauf einer Forward Recovery

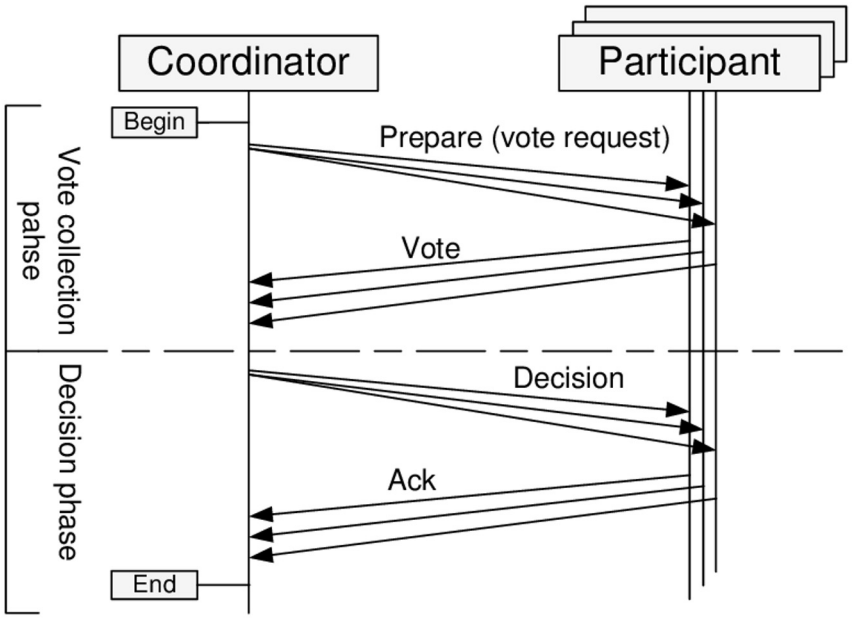
Weitere Lösungen

Two-Phase-Commit-Protokoll

- Atomarität von Transaktionen wird gewährleistet
- Prepare(Voting)-Phase, abstimmen über Durchführung der Transaktion
- Commit(Decision)-Phase, Auswertung über Voting

Nachteil gegenüber Saga Pattern

- Ressourcen werden lange blockiert
- Nur so schnell wie schwächstes Glied

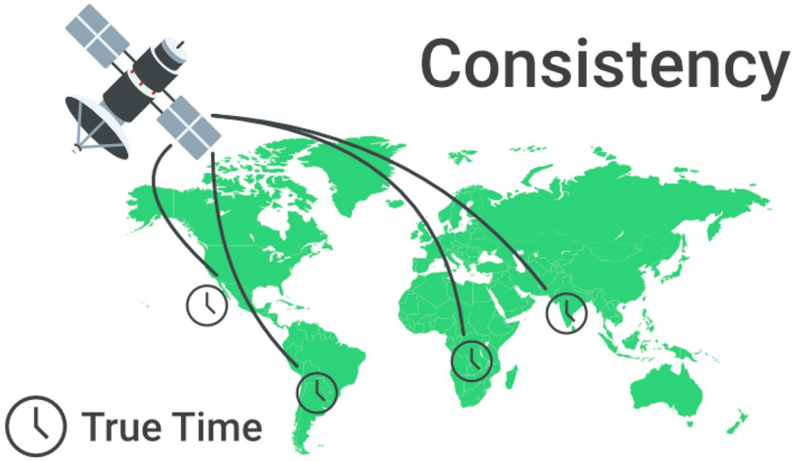


Ablauf Two-Phase-Commit-Protokoll [24]

Weitere Lösungen

Spanner [10]

- Googles globale Datenbank mit hoher Verfügbarkeit
- Globale Datensynchronität über TrueTime API (System von Atomuhren)
- Im CAP Theorem CP-System mit 99,99% Verfügbarkeit



TrueTime API Illustration [25]

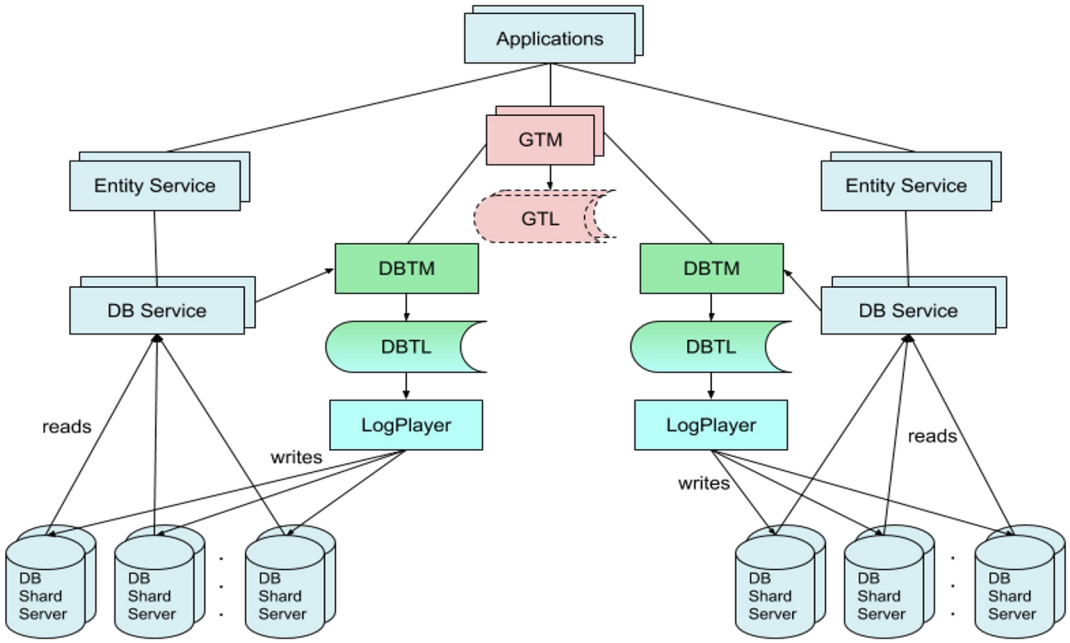
Nachteil gegenüber Saga Pattern

- Proprietär

Weitere Lösungen

GRIT [12]

- Microservices werden mit GRIT-Komponenten erweitert
- Erst theoretische Ausführung und merken des Schreibsets (kein Locking)
- Konfliktauflösung
- Tatsächliche Ausführung der Schreibsets



Aufbau GRIT System [26]

Nachteil gegenüber Saga Pattern

- Aufwändige Architektur

Frameworks

	Generell	Implementiert Saga Pattern	Besonderheiten
Axon [13]	Java Framework (Java Spring Boot microservice Application)	Ja, aber optional über Annotations	Verwendet CQRS, aufwendig für nicht-CQRS Anwendungen
Eventuate [16]	Open Source Plattform, lösen von Datenmanagement Problemen	Nein, außer Eventuate Tram Saga	Verfügt über viele Infrastrukturkomponenten, um verteilte Transaktionen umzusetzen.
Conductor (Netflix) [19]	Microservice basierte Arbeitsschritte verwalten	Nein, Workflow Ansatz ähnlich Saga Pattern	Saga entspricht Workflow, Aufgaben in Workflow sind Transaktionen
Long Running Actions (LRA) [20]	API für Koordination von Aktivitäten mit langer Laufzeit	Ja, orchestration Saga	Transaktionen bestimmen selbst, ob sie im Fehlerfall kompensiert werden müssen
SagaMAS [21]	Saga Pattern mit Multi Agent System kombiniert	Ja	Jeder Service verfügt über eigenen Agent, Agents kommunizieren untereinander, semi-orchestrated

Fazit

- SAGA-Pattern auch heutzutage immer noch relevant (Ursprung 1987)
- Verwendung von Microservice-Architekturen steigt und damit auch der Bedarf an verteilten Transaktionen
- SAGA bietet einfachen Ansatz Probleme von verteilten Transaktionen zu lösen
- Reihe von Frameworks vorhanden, die SAGA-Pattern umsetzen

Quellen

1. G. Vossen, ACID Properties. Boston, MA: Springer US, 2009, pp. 19–21. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9 831
2. What is a monolith? Accessed: 2021-11-24. [Online]. Available: http://www.codingthearchitecture.com/2014/11/19/what is a monolith.html
3. K. Dürr, R. Lichtenthäler, and G. Wirtz, "An evaluation of saga pattern implementation technologies." in ZEUS, 2021, pp. 74–82.
4. Microservices. Accessed: 2021-11-24. [Online]. Available: https://martinfowler.com/articles/microservices.html
5. B. Christudas, Practical Microservices Architectural Patterns: Event-Based Java Microservices with Spring Boot and Spring Cloud. Apress, 2019.
6. H. Garcia-Molina and K. Salem, "Sagas," ACM Sigmod Record, vol. 16, no. 3, pp. 249–259, 1987.
7. Distributed sagas. Accessed: 2021-12-03. [Online]. Available: https://speakerdeck.com/caitiem20/distributed-sagas-a-protocol-for-coordinating-microservices

Quellen

8. C. K. Rudrabhatla, “Comparison of event choreography and orchestration techniques in microservice architecture,” International Journal of Advanced Computer Science and Applications, vol. 9, no. 8, pp. 18–22, 2018.
9. J. Lechtenbörger, “2-phase commit protocol.”
10. E. Brewer, “Spanner, truetime and the cap theorem,” 2017.
11. paxos. Accessed: 2022-01-05. [Online]. Available: http://www4.cs.fau.de/Lehre/WS03/V VA/Skript/VA-4c.pdf
12. G. Zhang, K. Ren, J.-S. Ahn, and S. Ben-Romdhane, “Grit: consistent distributed transactions across polyglot microservices with multiple databases,” in 2019 IEEE 35th International Conference on Data Engineering (ICDE). IEEE, 2019, pp. 2024–2027.
13. axon. Accessed: 2021-12-15. [Online]. Available: https://docs.axoniq.io/reference-guide/v/3.1/part-ii-domain-logic/sagas
14. Cqrs. Accessed: 2021-12-15. [Online]. Available: https://microservices.io/patterns/data/cqrs.html

Quellen

15. M. Stefanko, O. Chaloupka, B. Rossi, M. van Sinderen, and L. Maciaszek, “The saga pattern in a reactive microservices environment,” in Proc. 14th Int. Conf. Softw. Technologies (ICSOFT 2019). SciTePressPrague, Czech Republic, 2019, pp. 483–490.
16. eventuate. Accessed: 2021-12-18. [Online]. Available: https://eventuate.io/abouteventuatetram.html
17. eventuatetram. Accessed: 2021-12-18. [Online]. Available: https://github.com/eventuate-tram/eventuate-tram-core
18. eventuatetramsaga. Accessed: 2021-12-18. [Online]. Available: https://github.com/eventuate-tram/eventuate-tram-sagas
19. conductor. Accessed: 2021-12-19. [Online]. Available: https://netflix.github.io/conductor
20. LRA (Long Running Actions). Accessed: 2021-12-19. [Online]. Available: https://download.eclipse.org/microprofile/microprofile-lra-1.0-M1/microprofile-lra-spec.html

Quellen

21. X. Limón, A. Guerra-Hernández, A. J. Sánchez-García, and J. C. P. Arriaga, “Sagamas: a software framework for distributed transactions in the microservice architecture,” in 2018 6th International Conference in Software Engineering Research and Innovation (CONISOFT). IEEE, 2018, pp. 50–58.
22. msuse. Accessed: 2022-02-01. [Online]. Available: https://www.statista.com/statistics/1236823/microservices-usage-per-organization-size/
23. Microservice https://martinfowler.com/articles/microservices/images/sketch.png
24. Two-Phase-Commit https://www.researchgate.net/profile/Neeraj-Suri/publication/267854236/figure/fig2/AS:647167379243026@1531308135514/The-2-phase-commit-protocol.png
25. TrueTime API https://media.geeksforgeeks.org/wp-content/cdn-uploads/20210107221921/TrueTime.png
26. GRIT https://tech.ebayinc.com/assets/Uploads/Editor/GRIT-Protocol-for-Distributed-Transactions-across-Microservices6.png