MSA 전환 시 분산 트랜잭션 처리 설계 문서

1. 서론

1.1 배경

서비스 확장에 따른 모놀리식 아키텍처의 한계를 극복하기 위해 도메인별로 애플리케이션 서버와 데이터베이스를 분리하는 마이크로서비스 아키텍처(MSA) 전환이 필요한 상황입니다. 이러한 전환 과정에서 가장 중요한 과제 중 하나는 분산 환경에서의 데이터 일관성과 트랜잭션 처리입니다.

1.2 목적

본 문서는 MSA 전환 시 발생하는 분산 트랜잭션 처리의 한계점을 분석하고, 이에 대한 효과적인 대응 방안을 제시하여 시스템의 안정성과 확장성을 확보하는 것을 목표로 합니다.

1.3 범위

- 모놀리식에서 MSA로의 아키텍처 변화 분석
- 분산 트랜잭션 처리의 기술적 한계점 식별
- 대응 방안별 장단점 분석 및 적용 가이드라인 제시

2. 분석

2.1 모놀리식 아키텍처 (As-Is)

구조 특징:

- 단일 애플리케이션 서버에서 모든 비즈니스 로직 처리
- 단일 데이터베이스를 통한 중앙집중식 데이터 관리
- ACID 속성을 완전히 보장하는 로컬 트랜잭션

장점:

- 트랜잭션 관리가 단순하고 직관적
- 데이터 일관성 보장이 용이
- 개발 및 디버깅이 상대적으로 간단

한계점:

- 확장성 제약 (수직적 확장에 의존)
- 기술 스택 다양성 제한
- 부분적 장애가 전체 시스템에 영향

2.2 마이크로서비스 아키텍처 (To-Be)

구조 특징:

- 도메인별 독립적인 서비스 분리
- 각 서비스별 전용 데이터베이스
- API 게이트웨이를 통한 서비스 간 통신

예시 서비스 분리:

- **User Service**: 사용자 관리, 인증/인가
- **Order Service**: 주문 처리, 결제 관리
- **Product Service**: 상품 정보, 재고 관리
- **Notification Service**: 알림 발송

3. 분산 트랜잭션의 한계점

3.1 ACID 속성 보장의 어려움

Atomicity (원자성)

- 여러 서비스에 걸친 작업의 전체 성공/실패를 보장하기 어려움
- 네트워크 장애 시 부분 실패 상황 발생

Consistency (일관성)

- 서비스 간 데이터 동기화 지연으로 인한 일시적 불일치
- 각 서비스의 비즈니스 규칙 검증 복잡성 증가

Isolation (격리성)

- 분산 환경에서 동시성 제어의 복잡성
- 교착 상태(Deadlock) 탐지 및 해결의 어려움

Durability (지속성)

- 일부 서비스의 장애 시 데이터 손실 위험

- 백업 및 복구 전략의 복잡성

3.2 성능 및 확장성 이슈

동기화 오버헤드

- 분산 락(Distributed Lock) 관리 비용
- 합의 알고리즘(Consensus Algorithm)의 성능 영향

데이터베이스 커넥션 관리

- 다중 DB 연결로 인한 리소스 사용량 증가
- 커넥션 풀 관리의 복잡성

4. 대응 방안

4.1 Two-Phase Commit (2PC)

동작 방식:

1. **Prepare Phase**: 코디네이터가 모든 참여자에게 트랜잭션 준비 요청
2. **Commit Phase**: 모든 참여자가 준비 완료 시 커밋 수행

장점:

- ACID 속성을 완전히 보장
- 기존 개발자들에게 친숙한 개념

단점:

- 단일 실패점(Single Point of Failure) 존재
- 블로킹 프로토콜로 인한 성능 저하
- 확장성 제약

적용 권장 상황:

- 금융 거래 등 강한 일관성이 필요한 경우
- 서비스 수가 적고 네트워크가 안정적인 환경

4.2 Saga 패턴

Choreography 방식:

```
주문 프로세스:
Order Service → Payment Service → Inventory Service → Shipping Service
    ↓              ↓                ↓                 ↓
OrderCreated → PaymentProcessed → StockReserved → ShipmentScheduled

실패 시 보상 트랜잭션:
ShipmentCancelled → StockReleased → PaymentRefunded → OrderCancelled
```

Orchestration 방식:
중앙 오케스트레이터가 각 단계를 순차적으로 관리

장점:

- 높은 가용성과 확장성
- 각 서비스의 독립성 유지
- 복잡한 비즈니스 프로세스 처리 가능

단점:

- 최종 일관성(Eventual Consistency)만 보장
- 보상 트랜잭션 설계의 복잡성
- 디버깅 및 모니터링 어려움

4.3 Event Sourcing + CQRS

Event Sourcing:

- 상태 변경을 이벤트 스트림으로 저장
- 현재 상태는 이벤트를 재생하여 계산

CQRS (Command Query Responsibility Segregation):

- 명령(쓰기)과 조회(읽기) 모델 분리
- 각 모델별 최적화된 데이터 저장소 사용

장점:

- 완전한 감사 추적(Audit Trail) 제공
- 시점별 상태 복원 가능

- 높은 확장성과 성능

단점:

- 구현 복잡도 높음
- 이벤트 스키마 진화 관리
- 스냅샷 관리 필요

4.4 Outbox 패턴

동작 방식:

1. 로컬 트랜잭션으로 비즈니스 데이터와 이벤트를 함께 저장
2. 별도 프로세스가 Outbox 테이블을 폴링하여 이벤트 발행
3. 이벤트 발행 후 처리 완료 표시

장점:

- 로컬 ACID와 메시지 발행의 원자성 보장
- 최소한의 아키텍처 변경
- 구현이 상대적으로 단순

단점:

- 폴링으로 인한 지연시간
- Outbox 테이블 관리 필요
- 중복 이벤트 처리 고려

5. 결론

MSA 전환 시 분산 트랜잭션 처리는 복잡하지만 해결 가능한 과제입니다. 핵심은 비즈니스 요구사항에 맞는 적절한 일관성 수준을 선택하고, 단계적으로 전환하는 것입니다.

주요 권장사항:

- **업무 특성에 맞는 패턴 선택**: 강한 일관성 vs 최종 일관성
- **점진적 전환**: 단계적 마이그레이션
