RDBMS 레플리케이션과 데이터 정합성 관리

정승호

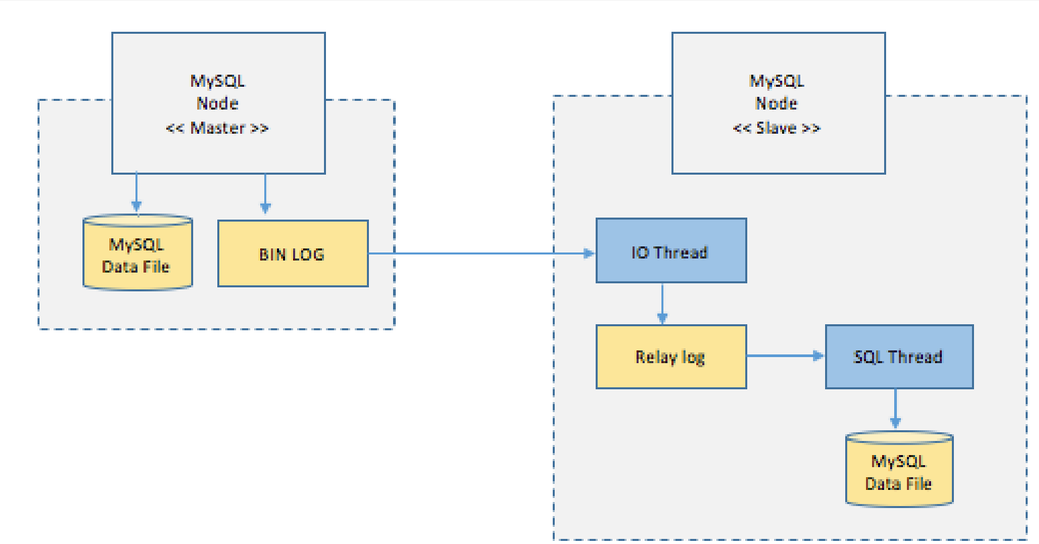
Contents

01. 레플리케이션과 클러스터링

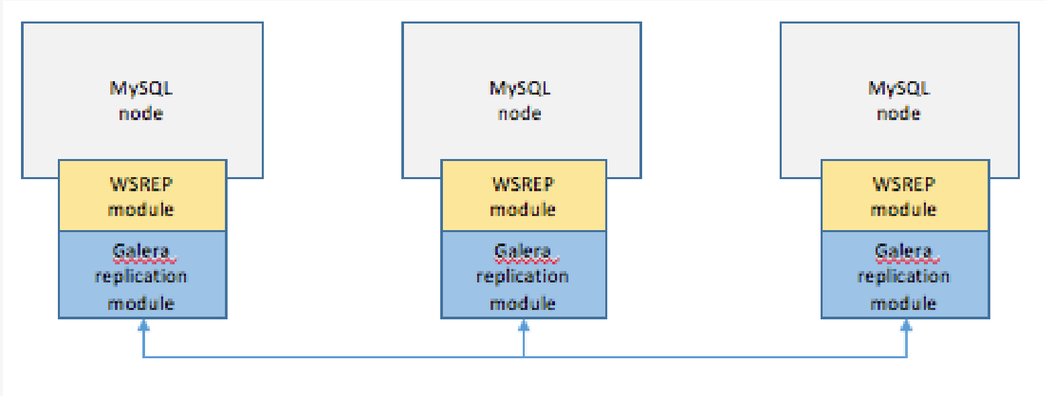
- Replication (복제)
  - 정의: 데이터를 복사하여 여러 노드에 저장하는 기술.
  - 목적: 고가용성(HA), 읽기 성능 확장(Read Scale-Out).

- Clustering (클러스터링)
  - 정의: 여러 서버를 논리적인 단일 시스템으로 묶는 기술.
  - 목적: HA, 부하 분산 (포괄적 방법론).

01. 레플리케이션과 클러스터링



레플리케이션



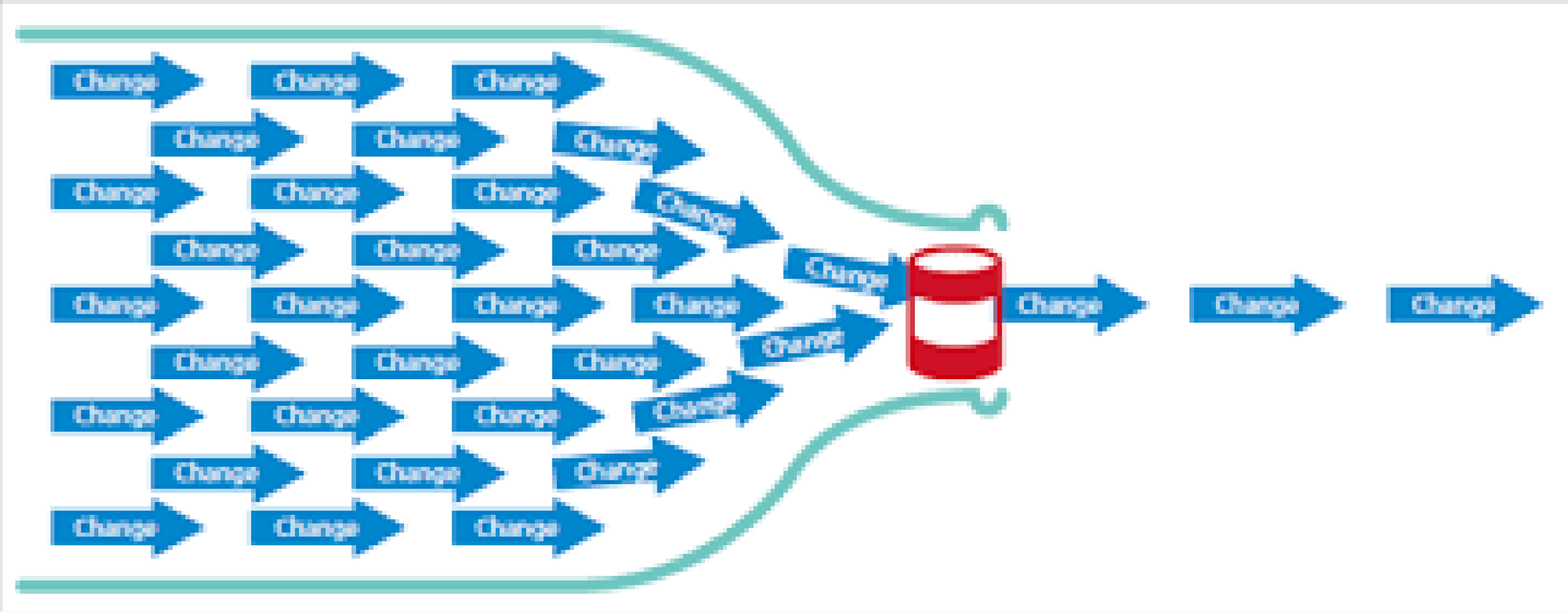
클러스터링

02. RDBMS와 NoSQL에서의 레플리케이션

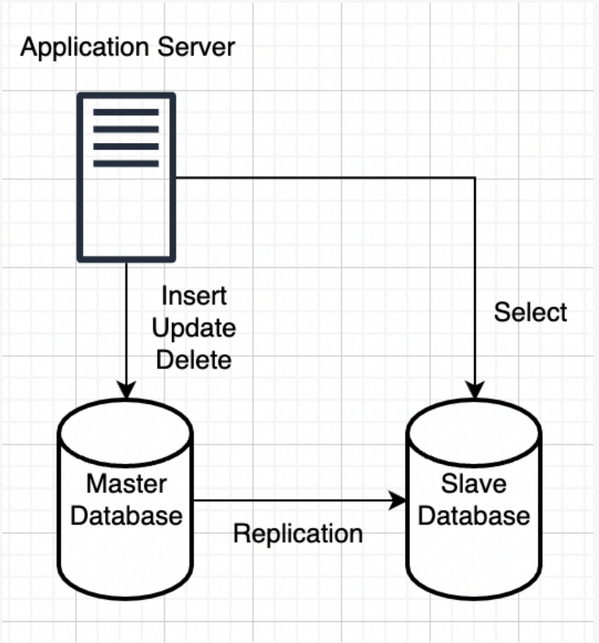
- RDBMS (e.g., MySQL)
  - 모델: Master-Slave Replication.
  - 특징: Binlog 기반 비동기 복제. 고가용성(HA)을 위해선 MHA, Orchestrator 등 별도 솔루션을 통한 수동/반자동 Failover 구성이 필요함.

- NoSQL (e.g., MongoDB, Redis)
  - 모델: Primary-Secondary (MongoDB Replica Set), Sentinel/Cluster (Redis)
  - 특징: 분산 환경을 전제로 설계됨. 자동 Failover (Leader Election)가 기본 내장되어 HA 구성이 용이

03. Read 트래픽 병목



04. Master-Slave 레플리케이션



- Master Node (Write Only)
  - INSERT, UPDATE, DELETE 연산 처리.
  - 변경 이력을 Binary Log (Binlog)에 기록.
- Slave Node (Read Only)
  - I/O Thread: Master의 Binlog를 Polling하여 자신의 Relay Log에 복사.
  - SQL Thread: Relay Log를 순차적으로 실행하여 데이터 동기화.
- 기대 효과: 읽기(Read) 부하를 Slave 노드 수만큼 수평 확장(Scale-Out) 가능.

05. Replication Lag

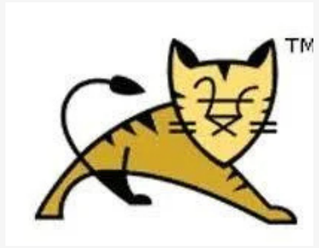
- 정의: Master에 Commit된 변경 사항이 Slave에 적용되기까지 걸리는 물리적인 시간 지연.
  - 원인: 비동기(Asynchronous) 방식, 네트워크 지연, Slave의 SQL Thread 처리 지연.

05. Replication Lag

T1 시점

Client

POST /join

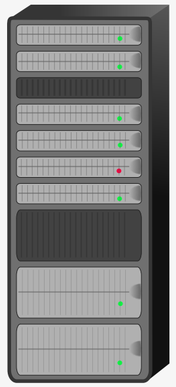


WAS

INSERT
COMMIT

Master DB

Replicating...



Slave DB

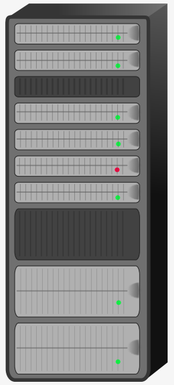
T2 시점

Client

GET /login

WAS

SELECT
Read T2



Slave DB

WAS: Return null (FAIL!)

06. Read/Write 스플리팅

- 목표: 애플리케이션 레벨에서 쿼리의 성격을 인지하고, 동적으로 DB 커넥션을 라우팅(Routing)한다.

- 전략:
  - Write 트랜잭션 (INSERT, UPDATE...) ➔ Master DB로 라우팅.
  - Read-Only 트랜잭션 (SELECT) ➔ Slave DB로 라우팅.

- Spring에서의 식별 기준: @Transactional 어노테이션의 readOnly 속성.
  - @Transactional (또는 readOnly = false) ➔ Master
  - @Transactional(readOnly = true) ➔ Slave

07. Spring에서의 구현 (1) - 핵심 컴포넌트

- AbstractRoutingDataSource (Spring Framework)
- ThreadLocal (Java JDK)
- AOP (@Aspect)

07. Spring에서의 구현 (1) - 핵심 컴포넌트

```java
// MyRoutingDataSource.java
public class MyRoutingDataSource extends AbstractRoutingDataSource {

    // AOP가 ThreadLocal에 설정한 값을 읽어옴
    @Override
    protected Object determineCurrentLookupKey() {
        // "MASTER" or "SLAVE"
        return DataSourceContextHolder.getDataSourceType();
    }
}
```

07. Spring에서의 구현 (1) - 핵심 컴포넌트

```
// DataSourceRoutingAspect.java
@Aspect
@Component
public class DataSourceRoutingAspect {

    @Before("@annotation(transactional)")
    public void setDataSourceKey(Transactional transactional) {
        if (transactional.readOnly()) {
            DataSourceContextHolder.setDataSourceType(DataSourceType.SLAVE)
        } else {
            DataSourceContextHolder.setDataSourceType(DataSourceType.MASTER
        }
    }

    @After("@annotation(transactional)")
    public void clearDataSourceKey() {
        DataSourceContextHolder.clear(); // 끝나면 ThreadLocal 정리
    }
}
```

07. Spring에서의 구현 (2) - 기본 동작 흐름

1. @Transactional(readOnly = true)가 붙은 서비스 메서드 호출.
2. AOP (@Before): readOnly = true 확인 ➔ ThreadLocal.set("SLAVE").
3. Repository가 쿼리 실행 (DB 커넥션 요청).
4. RoutingDataSource: determineCurrentLookupKey()가 ThreadLocal.get() 호출 ➔ "SLAVE" 반환.
5. RoutingDataSource가 Slave DB 커넥션을 반환함.
6. AOP (@After): ThreadLocal.remove()로 컨텍스트 정리.

07. Spring에서의 구현 (3) - 'Read after Write' 정합성 확보

- 남은 문제: '회원가입 직후 조회' 시나리오.
- Data Inconsistency 문제
  - T1: (Master) INSERT INTO user ... (회원가입 Commit).
  - T2: (Slave) SELECT * FROM user ... (로그인 시도).
  - 결과: T1의 데이터가 T2 시점에 아직 복제되지 않아, SELECT 결과가 null 이 됨.

- INSERT (Write) 트랜잭션과 SELECT (Read-Only) 트랜잭션이 분리되어 있어, SELECT가 Slave로 라우팅되는 문제가 여전함.

- 해결: ThreadLocal의 컨텍스트 유지 범위를 트랜잭션 단위가 아닌, HTTP Request 단위로 확장.

07. Spring에서의 구현 (3) - 'Read after Write' 정합성 확보

- 개선된 AOP 로직:
  - @Transactional(readOnly = false) (쓰기) 트랜잭션을 감지하면, ThreadLocal.set("MASTER").
  - 이 "MASTER" 플래그는 트랜잭션이 끝나도 **HTTP 요청이 끝날 때**까지 지우지 않음.
  - 이후 readOnly = true 트랜잭션이 호출되어도, AOP는 ThreadLocal에 "MASTER"가 이미 설정되어 있으면 readOnly 속성을 무시하고 Master 설정을 유지함.
  - HTTP 요청이 종료될 때 Filter 또는 Interceptor에서 ThreadLocal.remove().
- 결과: 한번이라도 쓰기 작업을 수행한 HTTP 요청은, 해당 요청이 끝날 때까지 모든 읽기 작업을 Master에서 수행

08. 결론

- 문제: Read 부하 분산을 위해 Master-Slave 레플리케이션 도입.
- 과제: Replication Lag로 인한 데이터 정합성 문제 발생.
- 해결
    - Spring AOP, RoutingDataSource, ThreadLocal을 조합하여 readOnly 속성 기반 Read/Write 스플리팅 구현.
    - 'Read after Write' 문제는 ThreadLocal의 컨텍스트 생명주기를 HTTP Request 단위로 확장하여 해결.
- 아키텍처는 성능(Performance)과 정합성(Consistency) 간의 트레이드오프(Trade-off)를 고려해야 함.
- 비동기 환경의 데이터 불일치 문제는, 애플리케이션 레벨의 컨텍스트 관리를 통해 해결할 수 있음.

Thank you

감사합니다.