**로그 시스템 개발**

**로그 시스템에서 외부 SDK를 사용하는 이유**

**1. 성능 문제: DB는 대량의 로그를 실시간으로 감당할 수 없다**

**(1) 초당 수천 개의 INSERT 요청 → DB 부하 증가**

* **트랜잭션 부하:**
  + INSERT마다 디스크 I/O가 발생 → DB 성능 저하
  + 트랜잭션이 너무 많으면 락이 걸려서 전체 성능이 떨어짐
* **DB 커넥션 부족:**
  + MMORPG 같은 대규모 서버는 게임 클라이언트도 DB를 사용함
  + 여기에 로그까지 INSERT 하면 DB 커넥션을 너무 많이 잡아먹음
  + **결과:** 로그 저장 때문에 게임 서버 성능도 느려짐

**2. 장애 대응: 장애 발생 시 로그 유실 문제**

**(2) DB가 죽으면 로그가 날아감**

* DB는 장애가 나면 저장이 불가능
* 게임 서버는 계속 로그를 남기려 하지만, DB가 다운되면 로그가 날아감
* **결과:** 나중에 문제를 추적하려 해도 로그가 없어서 원인 분석이 불가능

**해결 방법:** Redis, Kafka 같은 미들웨어를 거쳐서 로그를 저장하면,

* **DB 장애 시에도 로그를 안전하게 쌓아둘 수 있음**
* DB가 복구된 후 로그를 다시 처리 가능

**3. 효율적인 로그 처리: 배치(일괄 처리) 방식이 필요함**

**(3) 로그를 쌓아뒀다가 한꺼번에 DB로 저장해야 함**

* MMORPG 서버에서는 로그가 초당 수천 개 이상 발생
* DB는 "일정량을 모아서 한 번에 INSERT 하는 방식"이 훨씬 성능이 좋음
* Kafka, Redis 같은 미들웨어를 사용하면 **로그를 일시 저장**하고,  
  **일정 시간마다 배치로 묶어서 DB에 저장** 가능

**Bulk Insert란?**

* **한 번의 INSERT로 여러 개의 데이터를 삽입하는 기법**
* 개별 INSERT보다 훨씬 빠르고 효율적임
* 로그 시스템, 대량 데이터 적재에 자주 사용됨

**Bulk Insert의 장점**

1. **DB 부하 감소**
   * 개별 INSERT는 트랜잭션과 인덱스 업데이트가 매번 발생 → 성능 저하
   * Bulk Insert는 한 번에 처리하므로 오버헤드 감소
2. **성능 향상**
   * 예를 들어, 10,000개의 로그를 한 번에 삽입하면 성능이 훨씬 좋아짐
3. **트랜잭션 관리 최적화**
   * 여러 개의 INSERT를 하나의 트랜잭션으로 처리 가능
   * DB의 트랜잭션 특성(ACID)을 유지하면서도 성능을 극대화할 수 있음

**결론: 미들웨어가 필요한 이유**

|  |  |  |
| --- | --- | --- |
| **문제** | **DB 직접 Insert** | **미들웨어 사용** |
| **성능** | 초당 수천 개 Insert로 DB 부하 심함 | 로그를 모아두고 한 번에 저장 |
| **장애 대응** | DB가 죽으면 로그 유실 | DB가 죽어도 로그 저장 가능 |
| **배치 처리** | 매 요청마다 Insert | 일정량을 모아서 효율적으로 저장 |

**로그 시스템에서 자주 사용하는 외부 SDK 특징**

**1. Redis**

* **특징:** 메모리 기반의 고속 데이터 저장소
* **장점:**
  + 빠른 데이터 처리 속도 (메모리 기반)
  + 다양한 데이터 구조 지원 (리스트, 해시, Sorted Set 등)
  + 내장된 Stream 기능을 활용하면 로그 데이터 큐잉 가능
* **단점:**
  + 영속성이 약함 (AOF, RDB 방식 제공하지만 디스크 기반 DB보다 신뢰성 낮음)
* **로그 시스템에서의 활용:**
  + 임시적으로 로그를 저장하고 배치 처리 시 활용

**2. NATS.io**

* **특징:** 초경량 메시지 브로커, low latency, 분산 아키텍처에 최적화
* **장점:**
  + 매우 가볍고 빠른 메시징 (마이크로서비스 및 실시간 이벤트 처리에 적합)
  + 메시지 지연 시간이 짧고 고성능
  + 간단한 설치와 운영
* **단점:**
  + 메시지 지속성이 약함 (JetStream을 사용해야 메시지 저장 가능)
  + Kafka처럼 강력한 메시지 보장 기능 부족
* **로그 시스템에서의 활용:**
  + 실시간 로그 스트리밍
  + 빠른 처리 속도가 필요한 이벤트 기반 로그 전달

**3. Kafka**

* **특징:** 대용량 분산 로그 스트리밍 플랫폼, 메시지 큐 + 저장소 역할
* **장점:**
  + 강력한 내구성 (데이터 영속성 보장)
  + 대량 데이터 처리에 최적화됨 (분산 처리)
  + 메시지 재처리 및 리플레이 가능
* **단점:**
  + 운영이 비교적 복잡함 (클러스터 구성, 모니터링 필요)
  + 단순한 메시징보다는 대규모 데이터 스트리밍에 적합
* **로그 시스템에서의 활용:**
  + 로그를 안정적으로 저장하고 배치로 처리

장애 시 로그 유실 방지를 위한 대응정책

**1. Redis**

* **장애 방지 옵션: RDB, AOF (디스크 저장)**
* **Redis Sentinel** 또는 **Redis Cluster**를 사용하면 자동 Failover

**2. NATS.io**

* **JetStream: 디스크 저장**
* **Acknowledgement 설정: 로그가 DB로 저장되었는지 확인 후 삭제**

**3. Kafka**

* **장애 방지 옵션: Replication(복제), Ack(유실 방지), Retention(로그 유지 기간)**

**어떤 걸 선택해야 할까?**

1. **실시간 로그 처리 + 초고속 메시징이 필요하면** → **NATS.io**
2. **빠른 임시 저장소 + 주기적 배치 전송이 필요하면** → **Redis**
3. **안정적인 대량 로그 처리 + 영속성 보장이 필요하면** → **Kafka**

| **항목** | **Kafka** | **Redis Streams** | **NATS JetStream** |
| --- | --- | --- | --- |
| 메시지 전송 | 디스크 기반 (append log) | 메모리 기반 + RDB/AOF | 메모리 기반 (옵션: 디스크) |
| 소비 방식 | Pull-based | Blocking pop/poll | Push or Pull |
| 전송 레이턴시 | 수 ms~수십 ms | 수 μs~수 ms | 수 μs~수 ms |
| Throughput (기본) | 중간 (세팅에 따라 높음) | 높음 | 높음 |
| 사용 복잡도 | 높음 (설정 많음) | 낮음 | 중간 |

**추천 조합**

* Redis를 사용해 일시적으로 로그를 저장 → 일정량이 쌓이면 DB로 Insert

현재 MMORPG 서버라면 **Redis**이 가장 현실적인 선택이 될 가능성이 커. Redis로 임시 캐싱 후 DB로 대량 로그를 처리하면 효율적일 거야. 팀에서 논의할 때 이 점을 강조해 보면 좋을 것 같아!