

대용량 트래픽 서버를 Kafka/Redis로 만들며 배운 것들

김지호
KIM JEEHO
Web Backend
Engineer

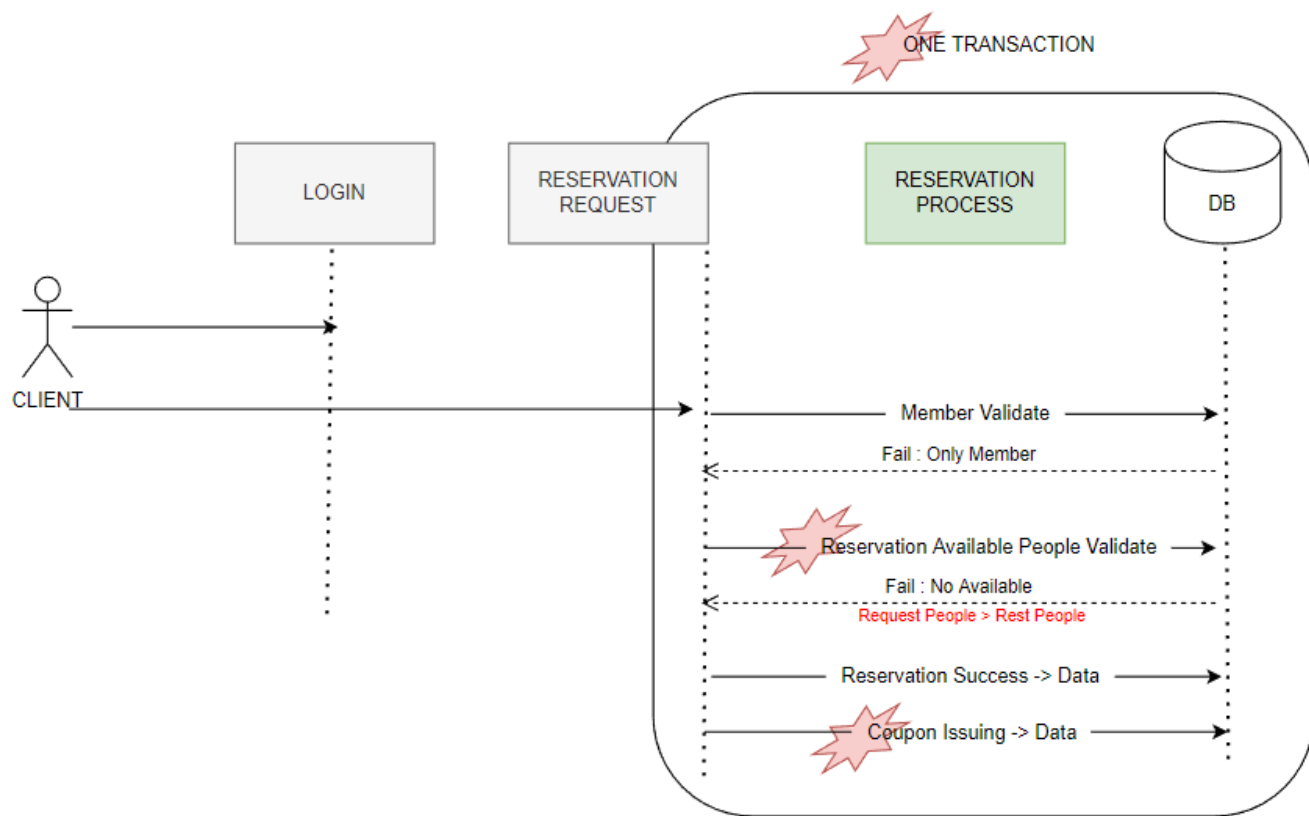
Things I've learned
From making high-capacity traffic server
Using Kafka/Redis



목차

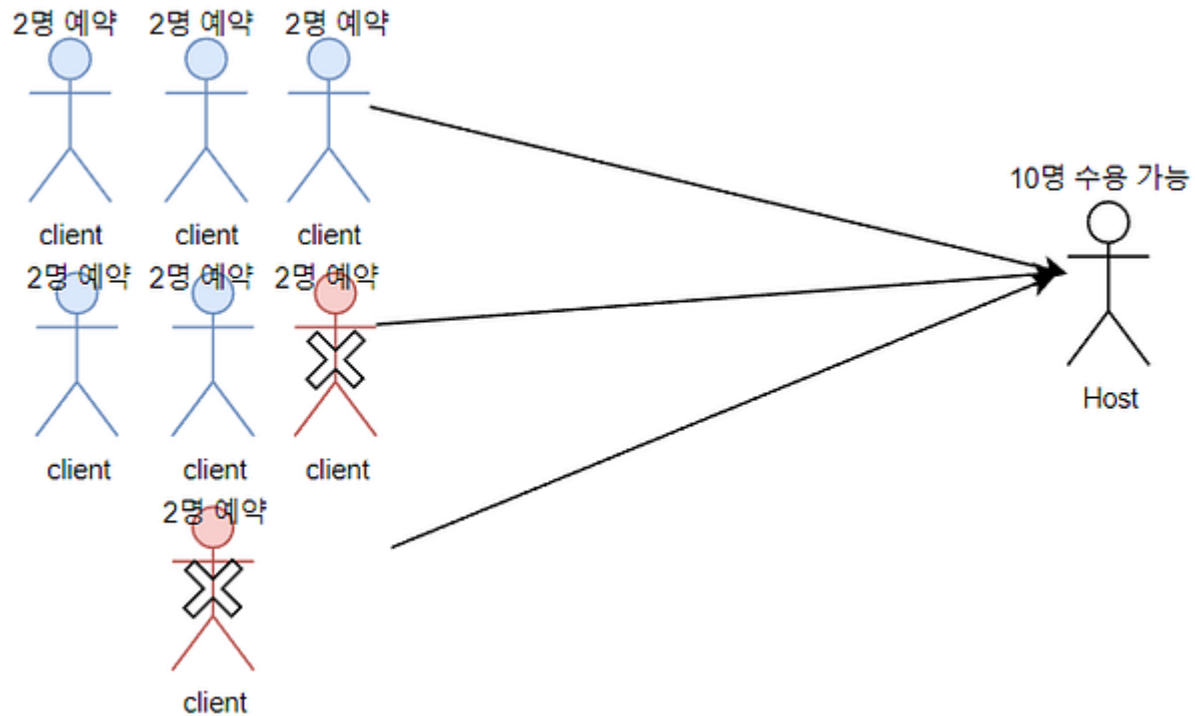
1. 문제 상황 – 동시성 고려하지 못한 서버 설계
 - 멀티 스레드와 동시성
2. 해결 방법 – Kafka/Redis/Redisson 기술과 개념 정리
3. 개선된 서버와 생각할 거리

1. 문제 상황 – 요건에 맞추어 군더더기 없이 만들어진 설계와 코드



- 회원에 한해서 예약이 가능
- 예약 요청 인원을 재고 요청 인원과 비교하여,
요청 인원 < 재고 인지 확인
- 재고 인원이 확보되는 경우 예약 테이블에 저장
- 쿠폰을 발급 받지 않은 회원인지 확인
- 쿠폰을 발급 받지 않은 회원에게 쿠폰 발급
(=쿠폰 이력 테이블에 저장)

1. 문제 상황 - 기대 결과



- 2명의 예약 신청 7개가 동시에 들어옴
- 요청 시도 7개 중 최대 수용 가능 인원 10명에 맞추어 5개 요청 성공
- 5개 요청이 성공하면 HOST의 수용 가능 인원 0명이 되며,
이후 요청은 재고 부족으로 예약 거절
- 예약 성공 요청에 대해서는 쿠폰 발급

1. 문제 상황 - 실제 결과

5/7 스레드 할당

```
-- [pool-2-thread-4] c.e.s.service.resrv.ResrvServiceImpl : 현재 호스트 예약 재고 : 10
-- [pool-2-thread-2] c.e.s.service.resrv.ResrvServiceImpl : 현재 호스트 예약 재고 : 10
-- [pool-2-thread-5] c.e.s.service.resrv.ResrvServiceImpl : 현재 호스트 예약 재고 : 10
-- [pool-2-thread-3] c.e.s.service.resrv.ResrvServiceImpl : 현재 호스트 예약 재고 : 10

1_2_0_, host0_.created_date as created_2_2_0_, host0_.modified_date as modified3_2_0_, ho
0_, host0_.apprv_yn as appr_yn6_2_0_, host0_.apprv_date as appr_da7_2_0_, host0_.farmsta
.intro as intro10_2_0_, host0_.ivt_ppl as ivt_ppl11_2_0_, host0_.max_ppl as max_ppl12_2_0
shortin14_2_0_, host0_.id as id15_2_0_ from host host0_ where host0_.hnum=?

-- [pool-2-thread-1] c.e.s.service.resrv.ResrvServiceImpl : 현재 호스트 예약 재고 : 10
```

```
28124 --- [pool-2-thread-3] c.e.s.service.resrv.ResrvServiceImpl : 사용자 예약 완료 / 현재 재고: 8
his (acct_yn, acctd_date, end_date, hnum, req_ppl, start_date, userid) values (?, ?, ?, ?, ?, ?, ?)
um as cpn_num1_0_0_, cpn0_.created_date as created_2_0_0_, cpn0_.modified_date as modified3_0_0_, cpn0_.ivt_cnt as
as max_cnt5_0_0_, cpn0_.reg_dt as reg_dt6_0_0_, cpn0_.reg_id as reg_id7_0_0_ from cpn cpn0_ where cpn0_.cpn_num=?
his (acct_yn, acctd_date, end_date, hnum, req_ppl, start_date, userid) values (?, ?, ?, ?, ?, ?, ?)
28124 --- [pool-2-thread-4] c.e.s.service.resrv.ResrvServiceImpl : 사용자 예약 완료 / 현재 재고: 8
his (acct_yn, acctd_date, end_date, hnum, req_ppl, start_date, userid) values (?, ?, ?, ?, ?, ?, ?)
um as cpn_num1_0_0_, cpn0_.created_date as created_2_0_0_, cpn0_.modified_date as modified3_0_0_, cpn0_.ivt_cnt as
as max_cnt5_0_0_, cpn0_.reg_dt as reg_dt6_0_0_, cpn0_.reg_id as reg_id7_0_0_ from cpn cpn0_ where cpn0_.cpn_num=?
col_0_0_ from cpn_issu cpnissu0_ where cpnissu0_.cpn_num=? and cpnissu0_.userid=?
col_0_0_ from cpn_issu cpnissu0_ where cpnissu0_.cpn_num=? and cpnissu0_.userid=?
28124 --- [pool-2-thread-1] c.e.s.service.resrv.ResrvServiceImpl : 사용자 예약 완료 / 현재 재고: 8
um as cpn_num1_0_0_, cpn0_.created_date as created_2_0_0_, cpn0_.modified_date as modified3_0_0_, cpn0_.ivt_cnt as
as max_cnt5_0_0_, cpn0_.reg_dt as reg_dt6_0_0_, cpn0_.reg_id as reg_id7_0_0_ from cpn cpn0_ where cpn0_.cpn_num=?
28124 --- [pool-2-thread-5] c.e.s.service.resrv.ResrvServiceImpl : 사용자 예약 완료 / 현재 재고: 8
col_0_0_ from cpn_issu cpnissu0_ where cpnissu0_.cpn_num=? and cpnissu0_.userid=?
um as cpn_num1_0_0_, cpn0_.created_date as created_2_0_0_, cpn0_.modified_date as modified3_0_0_, cpn0_.ivt_cnt as
as max_cnt5_0_0_, cpn0_.reg_dt as reg_dt6_0_0_, cpn0_.reg_id as reg_id7_0_0_ from cpn cpn0_ where cpn0_.cpn_num=?
col_0_0_ from cpn_issu cpnissu0_ where cpnissu0_.cpn_num=? and cpnissu0_.userid=?
28124 --- [pool-2-thread-2] c.e.s.service.resrv.ResrvServiceImpl : 사용자 예약 완료 / 현재 재고: 8
um as cpn_num1_0_0_, cpn0_.created_date as created_2_0_0_, cpn0_.modified_date as modified3_0_0_, cpn0_.ivt_cnt as
as max_cnt5_0_0_, cpn0_.reg_dt as reg_dt6_0_0_, cpn0_.reg_id as reg_id7_0_0_ from cpn cpn0_ where cpn0_.cpn_num=?
col_0_0_ from cpn_issu cpnissu0_ where cpnissu0_.cpn_num=? and cpnissu0_.userid=?
```

```
acct_yn, acctd_date, end_date, hnum, req_ppl, start_date, userid) values (?, ?, ?, ?, ?, ?, ?)
acct_yn, acctd_date, end_date, hnum, req_ppl, start_date, userid) values (?, ?, ?, ?, ?, ?, ?)
--- [pool-2-thread-5] c.e.s.service.resrv.ResrvServiceImpl : 사용자 예약 완료 / 현재 재고: 6
--- [pool-2-thread-4] c.e.s.service.resrv.ResrvServiceImpl : 사용자 예약 완료 / 현재 재고: 6
```

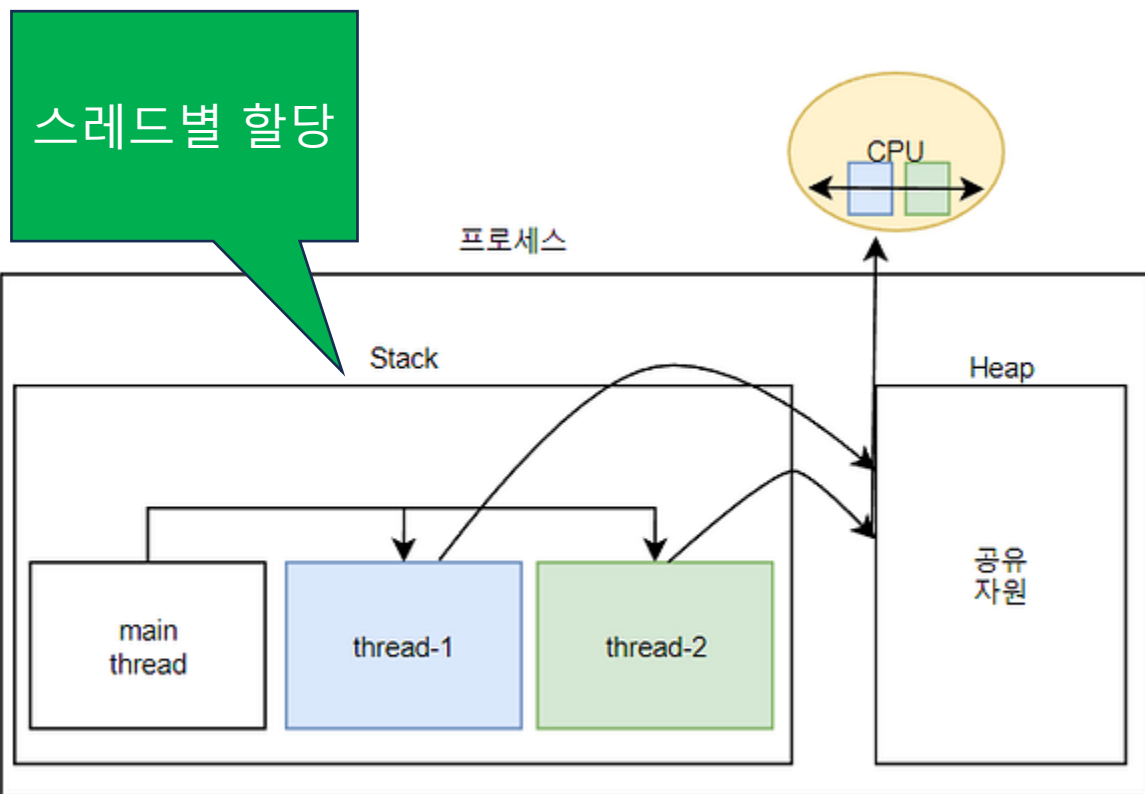
- 지정한 풀 사이즈에 따라 **5개의 요청**이 스레드에 배정됨
- 5개의 요청이 '**동시에**' 재고를 확인함
- 스레드 요청 '**순서에 상관없이**' 재고를 확인 및 처리

org.opentest4j.AssertionFailedError:
Expecting:
<6>
to be equal to:
<0>
but was not.
Expected :0
Actual :6

2/7 스레드 할당

1. 문제 상황 – 왜 이런 문제가 발생한 걸까?

멀티 스레드와 동시성



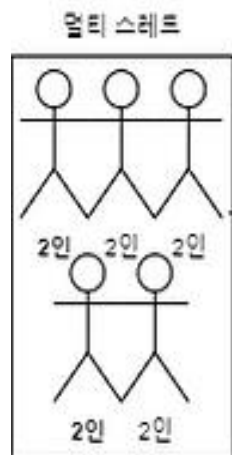
•멀티 스레드 :

한 프로세스에서 여러 개의 스레드가 실행되는 방식.
스레드는 자신이 속한 프로세스의 메모리를 '**공유**'

•동시성 :

하나의 코어에서 멀티 스레드가 '**번갈아**' 실행하는
성질

1. 문제 상황 – 왜 이런 문제가 발생한 걸까?



```
int curReqPpl = histRequestDto.getReqPpl();  
// ** 호스트 재고 확인  
int curLvtPplByHost = hostRepository.countLvtPpl(host.get().getHnum());  
log.info("현재 호스트 예약 재고 : {}", curLvtPplByHost);  
  
if (curReqPpl > curLvtPplByHost) {  
    // 인원이 많으면 실패해야함  
    log.info("재고 부족으로 예약 실패");  
    throw new Exception("재고인원 부족");  
}  
else {  
    // 호스트에서 재고 - reqPpl TODO MSA 고민  
    host.get().updateLvtPpl(histRequestDto.getReqPpl());  
    hostRepository.save(host.get());  
  
    // 예약 완료  
    histRequestDto.setUserid(userId);  
    resrvHisRepository.save(histRequestDto.toEntity());  
}
```

스레드가 함께 공유하는 부분

아직 재고의 변경이 생기지 않았는데
다른 스레드가 공유자원에 접근

고객2 고객1

먼저 실행된 스레드가
데이터를 변경 커밋하기 이전에
다른 스레드들이 메모리 **공유 자원**에 접근

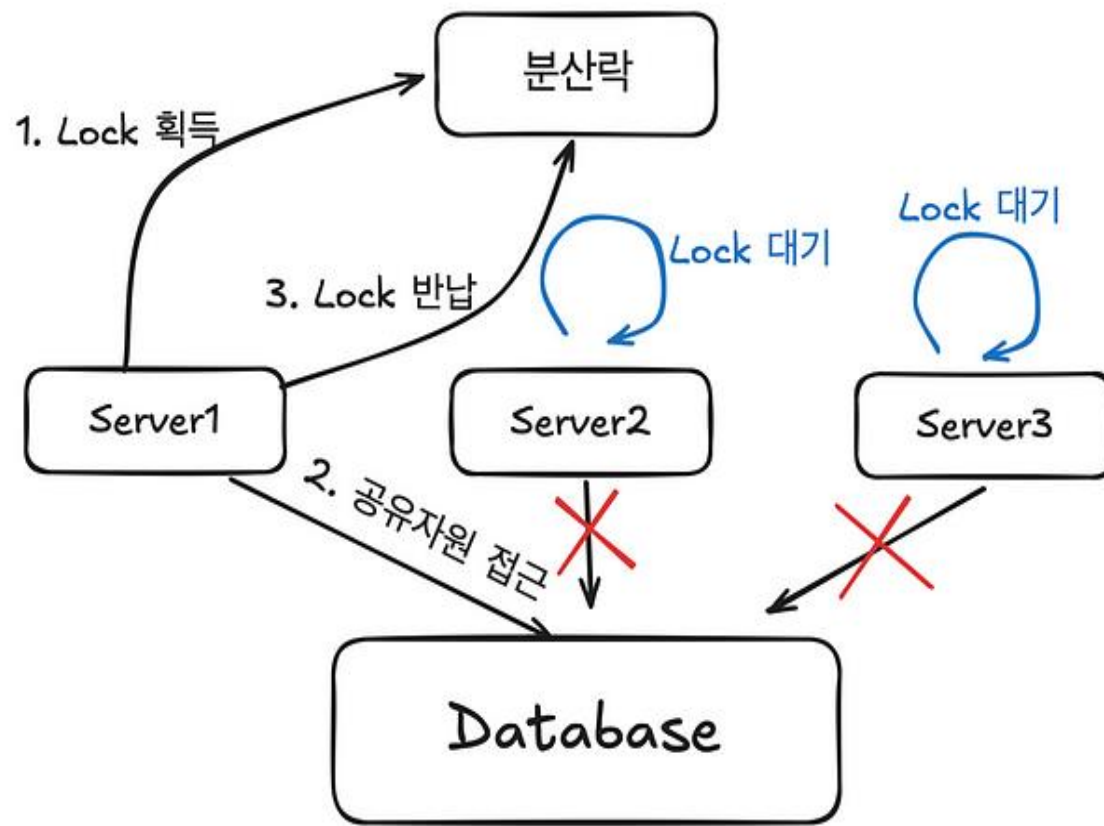
2. 해결방법 - 동시성 문제

Redisson 분산락

- 임계 구역에 락을 걸음 (!= 공유자원)
- 실제로 작업을 처리하는 부분이 '**싱글 스레드**' 로 동작하여 원자성 보장 (레디스의 원리)
- 스케일 아웃 된 DB(=분산DB) 환경에서도 동시성 제어가 가능함



- ✓ 공유 자원을 Redis 서버를 사용
- ✓ MSA 환경에서도 동시성 이슈 제어가 가능함
- ✓ DB의 동기화를 기다리지 않아도 됨



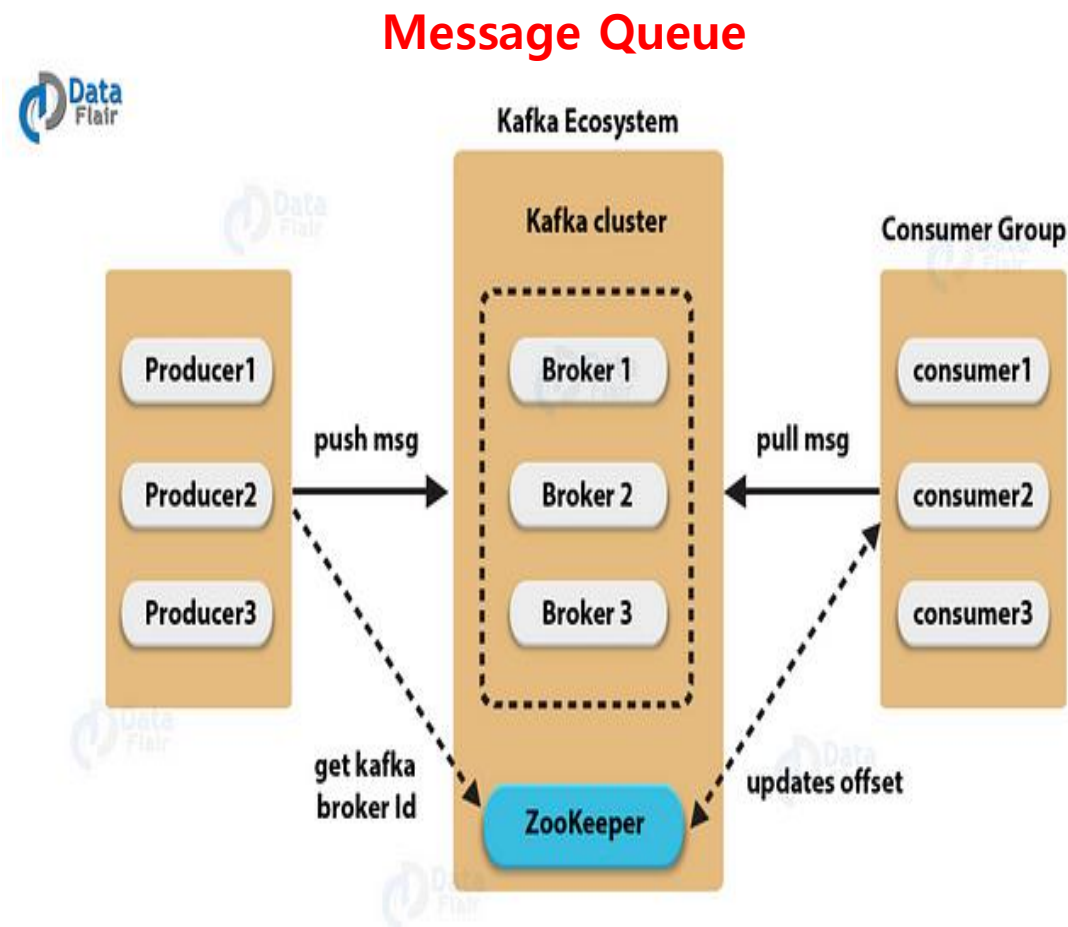
2. 해결방법 - 성능문제

카프카 - Produce / Consume

- 분산 메세징 시스템
- 메시지를 파일 시스템에 저장하여 영속성 보장
- Consumer가 broker로부터 '직접' 메시지를 가지고 옴 pull
- Producer 중심 형태로 '많은 양'의 데이터를 파티셔닝 하는데 초점



- ✓ Consumer 실패시 데이터 유실되면 안됨
- ✓ 데이터의 ETL은 크게 필요 없음
- ✓ '비동기'의 초점, 대용량 트래픽을 대비



2. 해결방법 - 디스크 I/O 문제

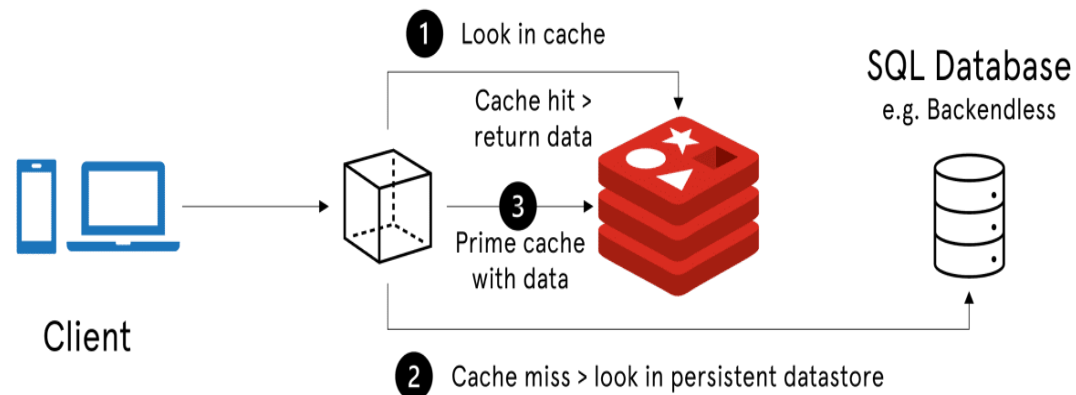
Redis - 분산 캐싱

- 다양한 데이터 구조를 지원
- 데이터의 영속성을 제공하며 Pub/Sub 및 트랜잭션 지원
- 싱글 스레드 기반이며 메모리 사용량이 상대적으로 높음

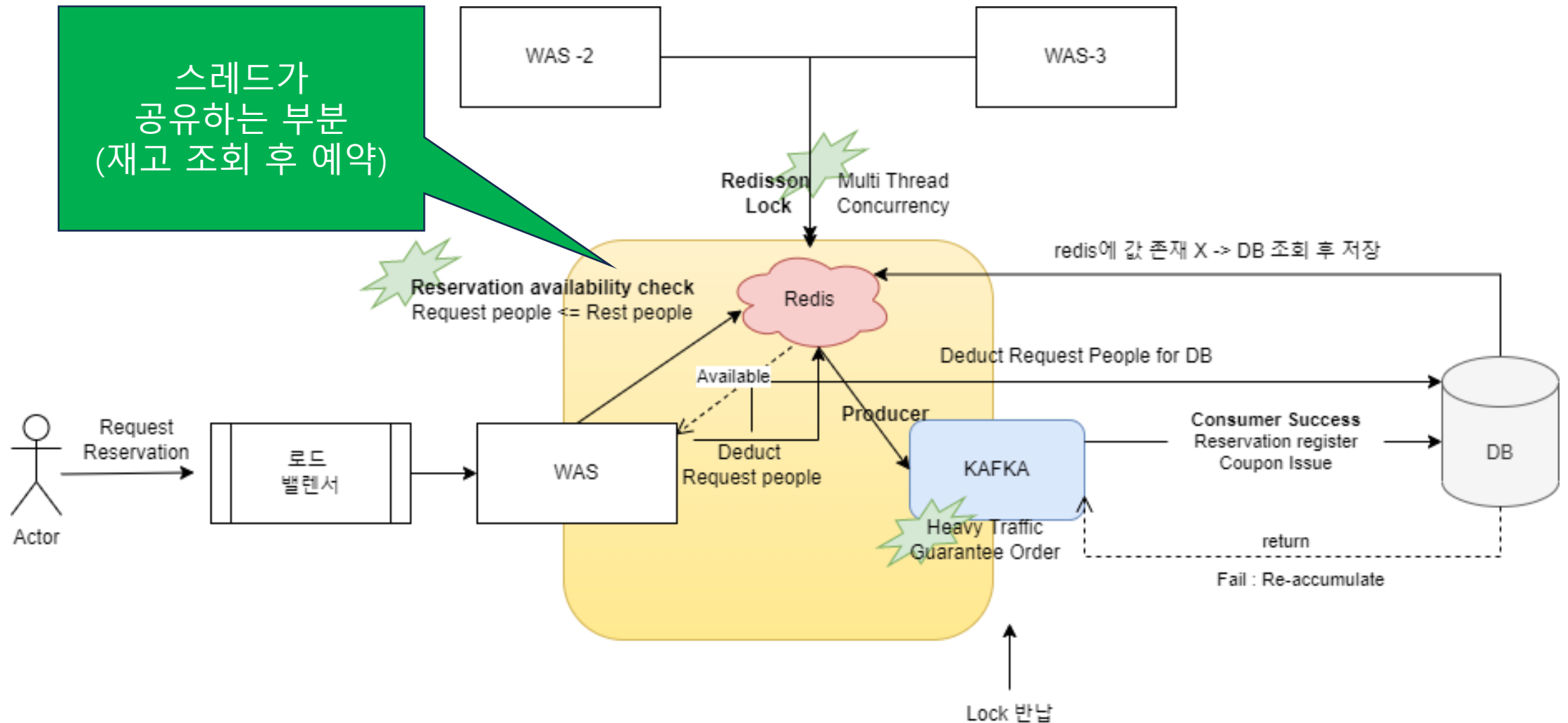


- ✓ 인 메모리 캐시의 경우 상품의 개수가 많으면 힙 메모리 차지 큼
- ✓ 분산 DB 를 고려하고 만든 아키텍처 이므로 분산 캐시 서버 필요함
- ✓ 적절한 캐시 히트 전략 필요

How Redis is typically used



3. 개선된 서버 - 최종 아키텍처 모델



3. 개선된 서버 - 최종 아키텍처 모델

결과

```
ce.resrv.RedisService : 현재 호스트 db 예약 재고 : 10
ce.resrv.RedisService : redis 예약가능 인원 차감 HostNum:1 : 10 ->8
created_2_2_0_, host0_.modified_date as modified3_2_0_, host0_.address as
6_2_0_, host0_.apprv_date as appr_vda7_2_0_, host0_.farmsts as farmsts8_2_0_,
.ivt_ppl as ivt_ppl11_2_0_, host0_.max_ppl as max_ppl12_2_0_, host0_.region as
15_2_0_ from host host0_ where host0_.hnum=?
ted_2_8_0_, user0_.modified_date as modified3_8_0_, user0_.auth_provider as
_0_, user0_.password as password7_8_0_, user0_.role as role8_8_0_ from user user0_
?, age=?, appr_vyn=?, appr_vdate=?, farmsts=?, gender=?, intro=?, ivt_ppl=?,
ce.resrv.ResrvServiceImpl : host 예약 가능인원 차감 : 8
```

캐시 전략 => Look
Aside / Write Through

```
service.kafka.KafkaConsumer : 카프카 소비 예약 히스토리 적재 : com.example.springboot
date as created_2_0_0_, cpn0_.modified_date as modified3_0_0_, cpn0_.ivt_cnt as
reg_dt6_0_0_, cpn0_.reg_id as reg_id7_0_0_ from cpn cpn0_ where cpn0_.cpn_num=?
service.resrv.ResrvServiceImpl : 현재 호스트 redis 예약 재고 : 8
where cpnissu0_.cpn_num=? and cpnissu0_.userid=?
service.kafka.KafkaConsumer : 사용자 쿠폰 확인 조회 : 0
date as created_2_0_0_, cpn0_.modified_date as modified3_0_0_, cpn0_.ivt_cnt as
reg_dt6_0_0_, cpn0_.reg_id as reg_id7_0_0_ from cpn cpn0_ where cpn0_.cpn_num=?
service.redis.RedisService : redis 예약가능 인원 차감 HostNum:1 : 8 ->6
e as created_2_2_0_, host0_.modified_date as modified3_2_0_, host0_.address as
rv_vyn6_2_0_, host0_.apprv_date as appr_vda7_2_0_, host0_.farmsts as farmsts8_2_0_,
ost0_.ivt_ppl as ivt_ppl11_2_0_, host0_.max_ppl as max_ppl12_2_0_, host0_.region as
as id15_2_0_ from host host0_ where host0_.hnum=?
cnt=?, max_cnt=?, reg_dt=?, reg_id=? where cpn_num=?
created_2_8_0_, user0_.modified_date as modified3_8_0_, user0_.auth_provider as
ae6_8_0_, user0_.password as password7_8_0_, user0_.role as role8_8_0_ from user user0_
service.kafka.KafkaConsumer : 쿠폰 재고 차감 후 현재 재고: 9
ress=?, age=?, appr_vyn=?, appr_vdate=?, farmsts=?, gender=?, intro=?, ivt_ppl=?,
service.resrv.RedisService : DB 차감 write through 예약 가능인원 차감 : 6
```

3. 개선된 서버 – 과연 최선일까

분산락 대신 비관락?



- ✓ DB의 락을 이용하는 방법 (버전 체크 후 롤백 try catch)
- ✓ 새로운 트랜잭션 생성 후 propagation의 수준을 정하는 방법

처음부터
동시성 문제가
없도록 만들어라

캐시 서버의 필요성?



- ✓ DB 서버도 메모리에서 값을 가져오기 때문에 매번 디스크 I/O가 필요한 것은 아님
- ✓ DB 서버 1대
- ✓ 트래픽의 양이 성능에 크게 영향을 줄 만큼 크지 않다면
- ▶ 네트워크 연결에 대한 비용이 더 클 수 있음

조사를 진행하며 배운 것들

- 트랜잭션과 락의 중요성
 - 피할 수 없는 공유 자원.....어떻게 관리할 것인가? -> Lock
 - 실패할 경우 어디까지 정보를 가지고 있을 것인가 -> Transactional
- 아키텍처는 도메인의 성격, 요구 조건을 고려
 - 트래픽, 업무에 성격에 따라 필요한 기능이 있음
 - "필요성" 에 대한 고민이 우선해야

무엇이 문제인가, 어떻게 해결 것인가, 이것이 최선인가

최적화를 할 때는 다음 두 규칙을 따르라.

첫 번째, **하지 마라.**

두 번째, (전문가 한정) **아직 하지 마라.**

- M. A. 잭슨

(컴퓨터 과학자, JSP 창시자)



고생하셨습니다