



# 가상면접 사례로 배우는 대규모 시스템 설계 기초



PPT by 김주혁

# 목차

---

- 01 1단계 문제 이해 및 설계 범위  
확정
- 02 2단계 개략적인 설계안 제시  
및 동의 구하기
- 03 3단계 상세 설계
- 04 4단계 마무리

4장  
처리율 제한 장치의 설계

처리율 제한 장치...?

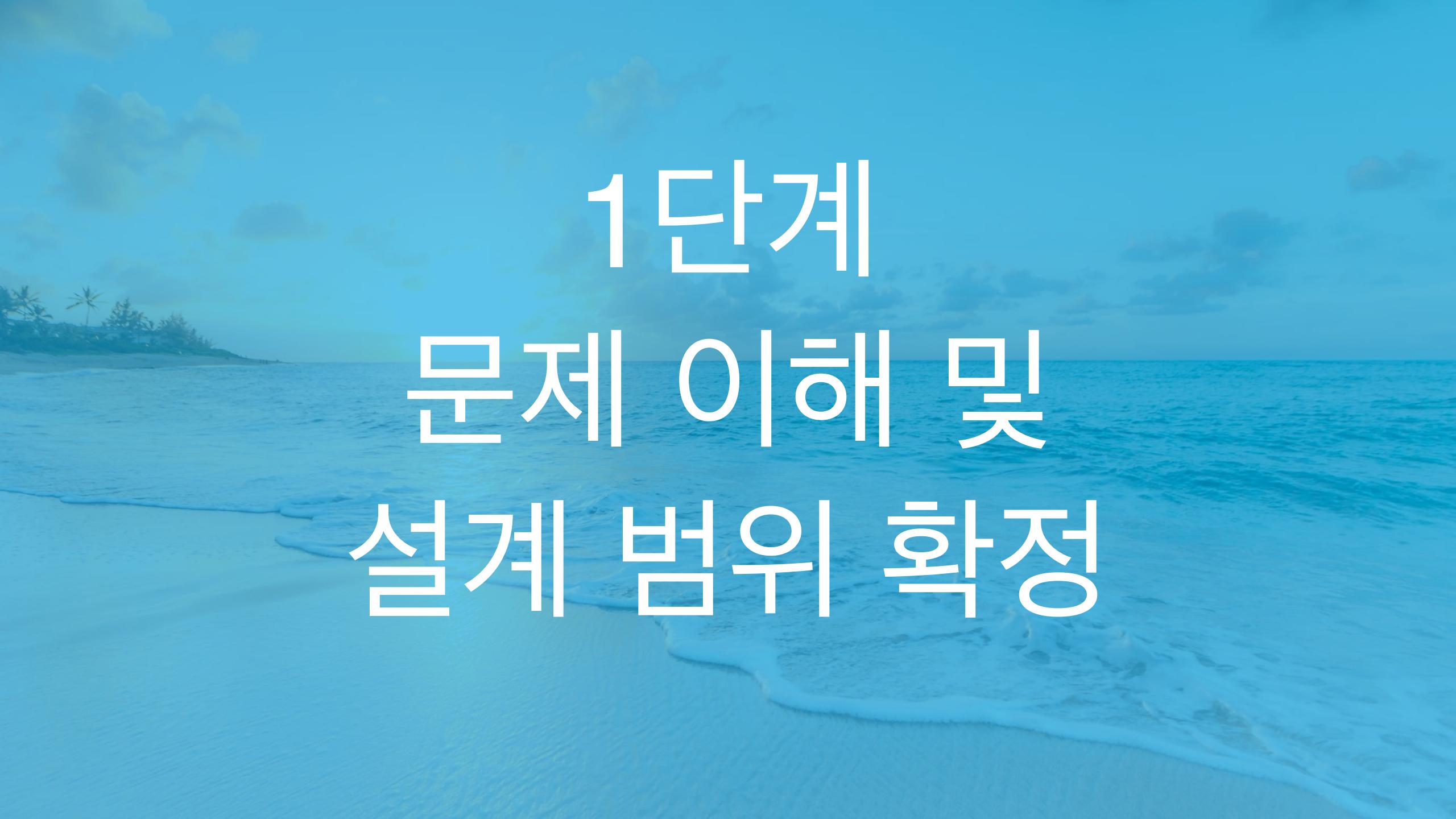




“

*DDoS*

”

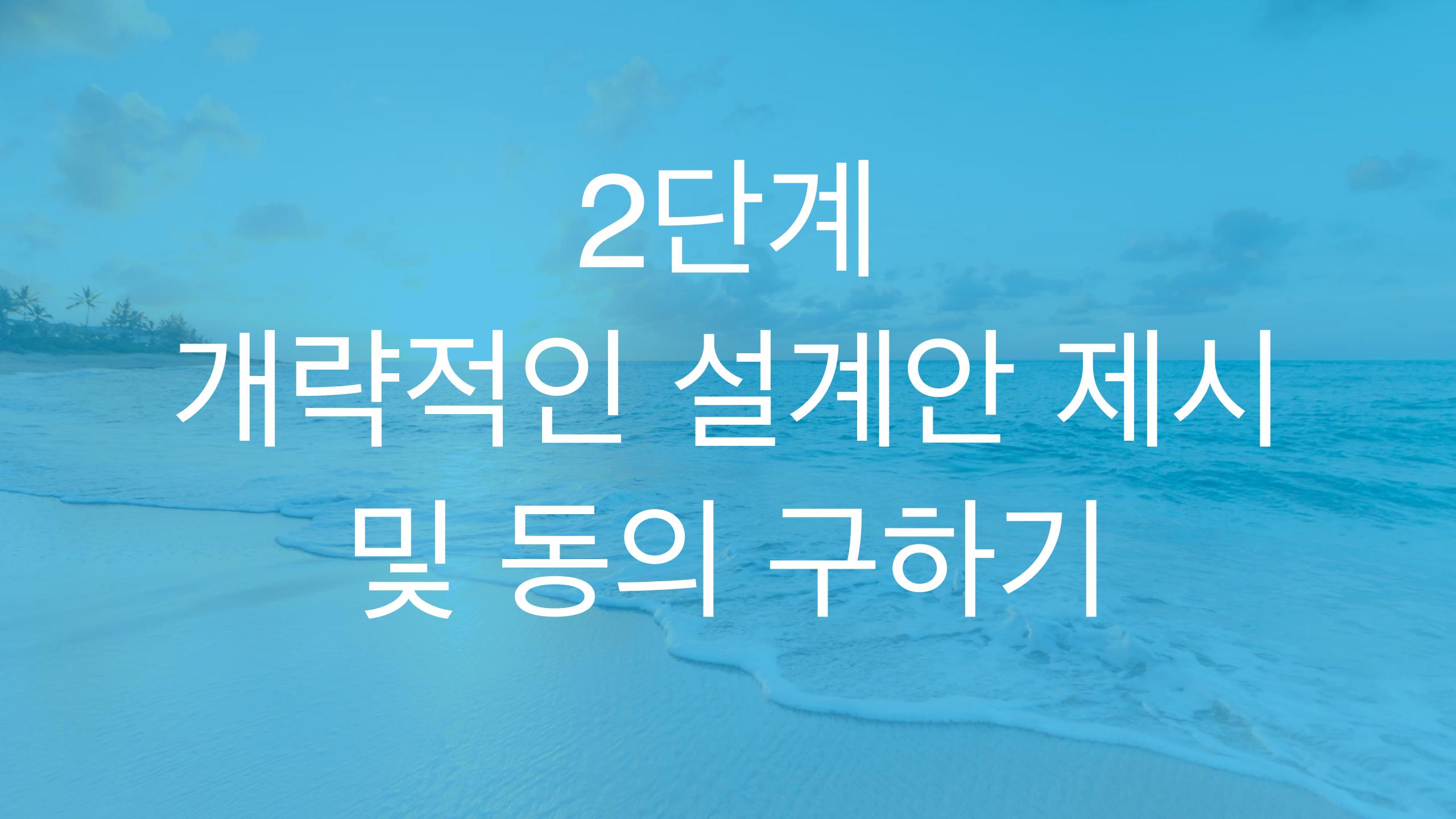


# 1단계 문제 이해 및 설계 범위 확정

- Q: 어떤 종류의 처리율 제한 장치를 설계해야 합니까? 어느 사이드에서 사용할 제한 장치입니까?  
A: 서버 API를 위한 제한 장치
- Q: 어떤 기준을 사용해서 제어해야 합니까?  
A: 다양한 형태의 제어 규칙을 정회할 수 있는 유연한 시스템
- Q: 시스템 규모는?  
A: 대규모
- Q: 분산 환경에서 작동합니까?  
A: 그렇다.
- Q: 독립 서비스입니까, Application의 코드 레벨입니까?  
A: 면접자의 선택의 영역
- Q: 사용자가 장치에 의해 필터링된다면 Notification이 있어야 합니까?  
A: 그렇다.

#### 질의 과정을 통한 요구사항

- 설정된 처리율을 초과하는 요청은 정확하게 제한한다.
- 낮은 응답시간(Low Latency): 이 장치가 HTTP 응답 시간에 끼치는 악 영향은 최대한 적어야 한다.
- 가능한 한 적은 메모리를 사용해야 한다.
- 분산형 처리율 제한: 하나의 처리율 제한 장치를 여러 서버나 프로세스가 공유할 수 있어야 한다.
- 예외 처리: 요청이 제한될 때는 사용자에게 분명한 알림이 있어야 한다.
- 높은 결합 감내성: 제한 장치에 장애가 생기더라도 전체 시스템은 정상 동작해야 한다.

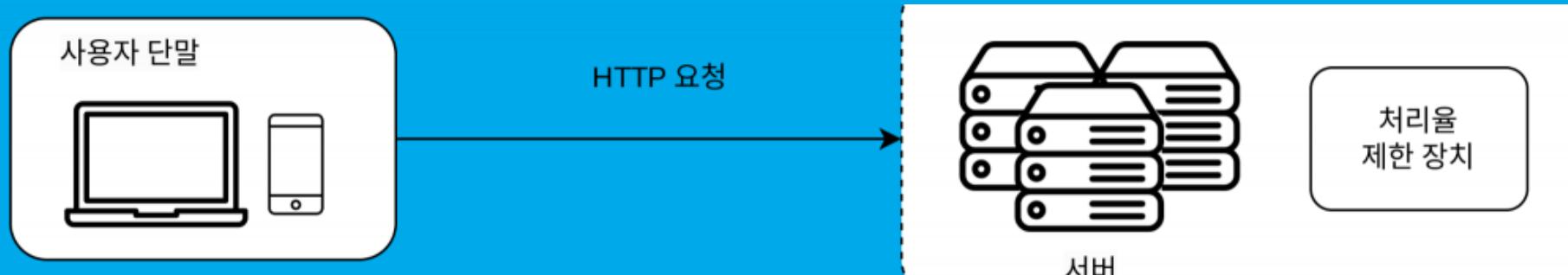


# 2단계 개략적인 설계안 제시 및 동의 구하기

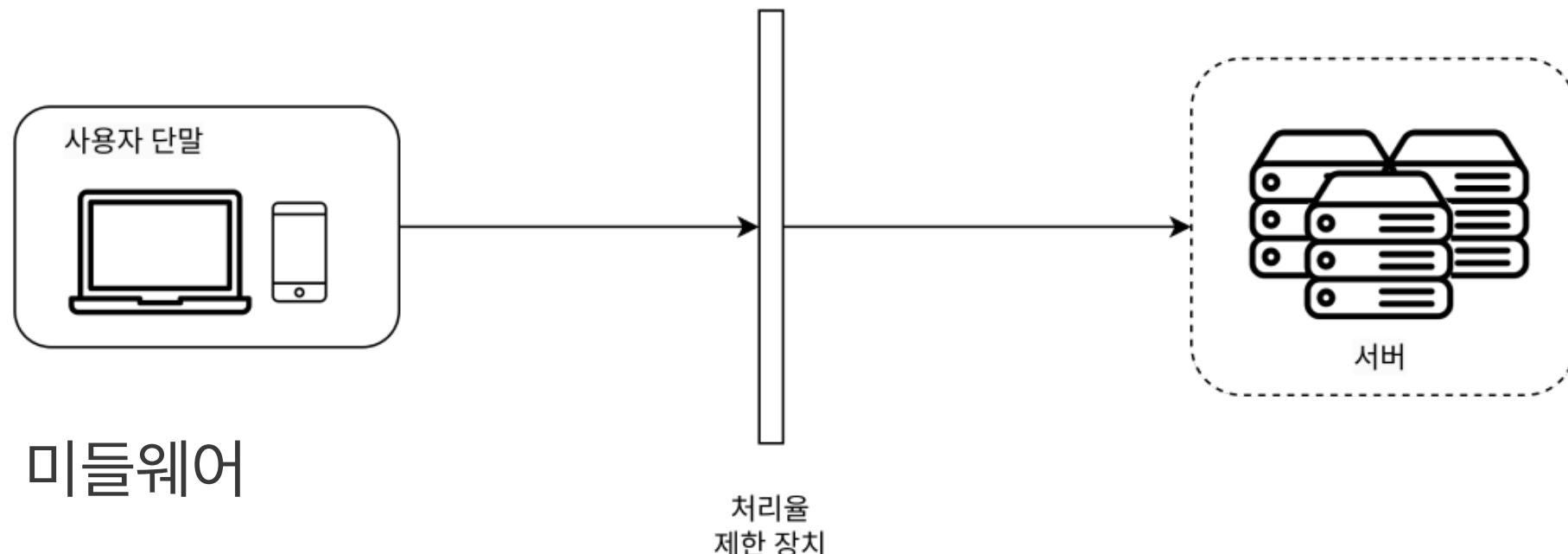


처리율 제한 장치는 어디에 두어야 하는가?





서버

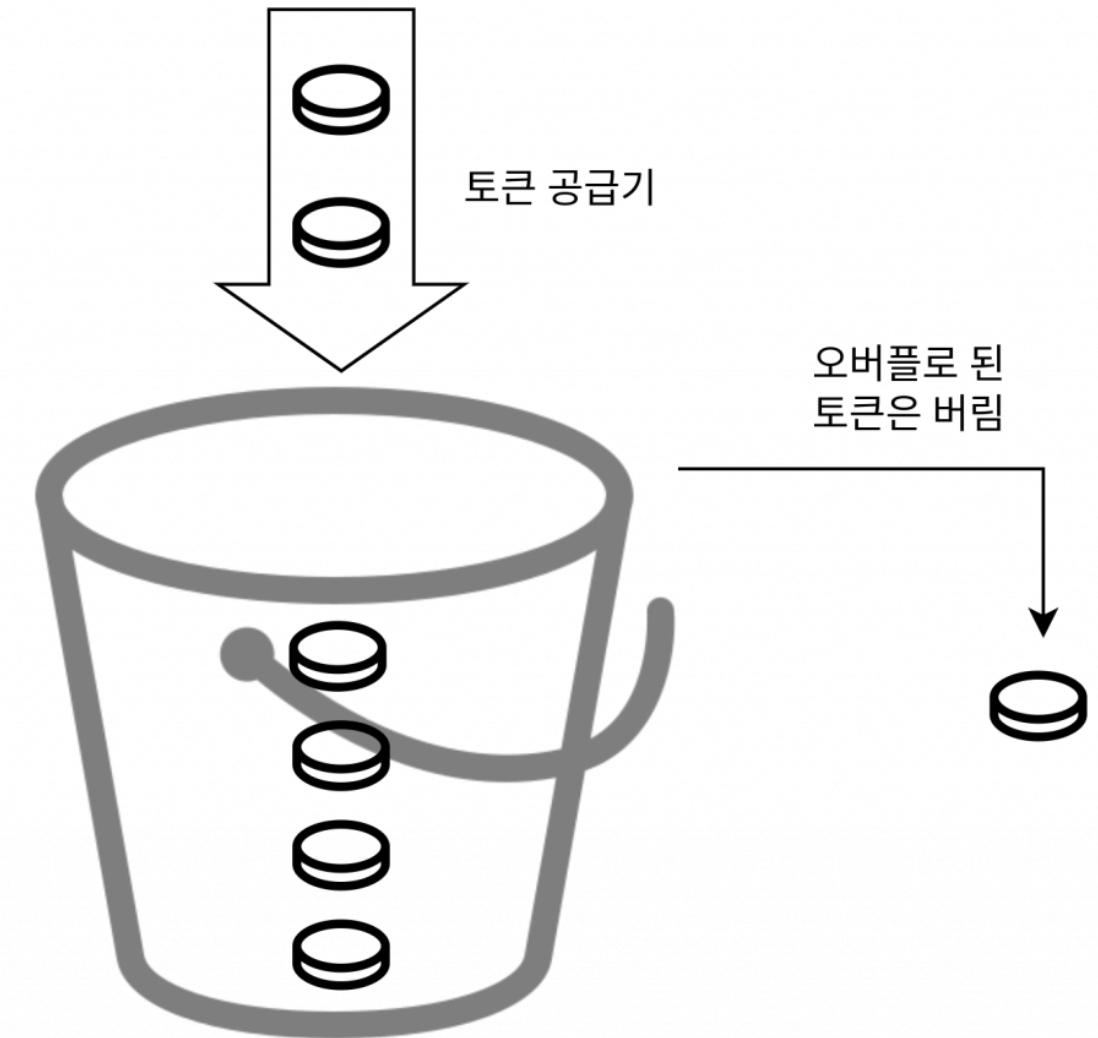


미들웨어

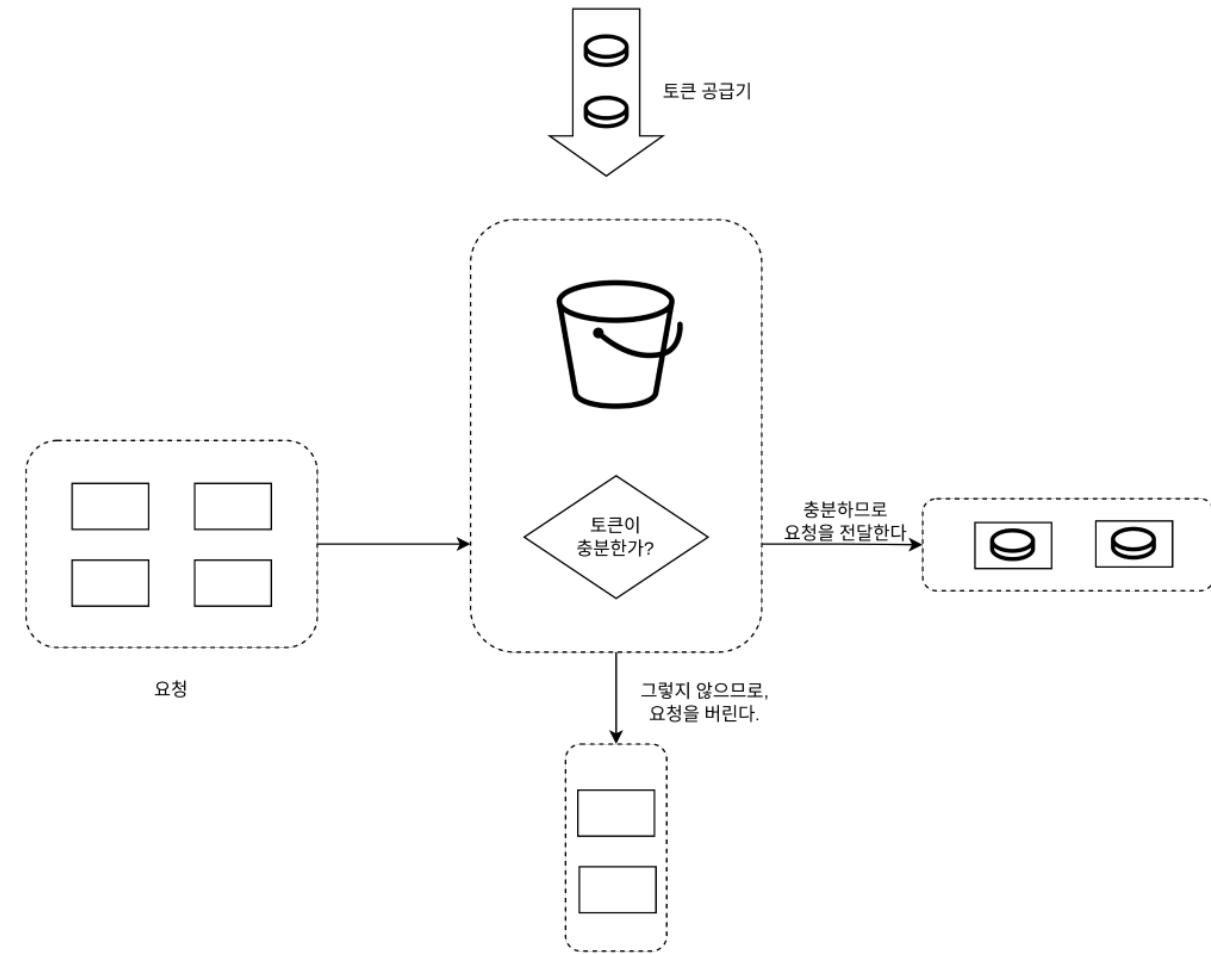
처리율  
제한 장치

# 토큰 버킷 알고리즘

# 토큰 버킷 알고리즘



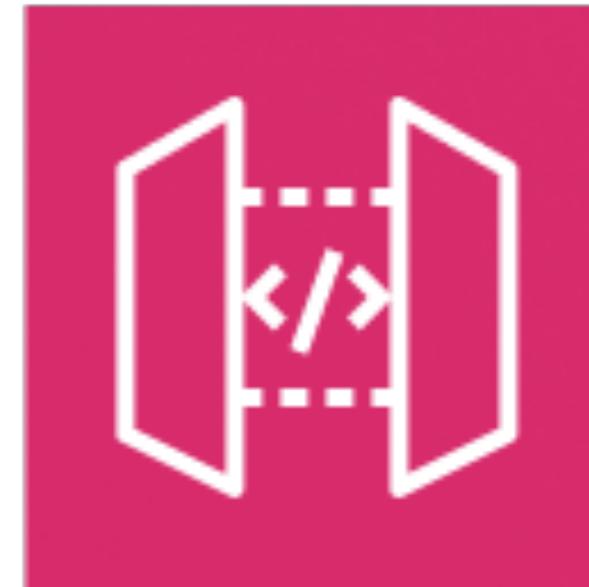
# 토큰 버킷 알고리즘



# 토큰 버킷 알고리즘

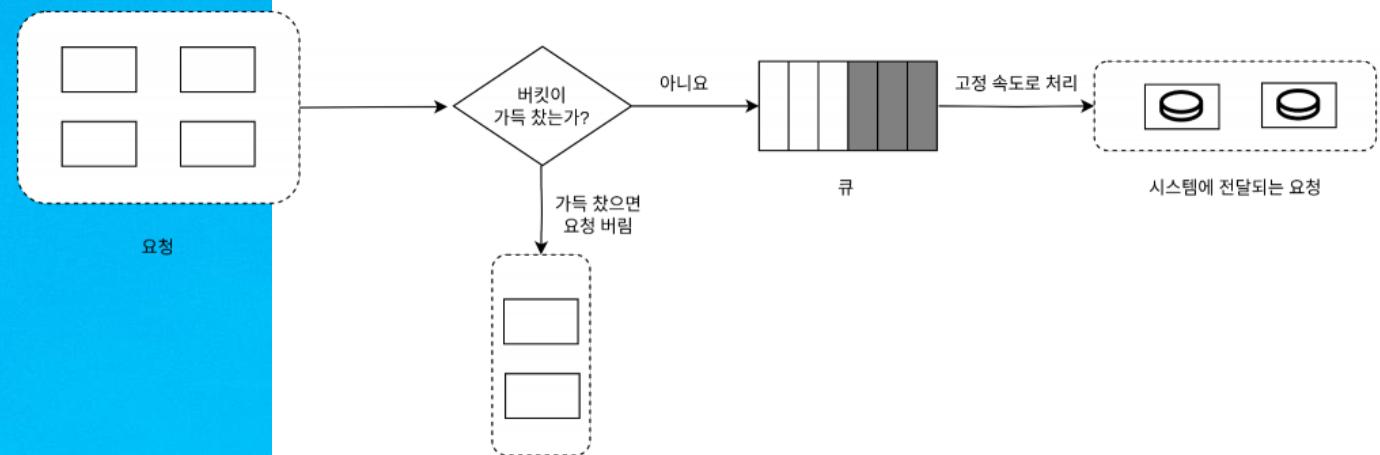


AWS API Gateway



# 누출 버킷

# 누출 버킷 알고리즘



# 쇼피파이 API별 비율 제한 비교(Compare rate limits by API)

## API별 비율 제한 비교

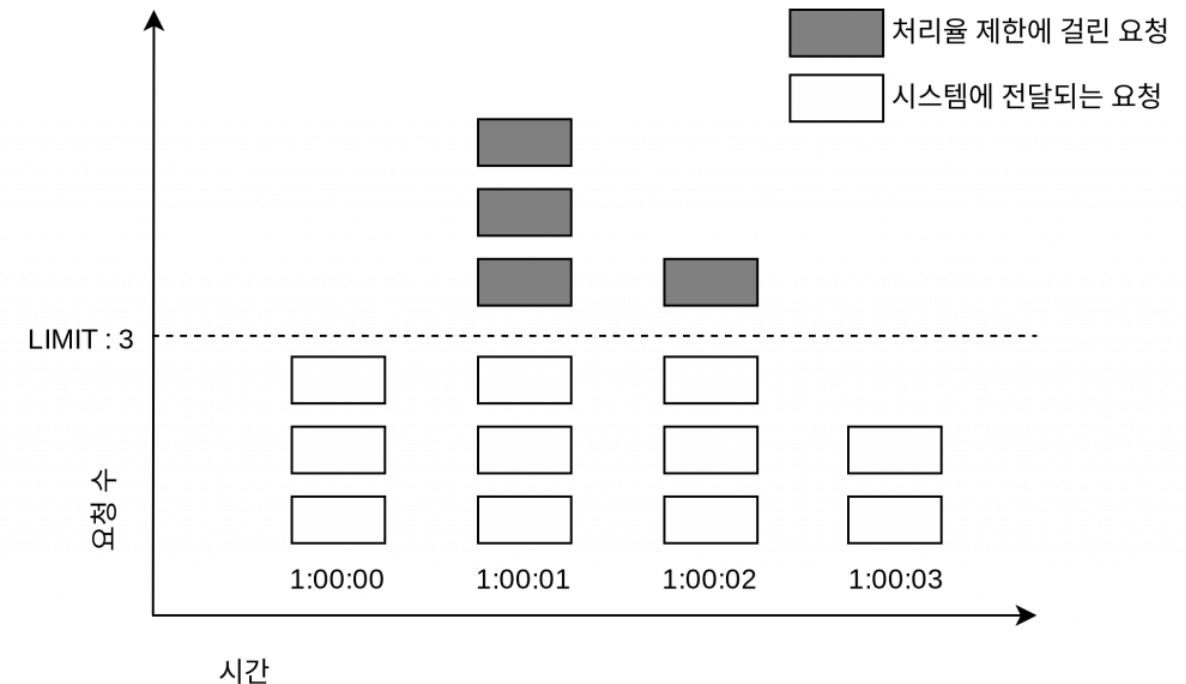
Shopify API는 다양한 속도 제한 방법을 사용합니다. 아래에 더 자세히 설명되어 있지만 주요 수치를 간단히 요약하면 다음과 같습니다.

API	속도 제한 방법	기준한도	고급 Shopify 한도	Shopify Plus 한도	Shopify 한도별 상거래 구성요소
관리 API ( <a href="#">GraphQL</a> )	계산된 쿼리 비용	100포인트/초	200포인트/초	1000포인트/초	없음
관리 API ( <a href="#">REST</a> )	요청 기반 한도	요청 2개/초	요청 4개/초	요청 20개/초	없음
매장 API	없음	없음	없음	없음	없음
결제 앱 API ( <a href="#">GraphQL</a> )	계산된 쿼리 비용	910포인트/초	910포인트/초	1820포인트/초	없음
고객 계정 API	계산된 쿼리 비용	100포인트/초	200포인트/초	200포인트/초	없음

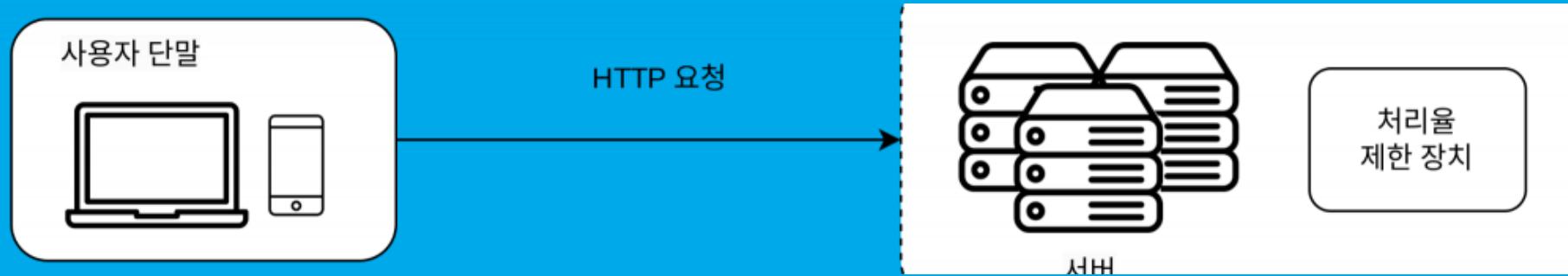
Shopify는 플랫폼 안정성을 보호하기 위해 일시적으로 API 속도 제한을 줄일 수 있습니다. 우리는 이러한 사례를 간단하고 드물게 유지하기 위해 노력할 것입니다. 그러나 제한을 적절하게 처리하도록 애플리케이션을 구축해야 합니다.

# 고정 윈도 카운터 알고리즘

# 고정 윈도 카운터 알고리즘



# 이동 윈도 로깅 알고리즘



이동 원도 로깅 알고리즘

# 이동 윈도 카운터 알고리즘

# 이동 윈도 카운터 알고리즘



그리슬  
보관하기 때문입니다.

한국기독교  
그리스

한국서

다른 하니  
-○이 알고리즘의 풍자  
현재 시간

처리율 한도: 분당 5개 요청

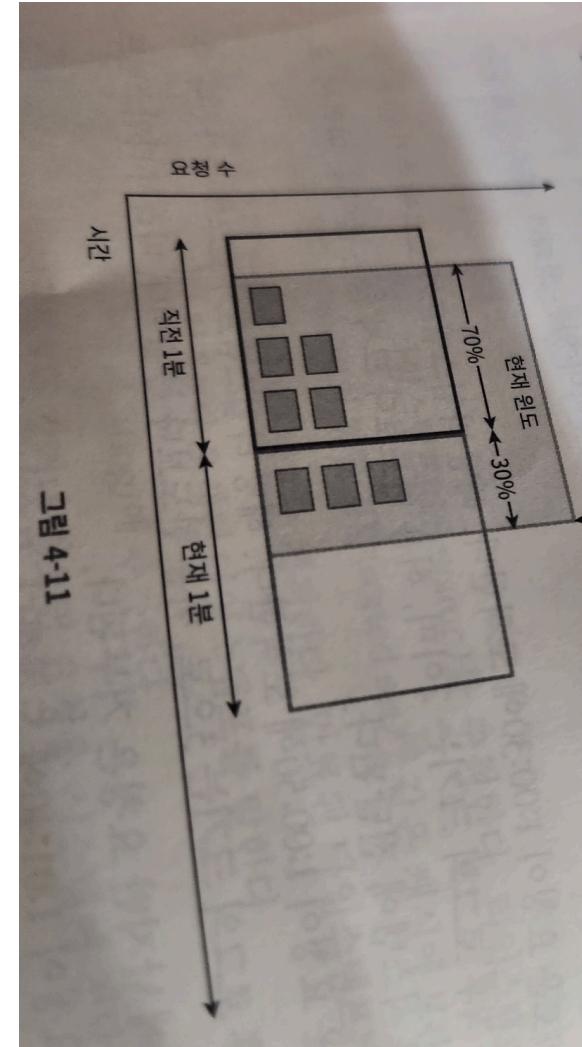
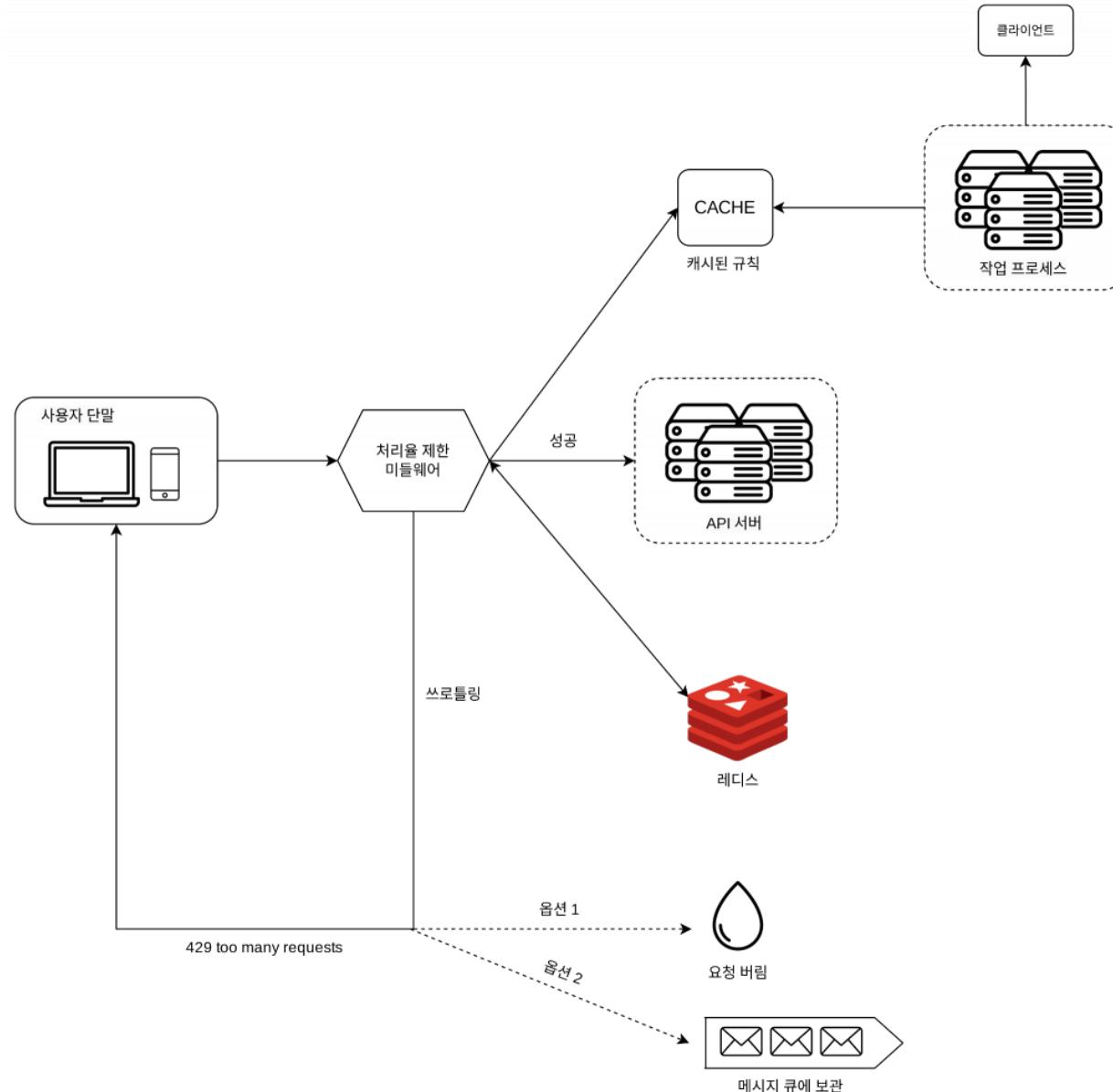


그림 4-11

문제 출제 및  
문제 해결

결제 작업

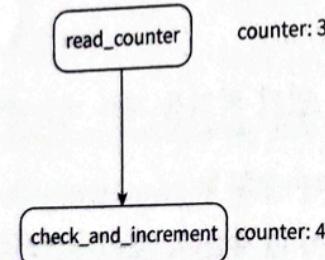
# 3단계 상세 설계



# 경쟁 조건

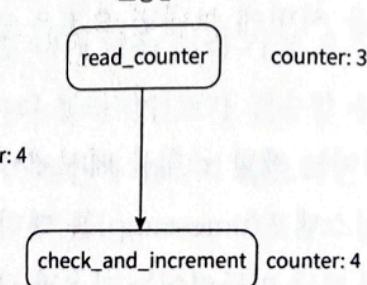
counter의 원래 값: 3

요청1



counter: 3

요청 2



counter: 3

counter의 값은 원래  
5가 되어야 함

그림 4-14

# 동기화

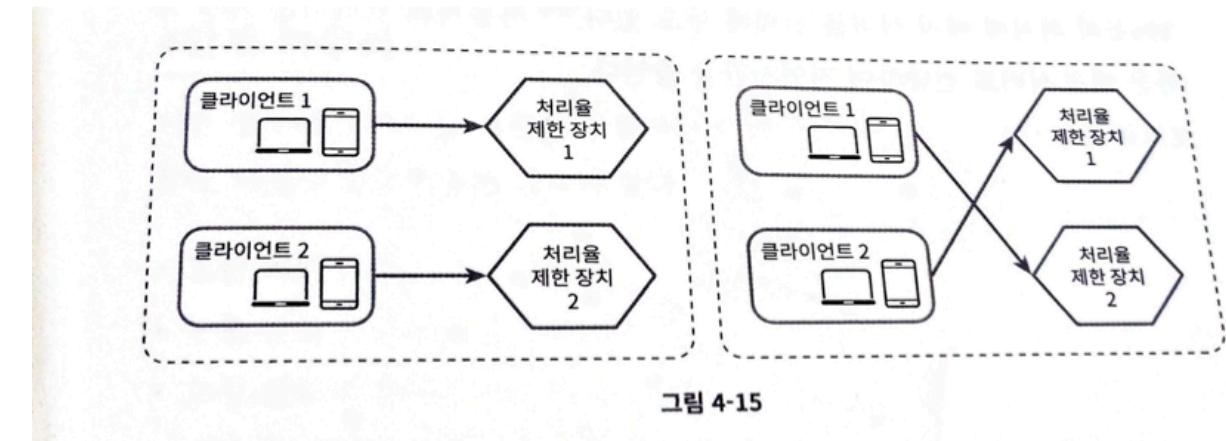


그림 4-15

# 동기화



System Design

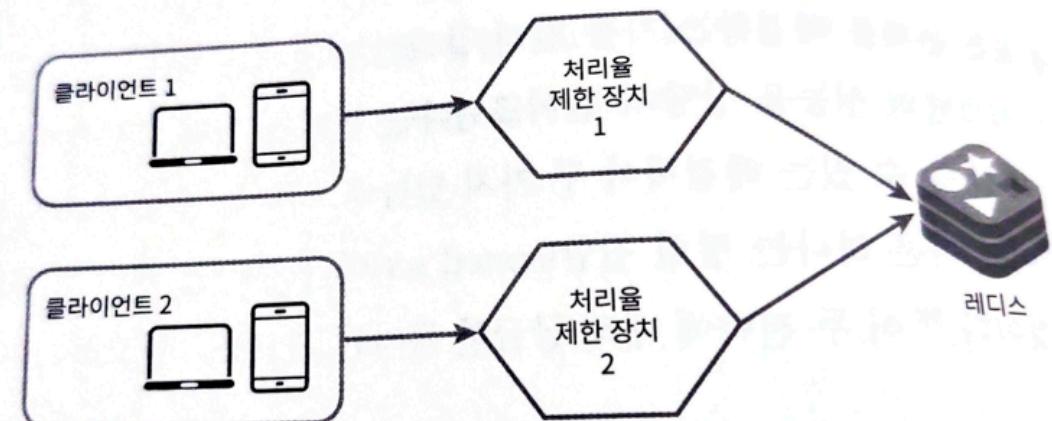


그림 4-16

# 4단계 마무리

다음의 질문은 도움이 될 수 있다.

- 경성 또는 연성 처리율 제한
  - 경성 처리율 제한 : 요청의 개수는 임계치를 절대 넘어설 수 없다.
  - 연성 처리율 제한 : 요청 개수는 잠시 동안은 임계치를 넘어설 수 있다.
- 처리율 제한을 회피하는 방법, 클라이언트를 어떻게 설계해야 하는가?
  - 클라이언트 측 캐시를 사용하여 API 횟수를 줄인다.
  - 처리율 제한의 임계치를 이해하고 단기간에 너무 많은 요청이나 메세지를 보내지 않도록 한다.
  - 예외나 에러를 도입하여 예외상황으로부터 우아하게(gracefully) 복구될 수 있도록 한다.
  - 재시도 로직은 충분한 백오프 시간을 둔다.

## 마치며.. 시스템 설계에 대한 사담..

Q : 캐시서버에서 유저에게 보여줄 문제 리스트의 기준은 어떻게 정해집니까? 이 기준이야말로 이 서비스에서 가장 중요한 정보입니다.

A : ... 캐시 서버를 없앨까요?

Q : 유저들마다 적용되는 기준은 어떻게 반영하실 건가요? 해당 사안은 특정 유저를 고려하지 않은 서비스 전반적인 기준인 것 같습니다.

A : ... 캐시서버를 두지 않고, 코드 레벨에서 비즈니스로직으로 처리하는 것이 옳은 것 같습니다.

출처



<https://jonghoonpark.com/2023/05/17/%EC%B2%98%EB%A6%AC%EC%9C%A8-%EC%A0%9C%ED%95%9C-%EC%9E%A5%EC%B9%98-%EC%84%A4%EA%B3%84>

<https://shopify.dev/docs/api/usage/rate-limits#compare-rate-limits-by-api>