처리율 제한 장치 설계

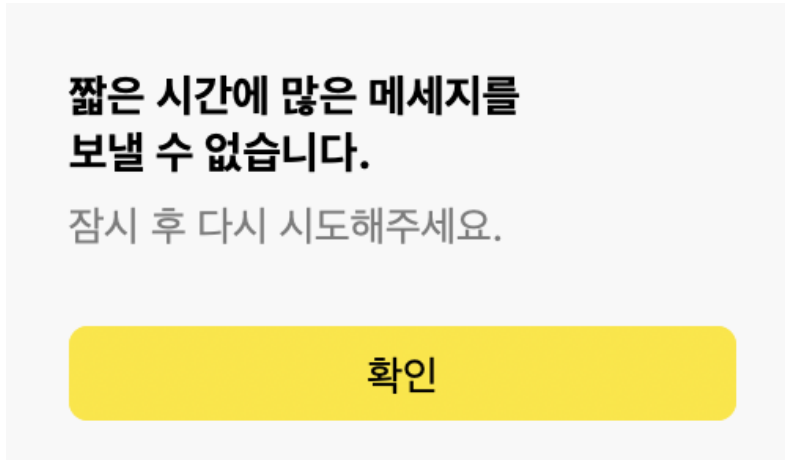
2025. 04. 08

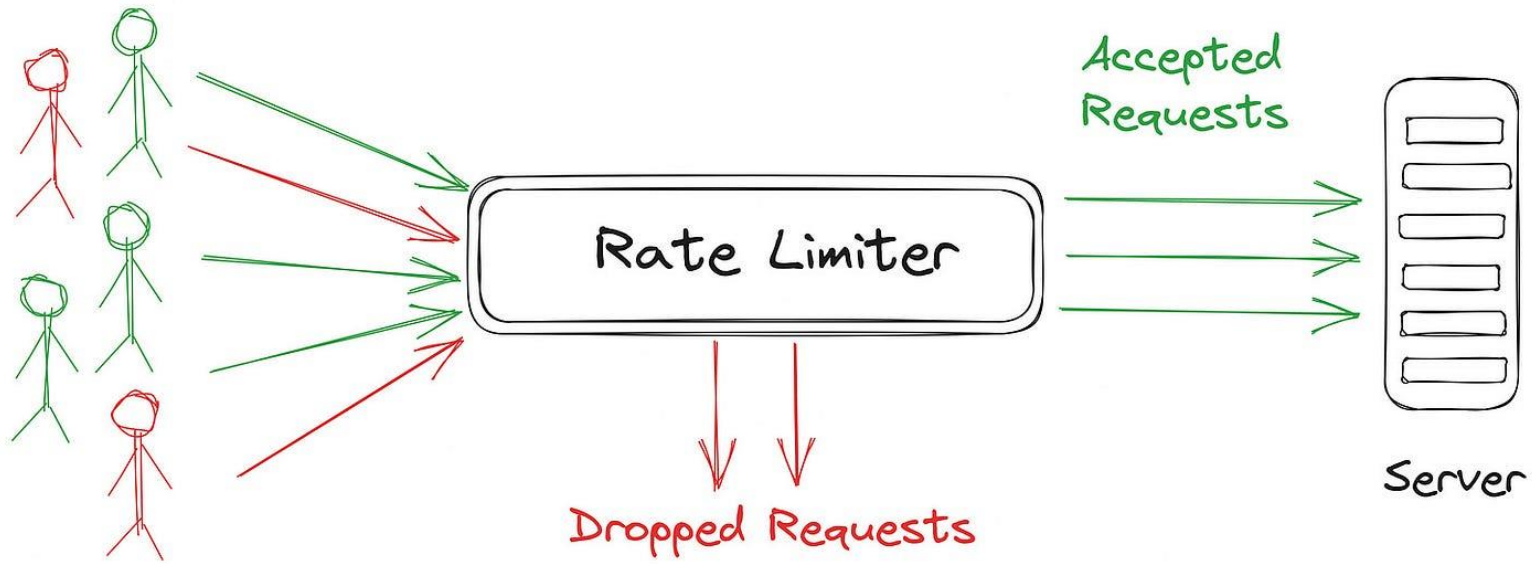
황지연

01

처리율 제한 장치란

1. 처리율 제한 장치 (Rate Limiter)란?





1. 처리율 제한 장치 (Rate Limiter)란?

네트워크 시스템에서 처리율 제한 장치란?

- 클라이언트 또는 서비스가 보내는 트래픽의 처리율을 제어하기 위한 장치

HTTP에서의 처리율 제한 장치

- 특정 기간 내 전송되는 클라이언트의 요청 횟수를 제한
- API 요청 횟수가 제한 장치에 정의된 임계치(threshold)를 넘어서면 추가로 도달한 모든 호출은 Block

Some Examples

- 사용자는 초당 2회 이상 새 글을 올릴 수 없다.
- 같은 IP 주소로는 하루에 10개 이상의 계정을 생성할 수 없다.
- 같은 디바이스로는 주당 5회 이상의 reward를 요청할 수 없다.

02

처리율 제한 장치를 두면 좋은 점

2. 처리율 제한 장치를 두면 좋은 점

1. DoS 공격에 의한 resource 고갈을 방지

2. 비용 절감

3. 서버 과부화 방지
    - Bot이나 사용자의 잘못된 이용 패턴으로 유발된 트래픽을 걸러낼 수 있음

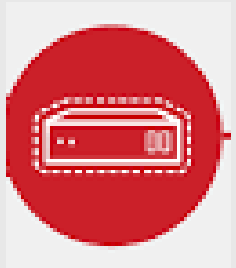
03

처리율 제한 장치를 어디에 두어야 하는가?

3. 처리율 제한 장치를 어디에 두어야 하는가?

client

server



middleware

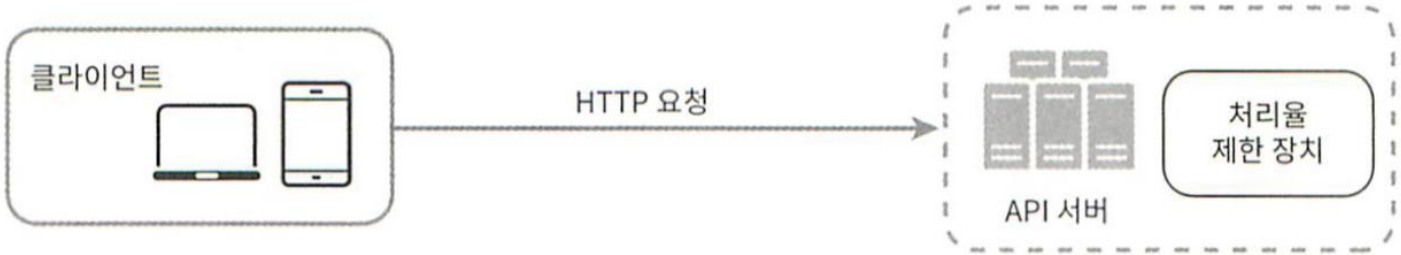
운영 체제와 응용 소프트웨어의 중간에서
조정과 중개의 역할을 수행하는 소프트웨어

3. 처리율 제한 장치를 어디에 두어야 하는가?

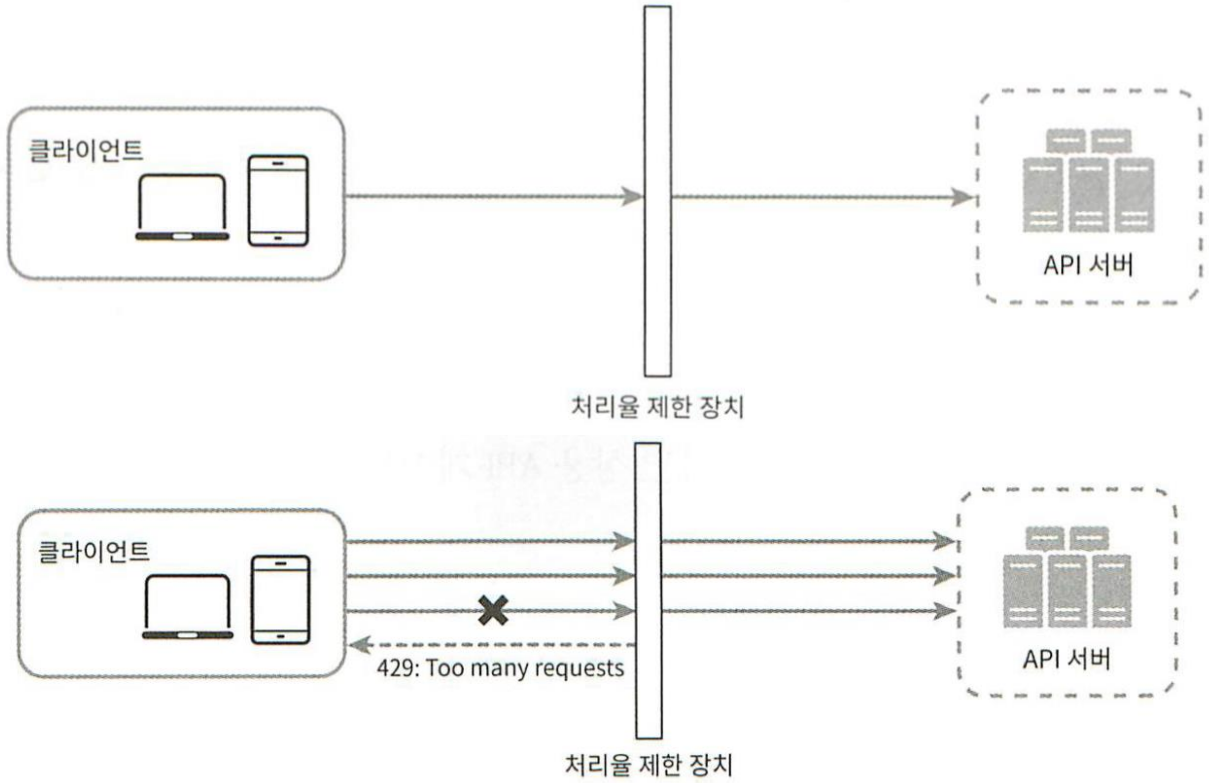
client

- 일반적으로 클라이언트는 처리율 제한을 안정적으로 걸 수 있는 장소가 아님
  - ➢ 클라이언트 요청은 쉽게 위변조가 가능
  - ➢ 모든 클라이언트를 통제하는 것이 사실상 어렵기 때문

server



3. 처리율 제한 장치를 어디에 두어야 하는가?



- 폭넓게 채택된 기술인 클라우드 마이크로 서비스의 경우 처리율 제한 장치는 보통 API Gateway라는 컴포넌트에 구현된다.

- API 게이트웨이란?
  - 처리율 제한, SSL 종단, Authentication, IP 허용 목록 관리 등을 지원하는 완전 위탁관리형 서비스

3. 처리율 제한 장치를 어디에 두어야 하는가?

정답은 없다. 상황에 따라 결정하면 된다.

- ✓ On-premise 환경의 기술 스택이 서버 측 구현을 지원하기 충분한가?
- ✓ 필요로 하는 처리율 제한 알고리즘은 무엇인가?
- ✓ 우리의 설계가 마이크로 서비스에 기반하고 있고, 이미 API 게이트웨이를 설계에 포함시켰나?
  - 만약 그렇다면 처리율 제한 기능 또한 게이트웨이에 포함 시켜야 할 수도 있다.

04

처리율 제한 알고리즘

4. 처리율 제한 알고리즘

- 처리율 제한을 실현하는 알고리즘은 여러가지 인데, 각기 다른 장 단점을 가지고 있다.
- 이번 발표에서는 대표적인 알고리즘 3가지만 살펴보도록 하겠다.

  - 토큰 버킷 (token bucket)
  - 누출 버킷 (leaky bucket)
  - 이동 윈도 카운터 (sliding window counter)

4. 처리율 제한 알고리즘

1. 토큰 버킷 알고리즘

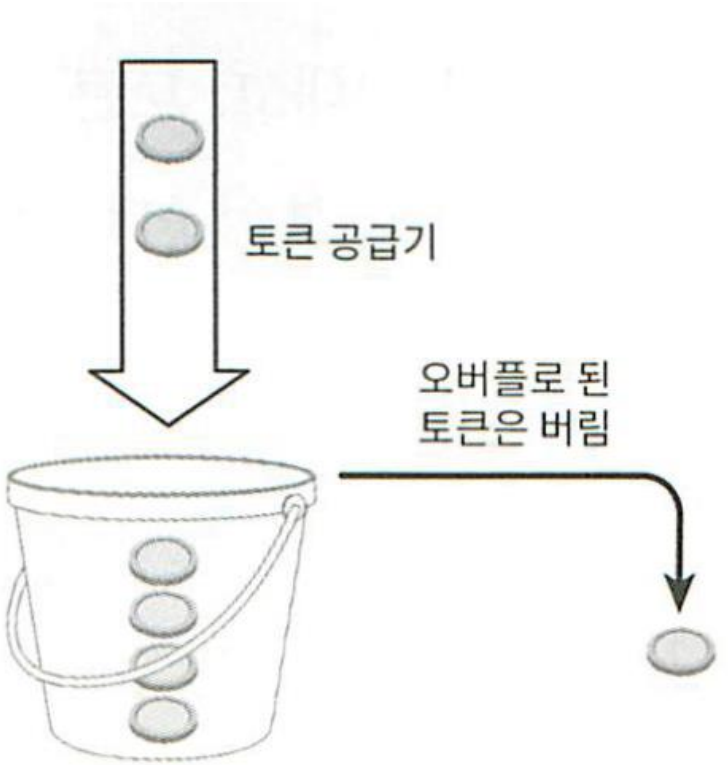
- 간단하고, 알고리즘에 대한 세간의 이해도가 높은 편이라 많은 기업들이 보편적으로 사용하는 알고리즘
  - 아마존, stripe 등의 기업이 사용 중

- 토큰 버킷

  : 지정된 용량을 갖는 컨테이너

  사전 설정된 양의 토큰이 주기적으로 채워진다.

  토큰이 꽉 찬 버킷에는 더 이상의 토큰은 추가 되지 않는다.



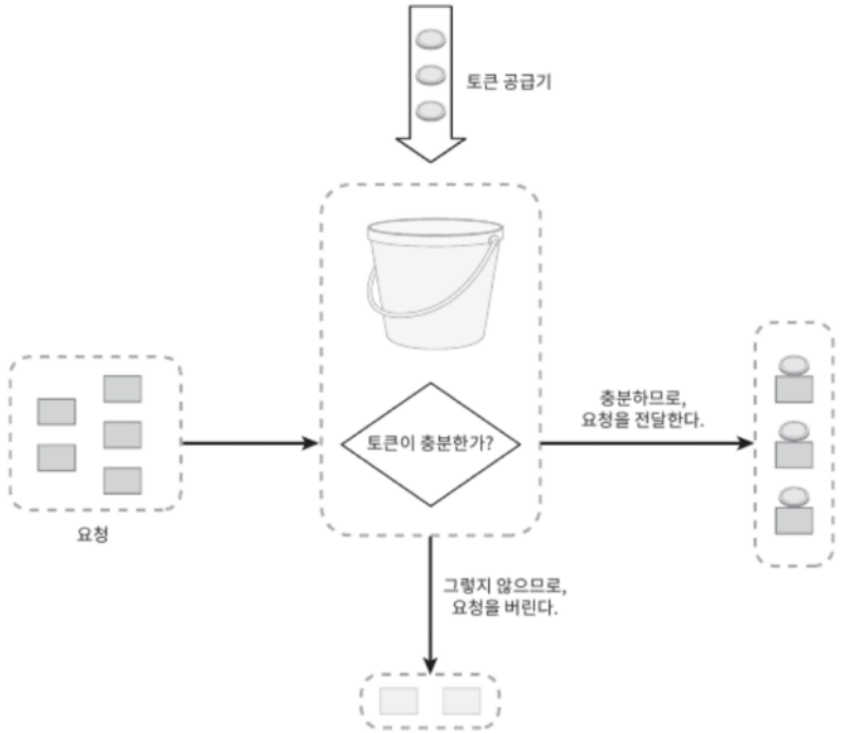
4. 처리율 제한 알고리즘

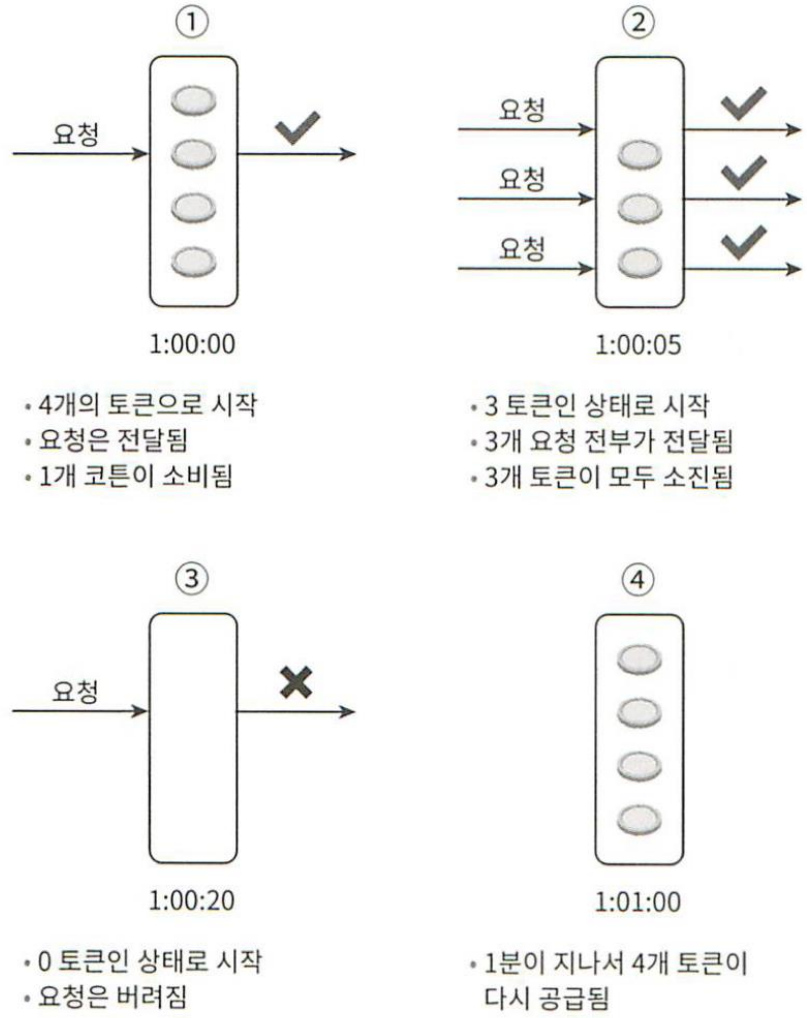
1. 토큰 버킷 알고리즘

1. 각 요청은 처리될 때 마다 하나의 토큰을 사용한다.
2. 요청이 도착하면 버킷에 충분한 토큰이 있는지 검사한다.

3-1. 충분한 토큰이 있는 경우, 버킷에서 토큰을 하나 꺼낸 후 요청을 시스템에 전달한다.

3-2. 충분한 토큰이 없는 경우, 해당 요청은 버려진다.



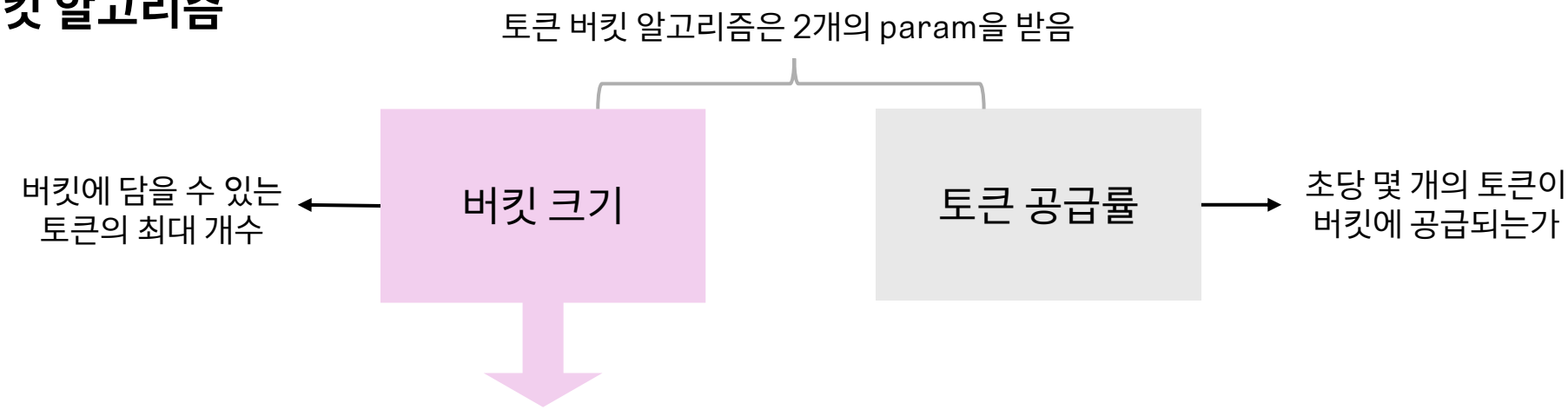


- refill rate : 분당 4

token bucket size : 4인 상황

4. 처리율 제한 알고리즘

1. 토큰 버킷 알고리즘



Q : 버킷은 몇 개나 사용 해야 할까?

A : 공급 제한 규칙에 따라 다르다.

- 통상적으로는 API 엔드 포인트마다 별도의 버킷을 둔다.
- IP 주소별로 처리율 제한을 적용해야 한다면 IP 주소마다 버킷을 하나씩 할당해야 한다.
- 시스템의 처리율을 초당 10,000개 요청으로 제한하고 싶다면,
  모든 요청이 하나의 버킷을 공유하도록 해야 한다.

4. 처리율 제한 알고리즘

1. 토큰 버킷 알고리즘

Pros

- 구현이 쉽다
- 메모리 사용 측면에서 효율적
- 짧은 시간에 집중되는 트래픽도 처리 가능
- 버킷에 남은 토큰이 있기만 하다면 요청은 시스템에 전달된다

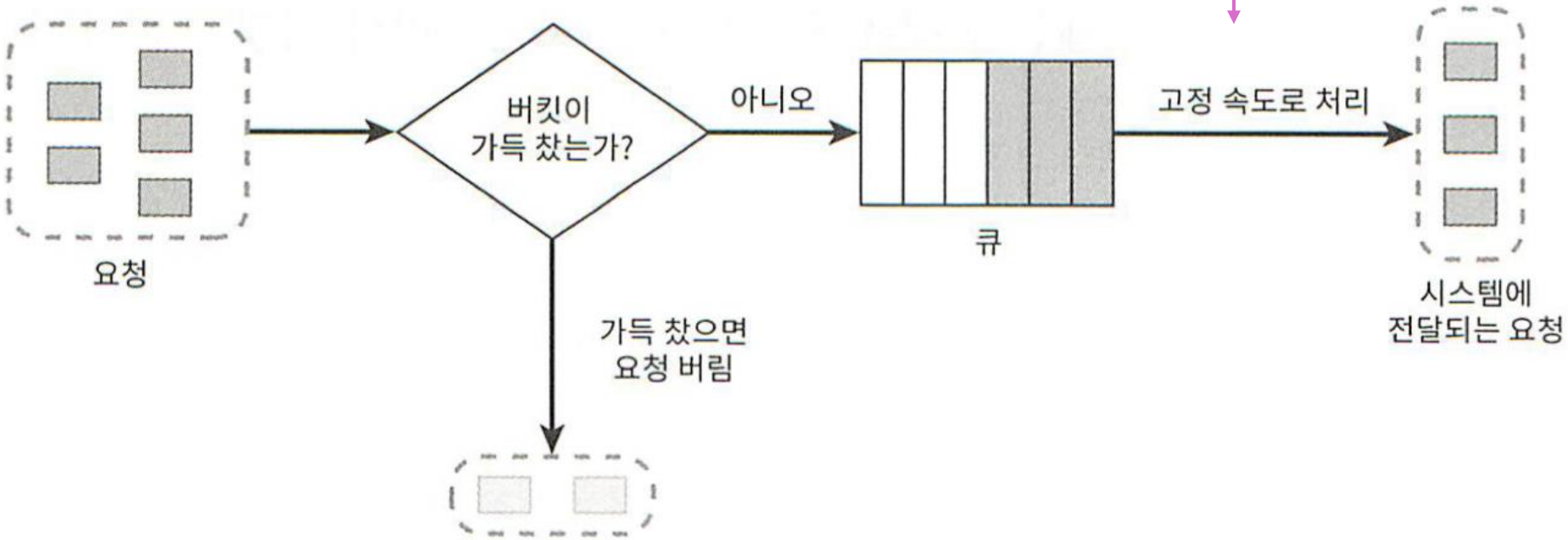
Cons

- 버킷 크기와 토큰 공급률이라는 두 개의 인자 값을 적절하게 튜닝하는 것이 까다롭다.

4. 처리율 제한 알고리즘

2. 누출 버킷 알고리즘

- 토큰 버킷 알고리즘과 비슷하지만 요청 처리율이 고정되어 있다는 점이 다르다.
- 보통 FIFO Queue로 구현한다.
- 두 개의 인자(parameter)를 가진다.
  1. 버킷 크기 (= 큐 사이즈) : 지정된 시간 당 몇 개의 항목을 처리할 것인가
  2. 처리율 : 지정된 시간당 몇 개의 항목을 처리할지 지정하는 값



4. 처리율 제한 알고리즘

2. 누출 버킷 알고리즘

Pros

- 큐의 크기가 제한되어 있어
  메모리 사용량 측면에서 효율적

- 고정된 처리율을 가지고 있기 때문에
  안정적 출력이 필요할 경우 적합

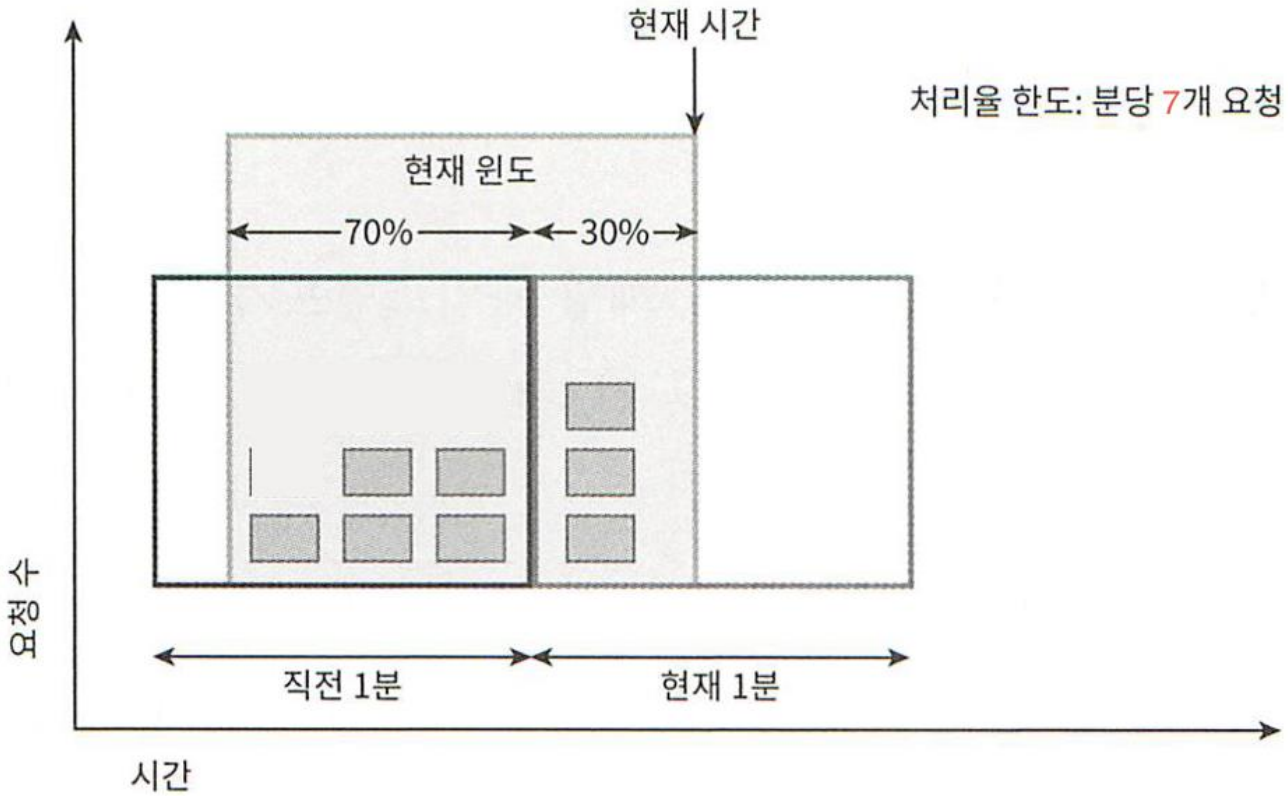
Cons

- 단 시간에 많은 트래픽이 몰리는 경우,
  큐에는 오래된 요청들이 쌓이게 되고
  그 요청들을 제때 처리하지 못하면
  최신 요청들은 버려짐

- 두 개의 인자를 올바르게 튜닝하기가
  까다로움

4. 처리율 제한 알고리즘

3. 이동 윈도 카운터 알고리즘



<현재 상황>

- 현재 시간 : 현재 1분의 30%
- 직전 1분과 겹치는 비율 : 100 - 30 = 70%

➡️ 그렇다면 현재 1분의 30% 상황에서 들어온 요청의 개수는?

현재 1분간의 요청 수 +
직전 1분간의 요청 수 X 이동 윈도와 직전 1분이 겹치는 비율

- 요청의 개수 : 3 + 5 X 70% = 6.5개 (내림 시 6개)
- 처리율 한도 = 분 당 7개이므로 신규 요청 허용 가능!
  - 이후 요청은 X..

4. 처리율 제한 알고리즘

3. 이동 윈도 카운터 알고리즘

Pros

- 이전 시간대의 평균 처리율에 따라 현재 윈도의 상태를 계산하므로 짧은 시간에 몰리는 트래픽에도 잘 대응한다.
- 메모리 효율이 좋다.

Cons

- 직전 시간대에 도착한 요청이 균등하게 분포되어 있다고 가정한 상태에서 추정치를 계산하기 때문에 다소 느슨하다.

05

처리율 제한 아키텍처 설계

5. 처리율 제한 아키텍처 설계

기본 아이디어

: 얼마나 많은 요청이 접수되었는지를 추적할 수 있는 카운터를 추적 대상별로 두고,

이 카운터의 값이 어떤 한도를 넘어서면 한도를 넘어 도착한 요청은 거부하는 것

어떤 추적 대상을
카운트 할 것인가??

- 사용자별로?
- IP 주소별로?
- API endpoint || service단위?

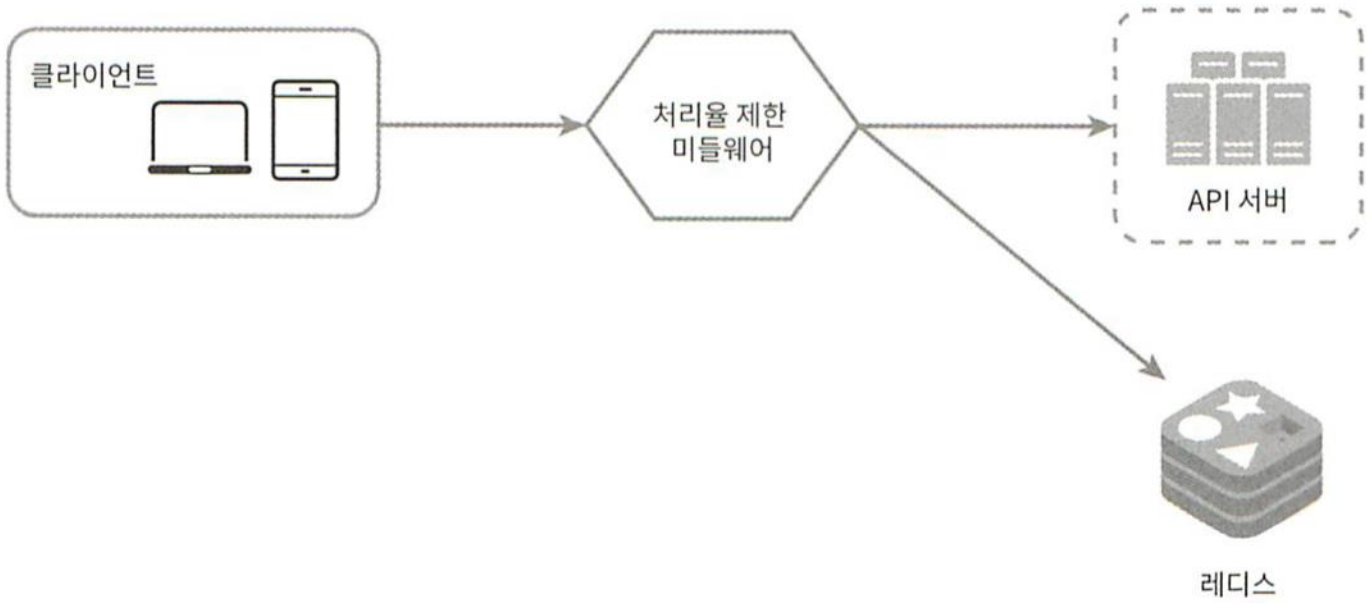
카운터는 어디에
보관해야할까?

- DB || Cache?
- DB는 Disk 접근이기 때문에 느리다.
- 따라서 메모리 상에서 동작하는 캐시가 바람직하다.

5. 처리율 제한 아키텍처 설계

캐시는 빠른 데다가 시간에 기반한 만료 정책을 지원하기 때문에 적합하다.

- 일례로 처리율 제한 장치를 구현할 때 자주 사용되는 메모리 기반 저장장치인 Redis가 있다.



5. 처리율 제한 아키텍처 설계

조금 더 상세한 설계를 해보자.

1. 처리율 제한 규칙은 어떻게 만들어지고 어디에 저장되는가?

➡️ 일반적으로 Configuration file 형태로 디스크에 저장된다.

2. 처리가 제한된 요청들은 어떻게 처리되는가?

➡️ 어떤 요청이 한도 제한에 걸리면 API는 HTTP 429 응답을 클라이언트에게 보낸다.

경우에 따라서는 한도 제한에 걸린 메시지를 나중에 처리하기 위해 Queue에 보관할 수도 있다.

5. 처리율 제한 아키텍처 설계

클라이언트는 자기 요청이 처리율 제한에 걸리고 있는지를 어떻게 감지할까?

자신의 요청이 처리율 제한에 걸리기까지 얼마나 많은 요청을 보냈는지 어떻게 알 수 있을까?

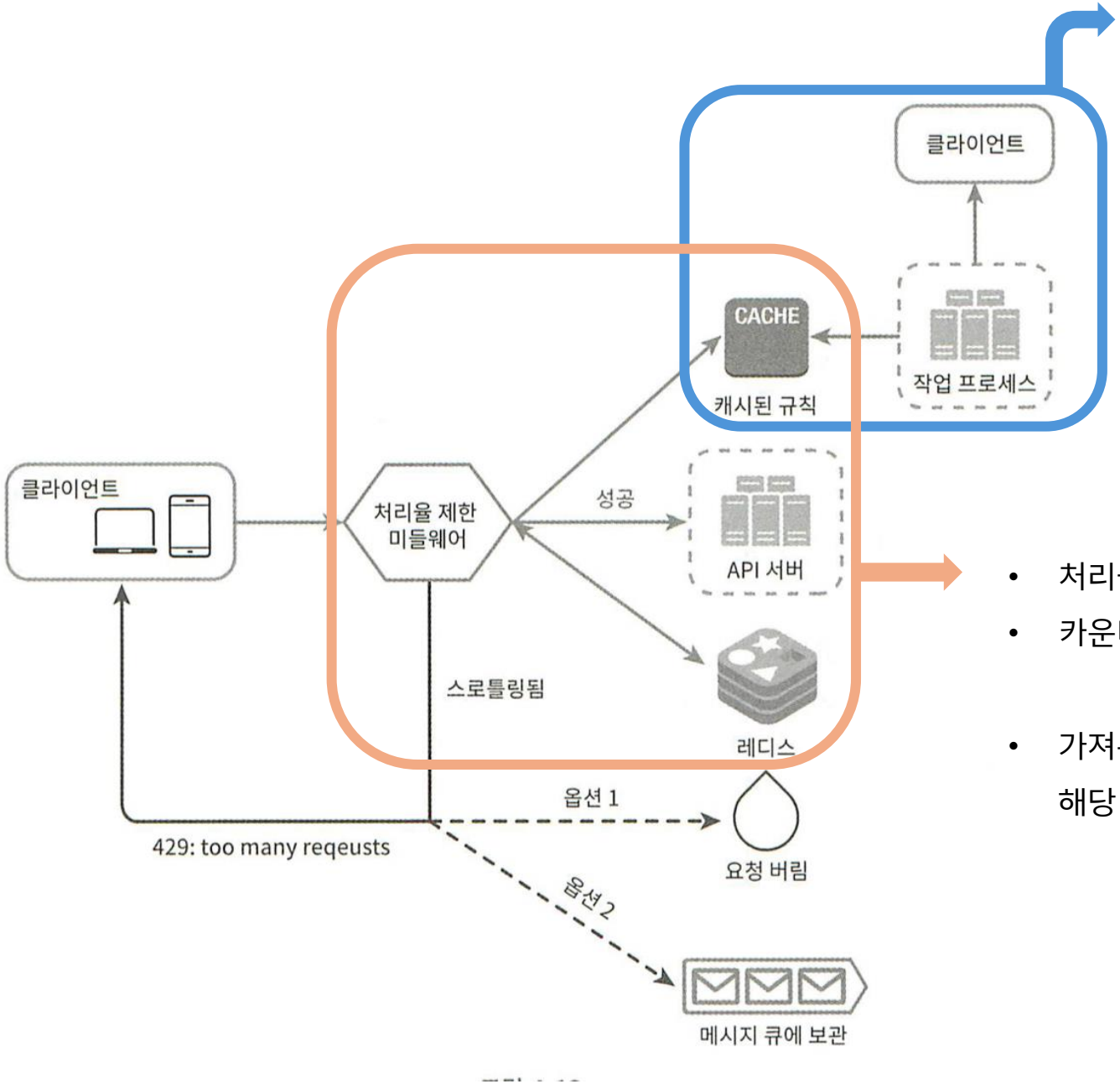
HTTP Response Header

처리율 제한 장치는 다음의 HTTP Header를 클라이언트에게 보낼 수 있다.

- X-Ratelimit-Remaining : 윈도 내에 남은 처리 가능 요청의 수
- X-Ratelimit-Limit : 매 윈도마다 클라이언트가 전송할 수 있는 요청의 수
- X-Ratelimit-Retry-After : 한도 제한에 걸리지 않으려면 몇 초 뒤에 요청을 다시 보내야 하는지 알림

➡️ 사용자가 너무 많은 요청을 보내면 429 too many requests 오류를 X-Ratelimit-Retry-After 헤더와 함께 반환하도록 한다.

5. 처리율 제한 아키텍처 설계



- 처리율 제한 규칙은 디스크에 보관
- 작업 프로세스 (worker)는 수시로 규칙을 디스크에서 읽어 캐시에 저장한다.

- 처리율 제한 미들웨어는 제한 규칙을 캐시에서 가져온다.
- 카운터 및 마지막 요청의 timestamp는 Redis 캐시에서 가져온다.

- 가져온 규칙과 값들에 근거하여 해당 미들웨어는 해당 요청을 허용할지 거부할지 결정한다.

06

분산 환경에서의 처리율 제한 장치 구현

6. 분산 환경에서의 처리율 제한 장치 구현

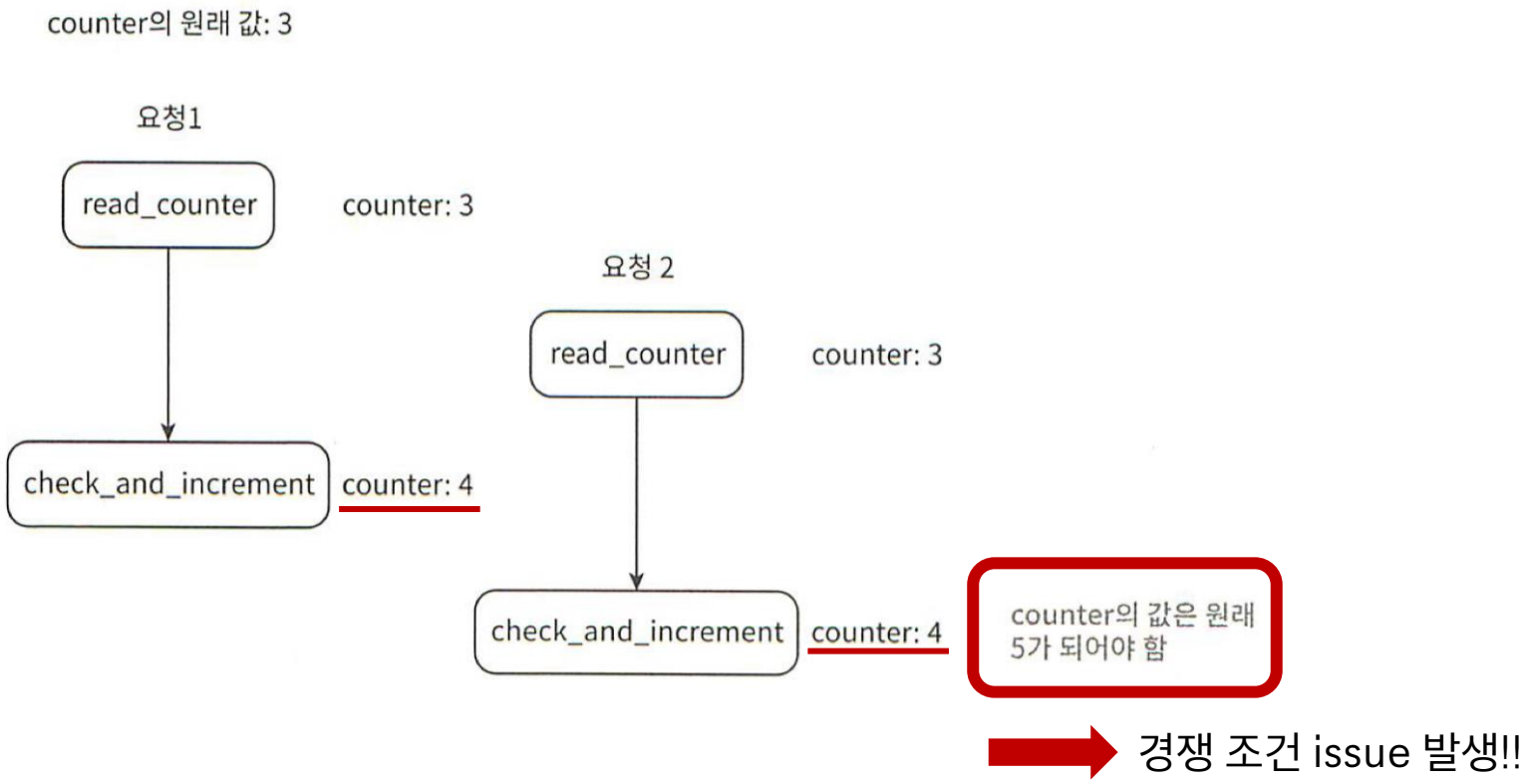
단일 서버를 지원하는 처리율 제한 장치를 구현하는 것은 어렵지 않다.
하지만 여러 대의 서버와 병렬 스레드를 지원하도록 시스템을 확장하는 것은 또 다른 문제이다.

두 가지의 어려운 문제가 존재한다.

1. 경쟁 조건 (race condition)
2. 동기화 (synchronization)

6. 분산 환경에서의 처리율 제한 장치 구현

1. 경쟁 조건



6. 분산 환경에서의 처리율 제한 장치 구현

1. 경쟁 조건

- 경쟁 조건을 해결하는 방법들

방법	Lock	Lua Script	Sorted Set (Redis의 자료구조)
특징	• 대표적인 경쟁 조건 문제의 해결책 • 시스템의 성능이 저하될 수 있음	• Lua 프로그래밍 언어로 작성된 사용자 지정 스크립트 • Redis내에서 복잡한 작업을 Atomic 하게 수행	• 우선 순위 처리가 가능 • 특정 점수 범위 내의 요소들을 효율적으로 조회할 수 있어 시간 기반의 작업 스케줄에 유용 • 주로 슬라이딩 윈도우 로그를 구현할 때 사용

잘 사용하지 않음

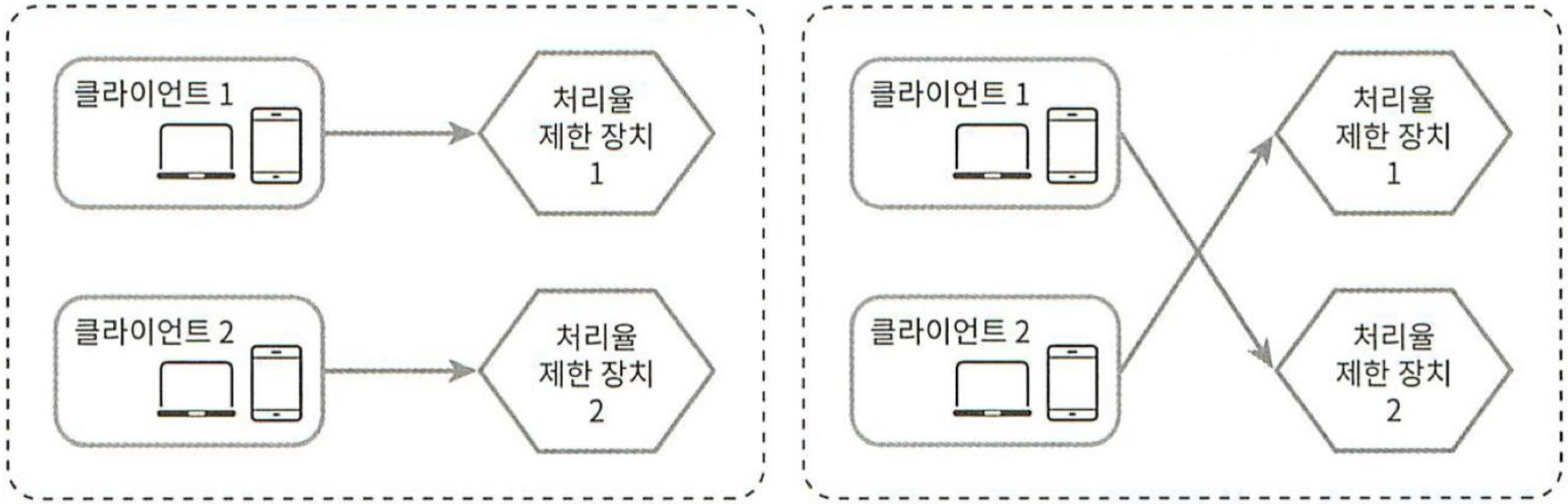
6. 분산 환경에서의 처리율 제한 장치 구현

2. 동기화 이슈

만약 한 대의 처리율 제한 장치 서버로는 충분하지 않아 여러 대의 처리율 제한 장치를 써야 하면 어떤 문제가 발생할까?

- 동기화 문제 발생
  - ➢ 처리율 제한 장치1과 2는 항상 같은 상태를 유지하여야 한다.

- web 계층의 state-less 관련 문제
  - ➢ 처음 요청을 보낸 후 다음 요청을 보낼 때 각기 다른 제한 장치로 보낼 수도 있다. (Pic. right)



6. 분산 환경에서의 처리율 제한 장치 구현

2. 동기화 이슈

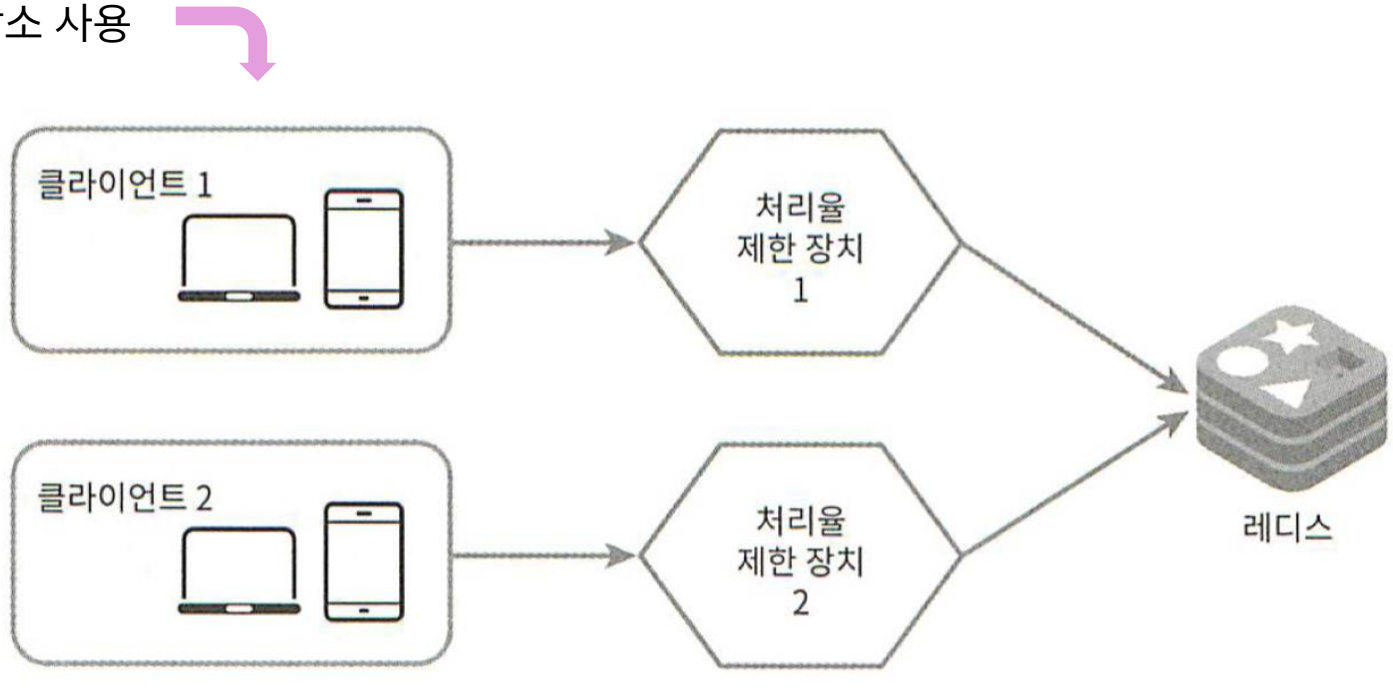
해결책 1. 고정 세션을 활용하여 클라이언트로부터의 요청을 항상 같은 처리율 제한 장치로 보낼 수 있도록 하기

➡️ 비추천!

why?) 규모면에서 확장 가능하지도 않고 유연하지도 않기 때문이다.

해결책 2. Redis와 같은 중앙 집중형 데이터 저장소 사용

➡️ 추천!



6. 분산 환경에서의 처리율 제한 장치 구현

구현을 해보자! 다음 시간에

with JWT, Spring Security Filter, Redis, Token Bucket Algorithm(Bucket4j)