[4장] 처리율 제한 장치의 설계

가상 면접 사례로 배우는 대규모 시스템 설계 기초

이민석 / unchaptered

https://inblog.ai/monthly-cs
https://github.com/monthly-cs/2024-03-system-design-interview-1

처리율 제한 장치(Rate Limiter)란?

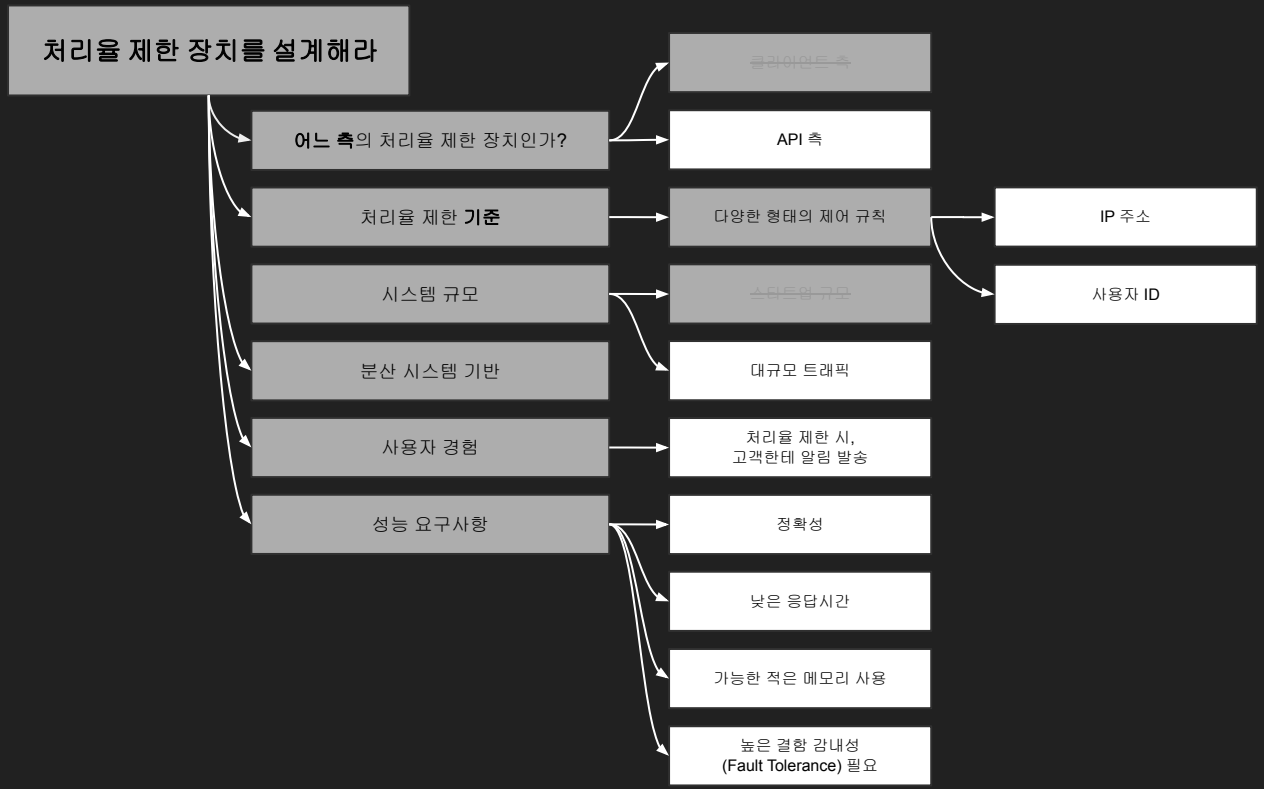
클라이언트/서비스가 보내는 트래픽의 **처리율**을 제어하기 위한 장치

- DoS(Denial of Service) 공격에 대한 자원 고갈 방지
- 비용 절감
- 서버 과부화 방지

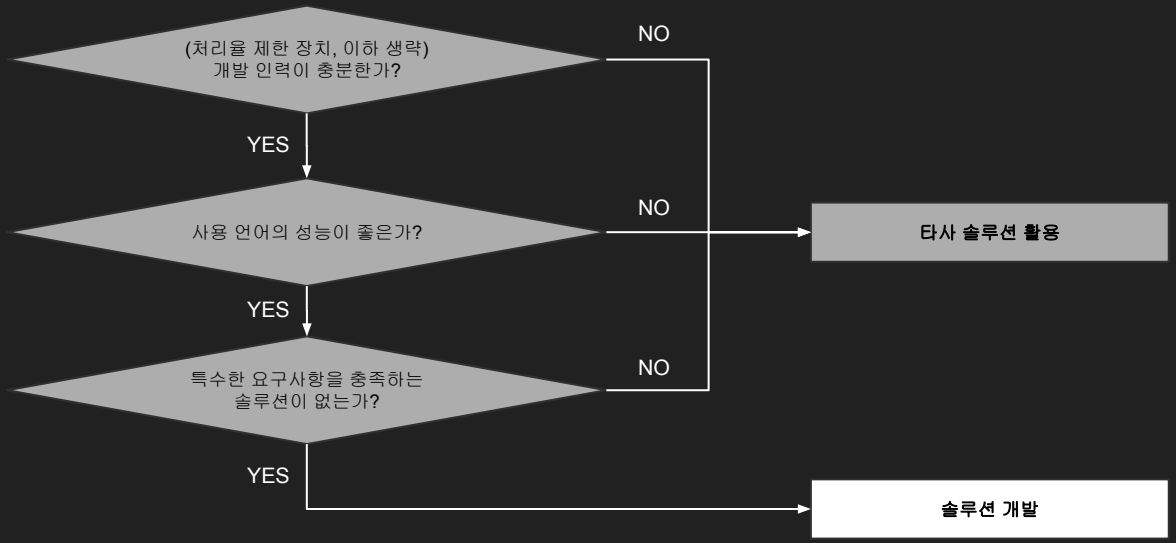
1단계 | 요구사항 분석 | 문제 이해 및 설계 범위 확정

1. 시스템 설계를 위한 **기능 요구사항** 탐색
2. 개발팀의 기술 스텍 및 리소스 파악
3. **정리**

1단계 - 1 | 기능 요구사항 탐색



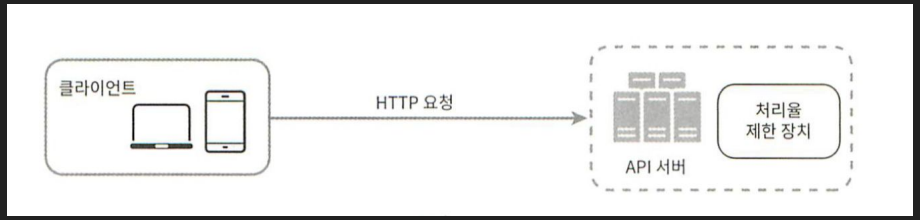
1단계 - 2 | 개발팀의 스텍 및 리소스 파악



2단계 | 초기 설계 | 개략적인 설계안 제시 및 동의 구하기

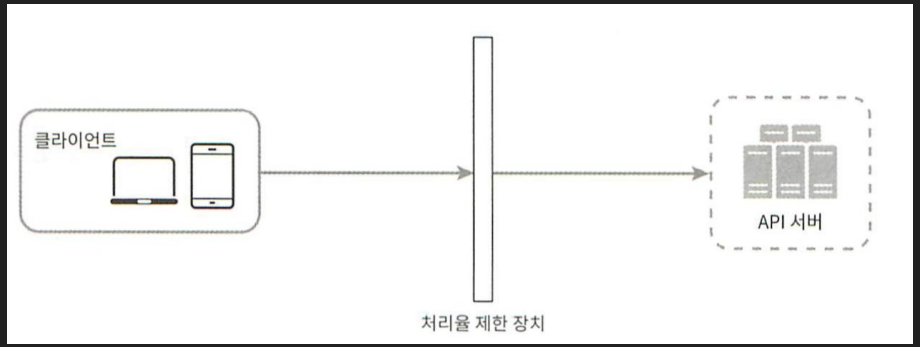
1. 처리율 제한 장치의 **위치**
2. 처리율 제한 **알고리즘**의 선택
3. 처리율 제한 장치의 개략적인 **아키텍처** 설계

2단계 - 1 | 처리율 제한 장치의 위치



관점 1
API 서버에서 사용하는 언어에 따라서 처리율 제한 장치의 성능에 영향이 간다.

관점 2
N개의 API 서버에 분할된 N개의 처리율 제한 장치는 동시성 이슈를 겪을 가능성이 매우 높다.

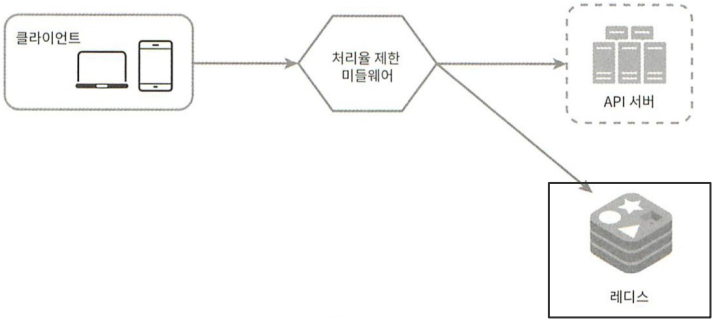


관점 3
처리율 제한 장치가 SPoF으로 작용할 가능성이 있다.
처리율 제한 장치가 다운되었을 때, 우회 경로가 설정되어 있어야 할 것 같다.

2단계 - 2 | 처리율 제한 알고리즘의 선택

	요구사항	토큰 버킷 알고리즘	누출 버킷 알고리즘	고정 윈터 카운터 알고리즘	이중 윈도 로깅 알고리즘	이중 윈도 로깅 알고리즘
우선순위 높음	정확성	○	○	※	○	※
	낮은 응답시간	○	※	?	?	?
	가능한 적은 메모리 사용	○	○	○	※	○
우선순위 낮음	높은 결함 감내성 (Fault Tolerance) 필요	?	?	?	?	○

2단계 - 2 | 처리율 제한 장치의 개략적인 아키텍처 설계



관점 1
기준 당 요청량인 **카운터**는 아래 이유로 인메모리 데이버데이스 사용
1. 10ms 대에 해당하는 빠른 읽기, 쓰기 성능
2. Map 기반의 자료 구조로 O(1)으로 카운터에 적합
3. TTL 설정 지원

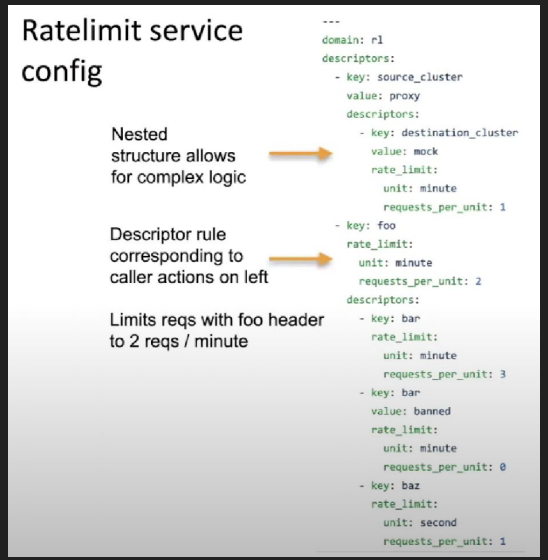
3단계 | 상세 설계

1. 처리율 제한 규칙 (Lyft → EnvoyProxy)
2. 처리율 한도 초과 트래픽의 처리
3. 상세 설계
4. 분산 환경에서의 제약 조건
   a. 경쟁 조건
   b. 동기화 이슈
   c. 성능 최적화
   d. 모니터링

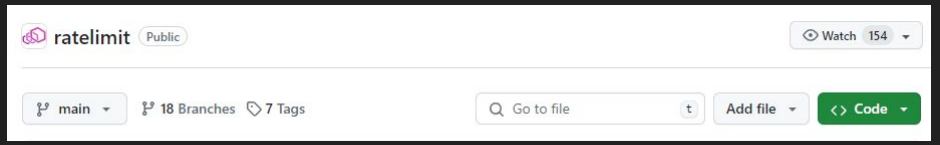
3단계 - 1(a) | 처리율 제한 규칙

일반적으로 **선언형 프로그래밍 방식**으로 구현되며, 설정 파일(configuration file) 들에 규칙을 명시한다.

학습 도서에는 github.com/lyft/ratelimit을 말했지만 현재는 github.com/envoyproxy/ratelimit으로 구현되어 있다.



https://youtu.be/2EKU8zCQAow?si=Ds_lDtly1KyLElot



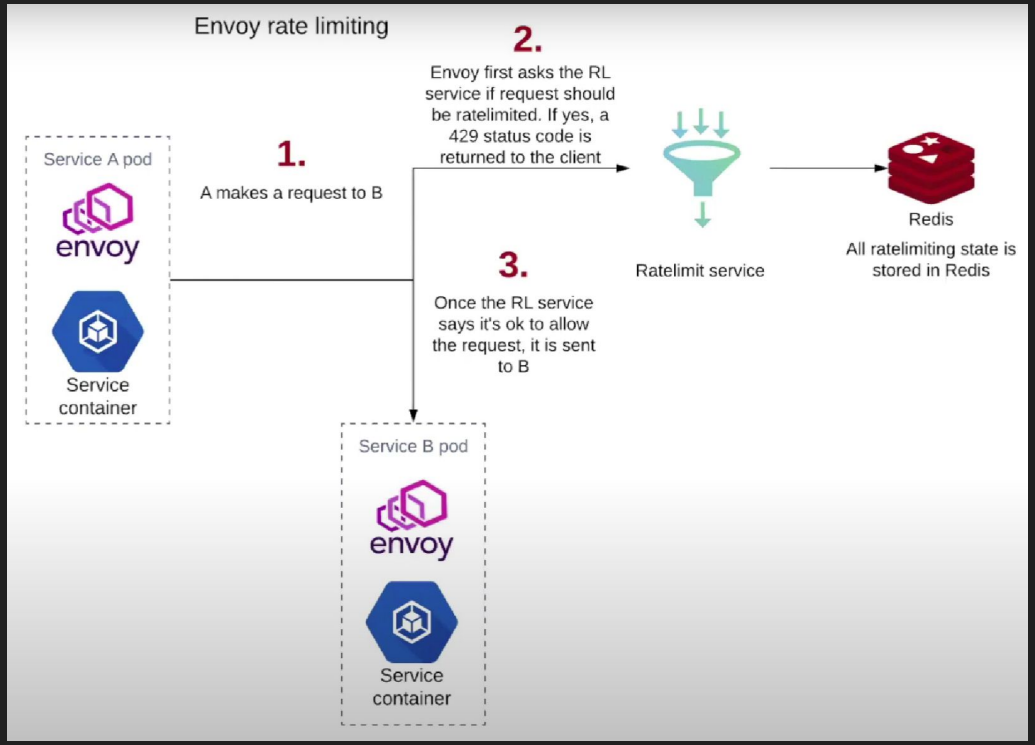
https://github.com/envoyproxy/ratelimit

3단계 - 1(b) | 처리율 제한 규칙 | Envoy란?

Envoy는 L4,L7에서 작동하는 Proxy 서버의 일종으로 아래의 기능들을 제공한다.
- DownStream/Upstream/Cluster/Network Filter/Connection Pools/Threading Model
- L7 Proxy, Splited Load Balancer, Block specific routes(/api/v1/admin)
- L4 Proxy(tcp router)
- DNS Record
- HTTPS on Envoy (lets encrypt),
- HTTP2/ on Envoy
- Disable 1.1/1.0 Enable TLS 1.2 and TLS 1.3 Only on Envoy
- SSL Labs Tes

https://youtu.be/40gKzHQWgP0?si=ZSnijs0BaMAyfyHj

3단계 - 1(c) | 처리율 제한 규칙 | EnvoyProxy RateLimit Tinder



How Tinder implemented Envoy global rate limiting at scaleVirtual - Yuki Sawa

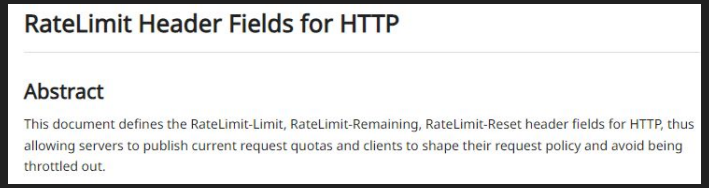
CNCF [Cloud Native Computing Foui
구독자 11.1만명
구독
46
공유

https://youtu.be/40gKzHQWgP0?si=ZSnijs0BaMAyfyHj

3단계 - 2 | 처리율 한도 초과 트래픽 처리

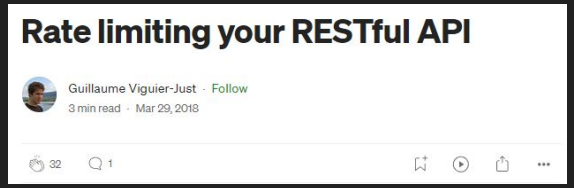
X-Ratelimit-Remaining : 윈도 내에 남은 처리 가능 요청의 수
X-Ratelimit-Limit : 매 윈도마다 클라이언트가 전송할 수 있는 요청의 수
X-Ratelimit-Retry-After : 한도 제한에 걸리지 않으려면 몇 초 뒤에 요청을 다시 보내야 하는지 알림

관점 1
일반적으로 특정한 프로그램에서 만든 Custom Header는 접두사에 X를 붙여서 나온다.



https://www.ietf.org/archive/id/draft-polli-ratelimit-headers-02.html

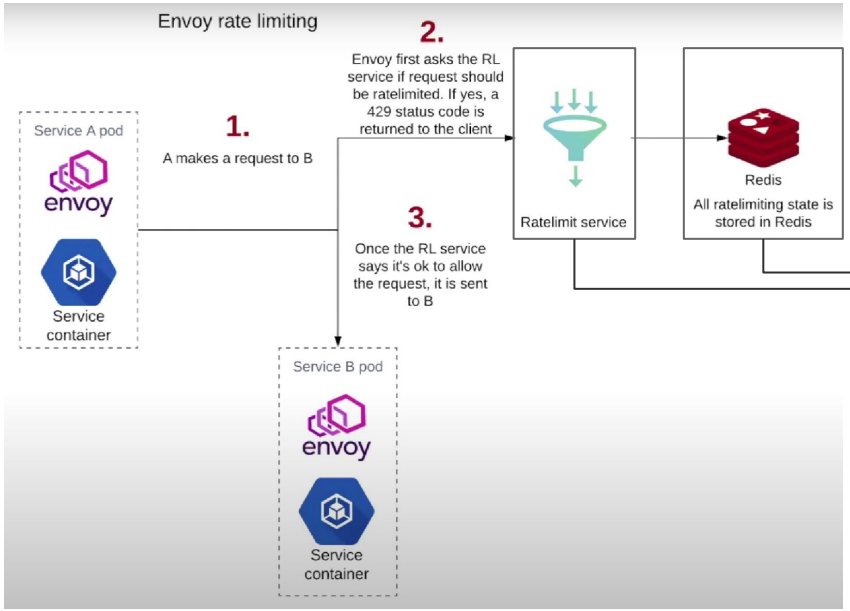
관점 2
2020년에 IEFE에 게시된 Ratelimit-Remaining 표준을 따르되, 요구사항에 맞춰서 수정된 Custom Header를 사용



https://medium.com/@guillaume.viguierjust/rate-limiting-your-restful-api-3148f8e77248

관점 3
Twitter에서는 X-Rate-limit-Remaining을 쓰고
GitHub에서는 X-Ratelimit-Remaining을 쓰듯이,
기업 마다 사용 사례가 다르다.

3단계 - 3 | 상세 설계

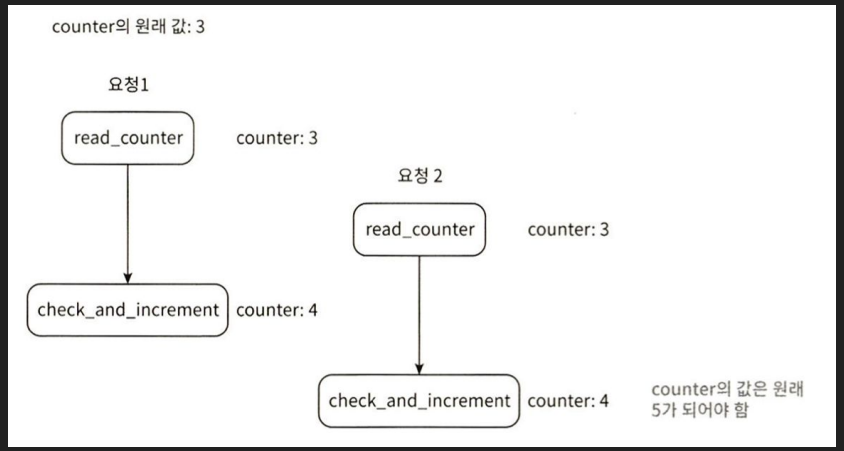


관점 1
처리율 제한 장치를 별도의 **사이드카**처럼 배포

관점 2
처리율 제한 장치의 **데이터 기록**은 레디스를 사용

https://youtu.be/40gKzHQWgP0?si=ZSnijs0BaMAyfyHj

3단계 - 4(a) | 분산 환경에서의 제약 조건 | 경쟁 조건

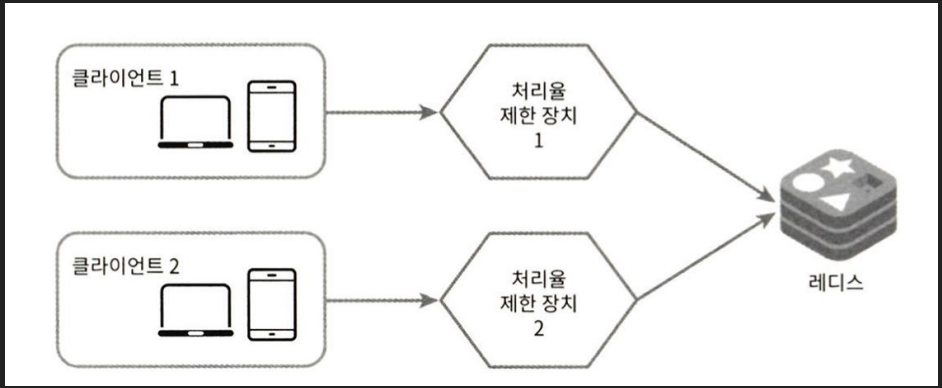


https://jjingho.tistory.com/155

관점 1
경쟁 조건의 문제를 해결하는 방법은 **Lock**이다.
하지만 처리율 제한 장치 설계의 두 번째 제한 사항인 **낮은 응답시간**에 반하는 기술적 선택이다.

관점 2
혹은 LuaScript나 레디스의 SortedSet을 사용하면 된다.

3단계 - 4(b) | 분산 환경에서의 제약 조건 | 동기화 이슈



https://jjingho.tistory.com/155

관점 1
분산 환경에서 동기화 이슈를 해결하는 쉬운 방법은 Sticky Session이지만, 처리율 제한 장치 설계의 두 번째 제한 사항인 **낮은 응답시간**에 반하는 기술적 선택이다.

관점 2
중앙 집중형 레디스를 사용하여 이 문제를 해결 가능

3단계 - 4(c) | 분산 환경에서의 제약 조건 | 성능 최적화

관점 1
Edge Computing을 통해서 지연시간 감소

관점 2
최종 일관성 모델을 사용하여, 데이터 동기화에 많은 시간 및 자원이 소모되지 않도록 구현
> 진짜 쉽게 생각하면,
> 최종 일관성 모델이란, “최종 데이터만 동기화되면 되는거다!”라고 생각해도 좋습니다.
해당 내용은 6장에서 자세히 나옵니다.

3단계 - 4(d) | 분산 환경에서의 제약 조건 | 모니터링

관점 1
OpenTelemetry라는 exporter를 통해서
github.com/envoyproxy/ratelimit의 작동 과정을
모니터링 할 수 있다.

4단계 - 1 | 요약

처리율 제한 장치는 다음과 같이 2가지로 구성됨

- 처리율 제한 장치 서비스
- 처리율 제한 장치 카운터

처리율 제한 장치 서비스는 직접 구현하거나 github.com/envoyproxy/ratelimit 등의 오픈소스를 활용할 수 있다.
처리율 제한 장치 카운터는 인메모리 데이터베이스인 Redis를 사용할 수 있다.

고가용성(HA)을 위해서 분산 환경에 배포하면, 경쟁 조건(Race Condition)이 생기고 이를 SortedSet으로 극복 가능하다.
만약 분산 환경에서, 동기화 이슈(Sync Issue)가 생기면 중앙 집중형 redis를 사용해서 이를 해결 가능하다.