

4장 : 처리율 제한 장치의 설계

📍 난이도	☆☆☆☆
📅 학습날짜	@2025년 12월 9일

개요

처리율 제한 장치

처리율 제한 장치의 위치

처리율 제한 알고리즘

- 1. 토큰 버킷 (Token Bucket)
- 2. 누출 버킷 (Leaky Bucket)
- 3. 고정 윈도우 카운터 (Fixed Window Counter)
- 4. 이동 윈도우 로깅 (Sliding Window Log)
- 5. 이동 윈도우 카운터 (Sliding Window Couter)

상세 설계

- 1. 처리가 제한된 요청들은 어떻게 처리하는가?
- 2. 처리율 제한 규칙은 어떻게 만들어지고 어디에 보관되는가?

추가 학습 권장사항

개요

해당 챕터는 특정 기간 내에 전송되는 클라이언트의 요청을 제한하는 처리율 제한 장치에 대해 설명한다.

처리율 제한 장치

처리율 제한 장치를 설계하였을 때의 **장점**은 다음과 같다.

- DoS (Denial Of Service) 공격에 의한 자원 고갈을 방지할 수 있다
 - 트위터 : 3시간 동안 300개의 트윗 가능
 - 구글 독스 : 분당 300회의 Read 제한
- 제한된 처리량을 토대로 비용을 절감할 수 있다.
- 서버 과부하를 방지할 수 있다.
 - 봇(Bot)으로 인해 발생하는 트래픽, 사용자의 잘못으로 발생하는 트래픽 등에 대해 대응 가능하다.

처리율 제한 장치의 위치

처리율 제한 장치는 보통, **429 (Too Many Request)** HTTP 상태 코드를 통해 너무 많은 요청이 들어왔음을 알린다.

이때, 처리율 제한 장치의 위치는 클라이언트, 서버, 미들웨어 3가지로 구분하여 생각해볼 수 있다.

1. 클라이언트

위변조의 위험이 있어서 일반적으로 처리율 제한 장치를 위치하지 않는다.



클라이언트에 처리율 제한 장치를 위치하면 안되는 이유, 최선의 방안

2. 서버

서버에서의 처리율 제한 장치는 일종의 서비스로 제공할 수 있다.



서버에서 처리율 제한 장치를 제공하는 방법

- 비밀번호를 Brute Force로 보호하기 위해서 내부에 로직을 포함?
- 서비스 내부에 포함시키는게 이상한가?
 - “처리율 제한 장치”의 개념에서는 결국 서버가 처리할 수 있는 양만 대응하면 되는거라 괜찮지 않나..?
 - Proxy 같은 역할로 요청이 바로 튕길것 같아서 처리율이 제한될것 같은데?

3. 미들웨어

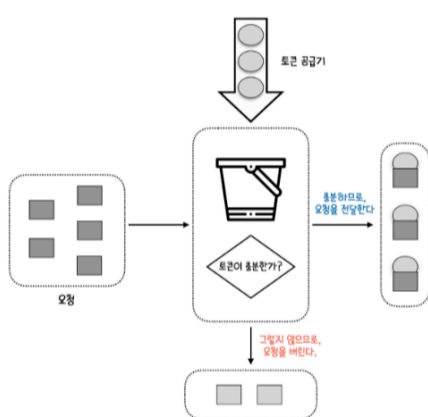
기업에서 대중적으로 사용되는 마이크로 서비스에서는 처리율 제한장치를 API 게이트웨이 컴포넌트 혹은 미들웨어에 구현한다.

[API 게이트웨이의 추가적인 역할]

- SSL 터미널
- 사용자 인증
- IP 허용 목록 (WhiteList)

처리율 제한 알고리즘

1. 토큰 버킷 (Token Bucket)



토큰 버킷은 지정된 용량을 갖는 컨테이너이다.

토큰 버킷 알고리즘은 구현이 쉽고 메모리 사용 측면에서도 효율적이지만, **버킷 크기와 토큰 공급률을 적절히 튜닝하기 어렵다**는 단점이 있다.

1. 각 요청은 처리될 때마다 하나의 토큰을 사용한다.
2. 요청이 도착하면 버킷에 토큰이 있는지 검사하고, 충분한 토큰이 있는 경우 토큰을 꺼내 요청을 시스템에 전달한다.
3. 토큰이 없다면 해당 요청은 버려지게 된다.

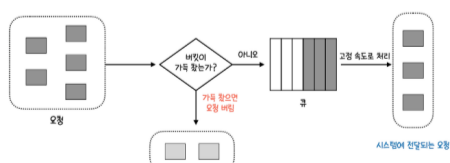
토큰 버킷 알고리즘은 보통 2개의 인자를 받는다.

하나는 버킷 크기로 **버킷에 담을 최대 토큰 개수**를 의미하고, 나머지 하나는 **토큰 공급률로 초당 버킷에 공급되는 토큰의 숫자**를 의미한다.



자주 사용되는 처리율 제한 알고리즘이라고 했는데, 버려지는 요청은 어떻게 핸들링해주나요? 말 그대로 그냥 버리나요?

2. 누출 버킷 (Leaky Bucket)



누출 버킷 알고리즘은 토큰 버킷 알고리즘과 비슷하지만 약간의 차이가 있다.

요청 처리율이 고정되어 있으며, **큐를 사용한다**는 점이다.

- 토큰 버킷 : 버킷의 최대 토큰 개수 고정, 토큰 공급률 고정
 - 요청 처리는 바로바로
- 누출 버킷 : 고정된 크기의 큐 존재, 고정 속도로 큐 처리
 - 요청 처리는 큐가 요청을 처리할 때 발생한다.

1. **요청이 도착하면 큐가 가득 차 있는지 본다.** 빈자리가 있는 경우에는 큐에 요청을 추가한다.
2. 큐가 가득 차 있는 경우에는 새 요청은 버린다.

3. 지정된 시간마다 큐에서 요청을 꺼내어 처리한다.

장점

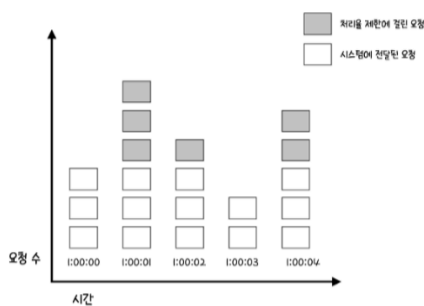
1. 큐의 크기가 제한되어 있어, 메모리 사용량 측면에서 효율적이다
2. 고정된 처리율을 갖고 있기 때문에 안정적 출력이 필요한 경우에 적합하다.

단점

1. 단시간에 많은 트래픽이 몰리는 경우 큐에는 오래된 요청들이 쌓이게 되고, 해당 요청들을 제때 처리하지 못하면 최신 요청들은 버려지게 된다

누출 버킷의 어려운 점은 큐에서 토큰을 처리하는 속도와 큐의 크기를 튜닝하기가 어렵다는 점이다.

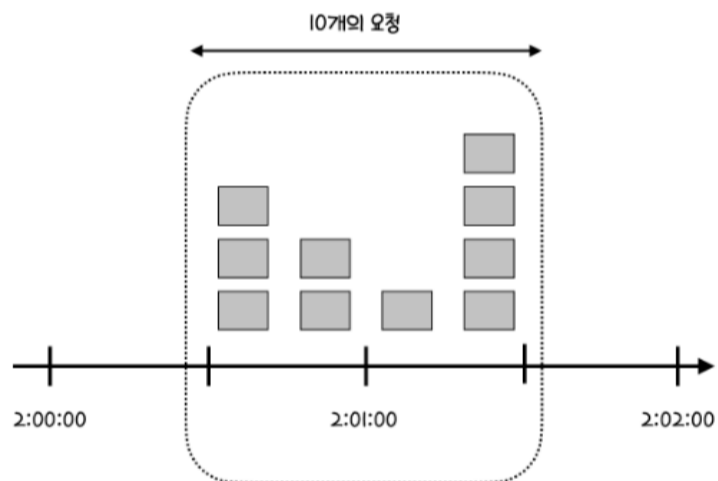
3. 고정 윈도우 카운터 (Fixed Window Counter)



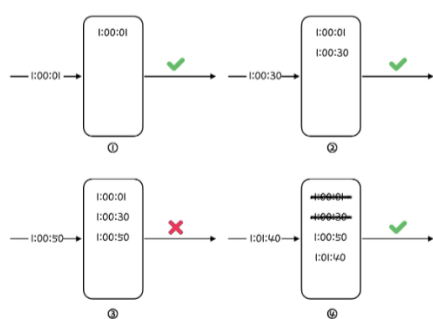
1. 타임라인을 고정된 간격의 윈도우로 나누고, 각 윈도우마다 카운터를 붙이낱.
2. 요청이 접수될 때마다 이 카운터의 값을 1씩 증가시킨다.
3. 카운터의 값이 사전에 설정된 임계치에 도달하면 새로운 요청은 새로운 윈도우가 열릴 때까지 버려진다.

치명적 단점 : 트래픽이 순간적으로 몰리는 상황에서 기대했던 시스템의 처리 한도보다 많은 양을 순간적으로 처리하게 된다.

ex) 윈도우의 크기가 1분일때, 30초 간격으로 요청이 집중되는 경우, 예상했던 최대5의 트래픽이 아닌 10의 트래픽이 집중되게 된다.



4. 이동 윈도우 로깅 (Sliding Window Log)



이동 윈도우 로깅 알고리즘은 고정 윈도우 알고리즘의 문제점을 해결한 알고리즘이다.

고정 윈도우와 달리, 이동 윈도우 로깅 알고리즘은 요청의 타임스탬프를 추적한다.

1. 각 요청의 타임스탬프를 추적한다.
 - 레디스의 정렬 집합 같은 캐시에 보통 보관한다.
2. 새 요청이 오면 만료된 타임스탬프는 제거한다. 만료된 타임스탬프는 그 값이 현재 윈도우의 시작 시점보다 오래된 타임스탬프를 의미한다.
3. 새 요청의 타임스탬프를 로그에 추가한다.
4. 로그의 크기가 허용치보다 같거나 작으면 요청을 시스템에 전달한다. 그렇지 않은 경우에는 처리를 거부한다.

→ 어느 순간의 윈도우를 보더라도, 허용되는 요청의 개수는 시스템의 처리율 한도를 넘지 않는다.

단점

- 거부된 요청의 타임스탬프도 보관하기 때문에 다량의 메모리를 사용한다.



거부된 요청을 저장하는 방법

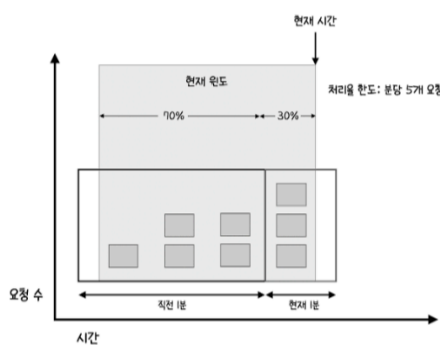
t1 t2 t3 t4 t5

- t3 거절
- t5 가 들어오면 t3 거절 + t4 수용 했으니 2개가 차서 t3 거절
 - 문제 : 계속 거절하게 되는 문제가 발생하지 않나?
 - 찾아봐야 하나, 맞다고 나왔음

실무에서 과연 쓰나?

- 프론트는 계속 튕기고
- 서버는 튕기고

5. 이동 윈도우 카운터 (Sliding Window Counter)



고정 윈도우 알고리즘과 이동 윈도우 로깅 방식을 결합한 방식이다

처리율 제한 장치의 한도가 분당 7개라고 할 때, 직전 1분 동안 5개의 요청, 현재 1분동안 3개의 요청이 왔다.

현재 1분에 새 요청이 왔을 때 윈도우에 몇 개 요청이 왔는지 계산한다.

- 현재 1분간의 요청수 + 직전 1분간의 요청수 x 이동 윈도우와 직전1분이 겹치는 비율

이 공식에 따르면 현재 윈도우에 들어있는 요청은 $3 + 5 \times 70\%$ 로 = 6.5개이다.

내림해도 되고 올림해도 되기 때문에 내린다고 치면 아직 1개의 요청까지는 더 받을 수 있고 이후에 들어오는 요청은 시스템으로 전달될 수 없다.

장점

- 이전 시간대의 평균 처리율에 따라 현재 윈도우의 상태를 계산하므로, 짧은 시간에 몰리는 트래픽에도 잘 대응한다
- 메모리 효율

단점

- 도착한 요청이 균등한 분포라고 가정을 하기 때문에 계산이 약간 느슨하긴 하지만 심각한 문제는 아니다.

CloudFare의 실험에 따르면 40억 개의 요청 가운데 버려지는 요청은 0.003%에 불과하다.

상세 설계

1. 처리가 제한된 요청들은 어떻게 처리하는가?

우선, 클라이언트는 429 - Too Many Requests 응답을 받는다.

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

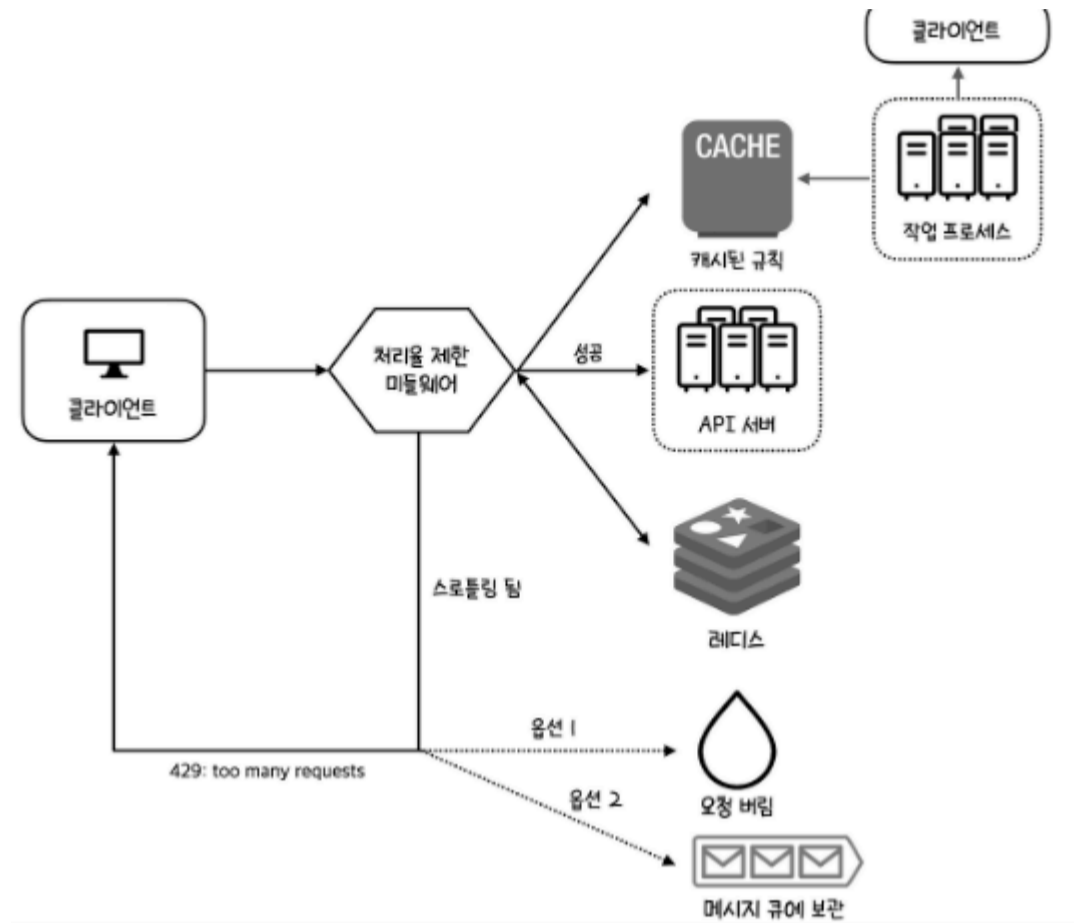
- ex. 주문 시스템에 과부하가 걸려 한도 제한이 걸린 경우에는, 해당 주문들을 보관했다가 나중에 처리할 수 있다.

또한, 클라이언트가 처리율 제한에 걸렸음을 확인할 수 있는 방법으로 커스텀 헤더를 붙여주기도 한다.

헤더명	용도
X-Ratelimit-Remaining	윈도우 내에 남은 처리 가능 요청의 수
X-Ratelimit-Limit	매 윈도우마다 클라이언트가 전송할 수 있는 요청의 수
X-Ratelimit-Retry-After	한도 제한에 걸리지 않으려면 몇 초 뒤에 요청들을 다시 보내야 할지에 대한 알림

2. 처리율 제한 규칙은 어떻게 만들어지고 어디에 보관되는가?

일반적으로 처리율 제한 규칙은 디스크에 보관한다. 이를 포함한 서비스 아키텍처는 다음과 같다.



만약, 마이크로 서비스를 구현하는 경우에는 여러 대의 서버와 병렬 스레드를 지원하도록 시스템을 확장하도록 해야한다.

이때, 고려해야 하는 사항들에는 다음이 있다.

- 경쟁 조건 (Race Condition)
 - 락을 통해 해결할 수 있다
 - 루아 스크립트 (Lua Script) 혹은 Redis 정렬 집합 (Sorted Set)을 활용할 수도 있다

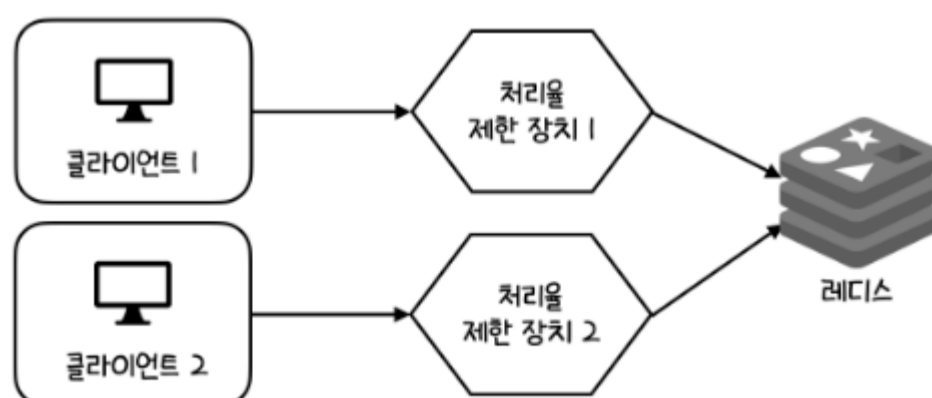
- 동기화 (Synchronization)

분산 환경에서 여러 대의 처리율 제한 장치 서버를 사용하는 경우, 동기화가 필요해진다.

이때, 고정 세션 (Sticky Session)을 적용하여 같은 클라이언트로부터의 요청을 같은 처리율 제한 장치로 보낼 수 있도록 할 수 있다.

- 다만, 고정 세션은 확장 가능성이 좋지 않다.

→ Redis와 같은 중앙 집중형 저장소를 많이 사용한다.



추가 학습 권장사항

1. 다양한 계층에서의 처리율 제한 방법

이번장은 7계층에서의 처리율 제한만 알아보았다.

이외에도 IP를 활용하는 3계층에서의 제한방법 등 다양한 방법이 존재하는 만큼 한번쯤 공부해보면 도움이 될 것이다.

2. 처리율 제한을 위해 클라이언트는 어떻게 설계하는 것이 바람직한가?
3. 클라이언트의 재시도 로직을 어떻게 구현하면 좋을까?