처리율 제한 장치의 설계

처리율 제한 장치

- 클라이언트 또는 서비스가 보내는 트래픽의 처리율을 제어하기 위한 장치
- 특정 기간 내에 전송되는 클라이언트의 요청 횟수를 제한

제어를 위한 장치는 어디에 설정할지

- 클라이언트는 위변조 가능
- 일반적으로 게이트웨이에 분포

초당 제어를 위한 알고리즘을 선택

- 토큰 버킷
  - 고정된 토큰의 개수로, 토큰을 몇개나 리필할지 고민해야함
  - 하지만, reset boundary에서 토큰이 재공급 될 때 burst 발생 가능
  - 이걸 방지하기 위해, 시간 대 별로 나눠서 토큰을 공급하면?
    - burst는 없지만.. 그만큼 처리시간이 느릴 것으로 예상
- 누출 버킷
  - 요청 처리율이 고정
  - FIFO 큐로 구현
  - 빈자리가 있는 경우 큐에 요청을 추가
  - 큐가 가득 차 있는 경우에는 새 요청을 버림
  - 지정된 시간마다 요청을 꺼내어 처리
- 고정 윈도우
  - window 위로 요청건수를 기록하여 window 요청 건수가 정해진 건수보다 크면 요청 처리 거부
  - reset boundary에서 burst 발생 가능
- 슬라이딩 윈도우
  - sliding window log
    - 로그를 남겨서 구현
    - 남긴 로그를 관리해야하기 때문에, 구현과 메모리 비용이 높음
  - sliding window counter
    - fixed window와 sliding window log를 보완
    - 요청건수에 대한 예측을 통해 가정하여 사용
    - 현재 시간의 요청 수 + 직전 시간의 요청 수 X 이동 윈도와 직전 1분의 겹치는 비율
- Rate Limit 알고리즘은 트래픽 패턴을 잘 분석한 다음 적합한 알고리즘을 선택해야 한다.
- 유료 서비스가 트래픽 체증에 민감해하지 않다면(관대한) Token Bucket 알고리즘을 선택하고 그 외에는 Fixed Wondow나 Sliding Window 알고리즘을 선택한다.

카운터는 어디에 배치할 것인가?

- RDB는 느리기 때문에 캐시에 저장해야 함.

- 캐시를 선택하는 이유는?
  - 메모리에서 동작해서 빠름
  - 만료 정책 설정 가능

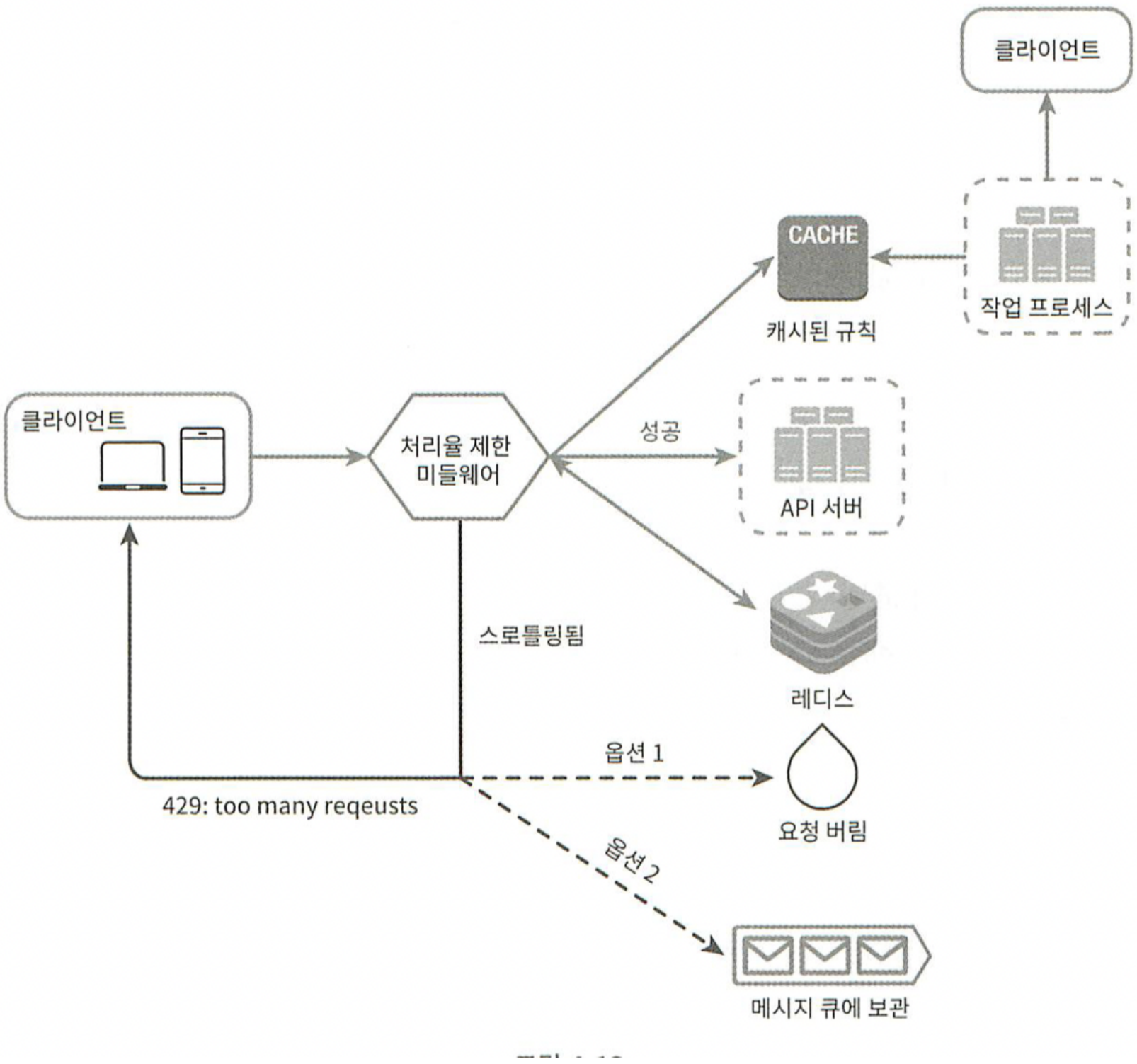
⇒ 클라이언트가 Rate limiter에게 요청을 보냄 → Rate limiter는 redis에서 카운터를 통해 제한을 넘었는지 확인 → 넘었다면 버리고, 아니면 API 서버에게 전달

처리율 한도 초과 트래픽 처리

- HTTP 429 (Too Many Requests) 를 클라이언트에게 전달
- 때에 따라서는, 한도 제한이 걸린 메시지를 나중에 처리하기 위해 큐에 보관할 수도 있음

처리율 제한 장치에서 사용하는 HTTP 헤더

- X-Ratelimit-Remaining : 윈도우에 남은 처리 가능한 요청 수
- X-Ratelimit-Limit : 매 윈도우마다 클라이언트가 전송 가능한 요청 수
- X-Ratelimit-Retry-After : 한도 제한에 걸리지 않으려면 몇 초 뒤에 요청을 다시 보내야 하는지 알림



- 처리율 제한 규칙은 디스크에 보관한다. 작업 프로세스는 수시로 규칙을 디스크에서 읽어 캐시에 저장한다.

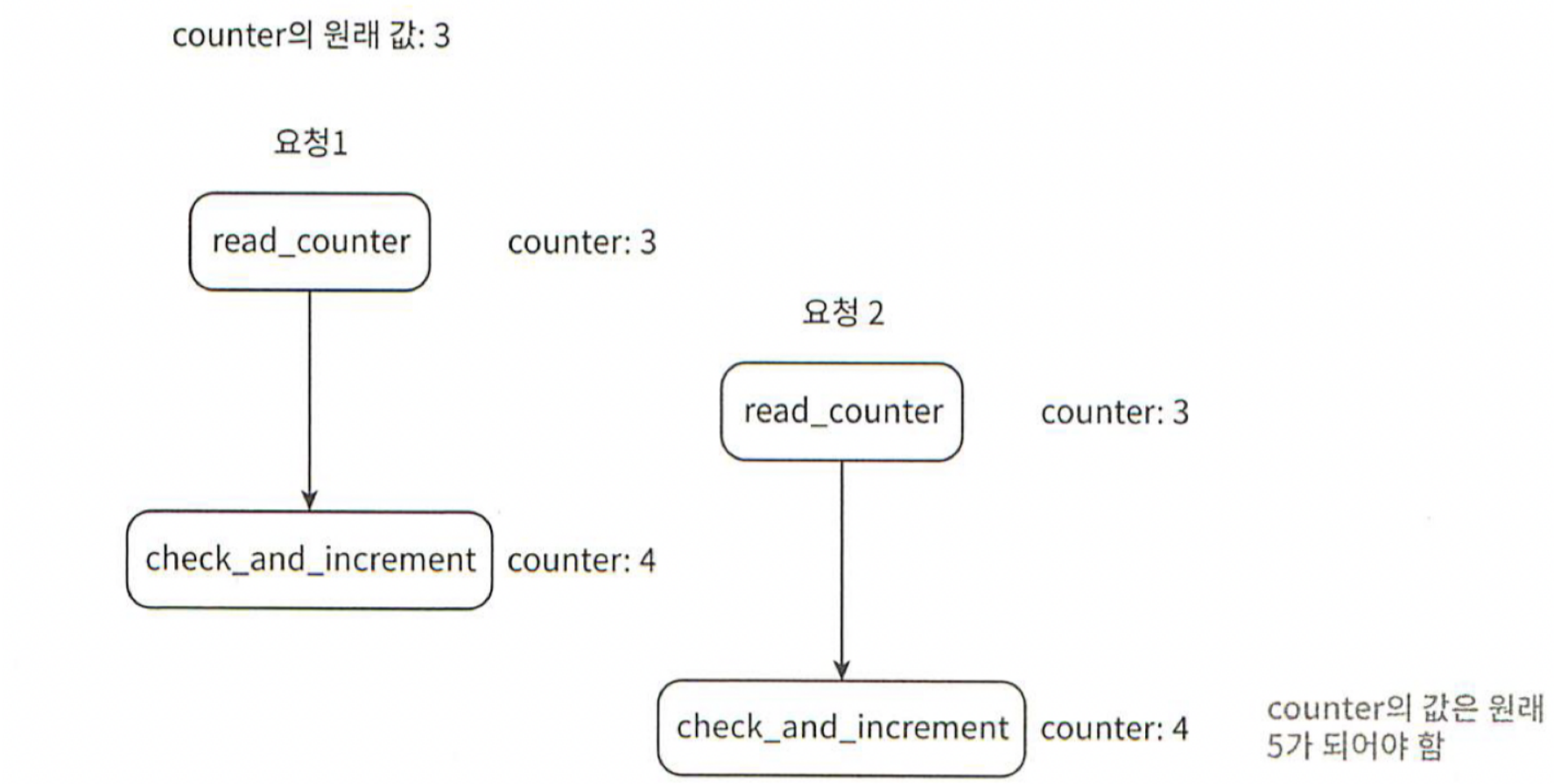
- 클라이언트가 요청을 서버에 보내면 요청은 먼저 처리율 제한 미들웨어에 도달한다.
- 처리율 제한 미들웨어는 제한 규칙을 캐시에서 가져온다.
  - 카운터 및 마지막 요청의 타임스탬프를 레디스 캐시에서 가져온다.
  - 가져온 값에 근거하여 해당 미들웨어는 다음과 같은 결정을 내린다.
    - 해당 요청이 처리율 제한에 걸리지 않은 경우에는 API 서버로 보낸다.
    - 해당 요청이 처리율 제한에 걸렸다면 429 too many requests 에러를 클라이언트에 보낸다.
      - 한편 해당 요청은 그대로 버릴 수도 있고 메시지 큐에 보관할 수 있다.

분산 환경에서 처리율 제한 장치의 구현

- Race condition
- Sychronization

경쟁 조건

- 레디스에서 카운터의 값을 read
- counter + 1 의 값이 임계치를 넘는지 확인
- 넘지 않는 다면, 레디스에 증가 된 값을 저장



- Lua script 혹은 sorted set 을 사용하여 해결
- lock을 거는 것은 비효율적

동기화 이슈

- Rate limiter 를 여러 개 두게 된다면, 동기화 이슈
- 각 Rate limiter 에 sticky session을 활용하는 방안도 있지만, 유연하게 사용 불가능하기 때문에 추천 X
- Redis와 같은 중앙 집중형 저장소 사용

성능 최적화

- 여러 데이터 센터를 사용하는 경우, 트래픽을 가장 가까운 엣지 서버로 전달하여 지연시간을 줄이자.
- 데이터를 동기화 할 때, 최종 일관성 모델을 사용하자.
    - 최종 일관성이란?
    - 데이터 변경이 발생했을때, 시간이 지남에 따라 여러 노드에 전파되면서 당장은 아니지만 최종적으로 일관성이 유지되는 것을 최종 일관성이라고 함.

모니터링

모니터링을 통해 처리율 제한 알고리즘과 규칙이 효과적인지 확인

- 너무 빡빡한 설정이라면, 많은 요청이 버려질 것이며 완화할 필요가 있음

경성 또는 연성 처리율 제한

- 경성 처리율 제한 : 요청의 개수는 임계치를 절대 넘어설 수 없다.
    - Leaky bucket
    - Sliding window log
- 연성 처리율 제한 : 요청 개수는 잠시 동안만 임계치를 넘어설 수 있다.
    - 다른 알고리즘들..

다양한 계층에서 처리율 제한

- 현재는 application 계층에서 처리율 제한을 알아보았음
- Iptable 등을 사용하면 IP 주소에 처리율 제한을 적용 가능

처리율 제한 회피

- 클라이언트 캐시를 통해 API 호출 횟수 감소
- 임계치를 통해, 적절한 호출 횟수
- 예외적 상황에서 gracefully한 복구가 가능하도록 설계
- 재시도시 충분한 텀을 두고 시도

예제

- A 회사는 B 회사의 API를 이용하여 서비스를 운영하고 있음.
- A 회사 서비스의 사용자가 갑자기 늘어나 B 회사에 갑자기 트래픽이 몰림.
- B 회사는 A 회사에게 1분 간격으로 **엄격하게** N번만 요청해달라고 부탁.

1. 이 때, 이걸 해결하려면 A 회사는 어떻게 해야할까?
2. 사용자가 A 회사 서비스를 사용 했을 때, 즉각적으로 결과를 받아야 한다면 어떻게 해야할까?
3. 반대로, 즉각적인 결과가 아니라 추후에 확인만 가능하면 되는 경우에는 어떻게 하는게 좋을까?

내가 생각한 답변

1. A → B 요청 사이에 Rate limiter를 두어, 특정 시간 동안 N번만 요청할 수 있도록 제한.
    - 이 때, 엄격하게 N번 요청을 해야하기 때문에 reset boundary에서 burst가 발생하지 않는 (즉, 경성 처리율 제한을 사용하는) 알고리즘을 선택해야 함.
2. 즉각적인 결과를 받아야한다면, Rate limit이 넘었을 경우 too many request를 사용자에게 “잠시후, 다시 요청해주세요..” 와 같은 응답을 보여주고 재시도 하는 방식으로.
3. 그게 아니라면, MQ에 담아 두었다가 재시도하는 방식으로 구현
    - 이 때, 사용자에게 재시도 완료 후 확인 가능한 푸시 알림이나 이메일 같은 것을 전달해줘도 좋을 것이고. (사용자가 일반 사용자라면..)
    - 기업 내 사용자나, 개발자라면 결과를 확인할 수 있는 trace id 같은 것을 전달해주기.