3주차

4장 처리율 제한 장치의 설계

처리율 제한 장치 : 클라언트 또는 서비스가 보내는 트래픽의 처리율을 제어하기 위한 장치

HTTP를 예로 들면 이 장치는 특정 기간내에 전송되는 클라이언트의 요청 횟수를 제한한다. API가 요청횟수가 제한 장치에 정의된 임계치를 넘어서면 추가로 도달한 모든 호출은 처리가 중단된다

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

API에 처리율 제한 장치를 두면 좋은 점

- DoS : DoS 공격은 네트워크, 호스트 혹은 다른 기반 구조의 요소들을 정상적인 사용자들이 사용할 수 없게 하는 것이다. 디도스는 분산된 도스 공격이라고 생각하면됨

- DoS 공격에 의한 자원 고갈을 방지
  - 트위터는 3시간 동안 300개의 트윗만 올릴 수 있도록 제한
  - 구글 독스 API는 사용자당 분당 300회의 read 요청만 허용
- 비용을 절감한다
  - 서버를 많이 두지 않아도 된다, 우선 순위가 높은 API에 더 많은 자원을 할당할 수 있다.
- 서버 과부하를 막는다.
  - 봇에서 오는 트래픽이나 사용자의 잘못된 이용 패턴으로 유발된 트래픽을 걸러낼 수 있다

1단계 : 문제 이해 및 설계 범위 설정

어떤 제한 장치를 만들 것인가?

- 클라이언트 측이 아닌 서버 측 API 를 위한 장치를 설계한다고 가정하자
- 다양한 형태의 제어 규칙을 정의할수 있는 유연한 시스템을 만들자
- 대규모 요청을 처리할 수 있어야한다
- 시스템은 분산환경에서 동작해야한다
- 독립된 서비스인지, 애플리케이션 코드에 포함 될지는 본인이 결정해라
- 요청이 거부당한 사용자에게 알려줘야한다

요구사항

- 설정된 처리율을 초과하는 요청은 정확하게 제한한다
- 낮은 응답시간 : 이 처리율 제한장치는 HTTP 응답시간에 나쁜 영향을 주어서는 곤란하다.
- 가능한 적은 메모리를 써야한다
- 분산형 처리율 제한 : 하나의 처리율 제한 장치를 여러 서버나 프로세스에서 공유할 수 있어야한다.
- 예외 처리 : 요청이 제한되었을 때는 그 사실을 사용자에게 분명하게 보여줘야한다
- 높은 결함 감내성 : 제한 장치에 장애가 생기더라도 전체 시스템에 영향을 주어서는 안된다.

2 단계 : 개략적 설계안 제시 및 동의 구하기

클라이언트-서버 통신 모델 사용하기

처리율 제한 장치는 어디에 둘 것인가?

클라이언트 측에 둔다면? 일반적으로 클라이언트는 처리율 제한을 안정적으로 걸 수 있는 장소가 못 된다. 클라이언트 요청은 쉽게 위변조가 가능해서다. 모든 클라이언트의 구현은 통제하는 것도 어려울 수 있다.

API 서버에 두는 대신 처리율 제한 미들 웨어를 만들어 해당 미들웨어로 하여금 API 서버로 가는 요청을 통제하도록 하는 것이다. - 추가 요청은 미들웨어에 의해 가로막히고 클라이언

트로 HTTP 상태코드 429가 반환된다.

클라우드 마이크로 서비스의 경우 API 게이트웨이에 구현한다 - API 게이트웨이는 처리율 제한, SSL 종단, 사용자 인증, IP 허용 목록 관리 등을 지원하는 완전 위탁관리형 서비스, 즉 클라우드 업체가 유지보수를 담당하는 서비스이다.

처리율 제한 장치를 어디에 두어야 하나?

프로그래밍 언어, 캐시서비스 등 현재 사용하고 있는 기술스택을 점검해서 현재 사용하는 언어가 서버측 구현을 지원하기에 효율이 충분한지 확인하라

서버측에 맞는 처리율 제한 알고리즘을 찾아라.

설계가 마이크로서비스에 기반하고 있고, 사용자 인증이나 IP 허용 목록관리등을 처리하기 위해 API 게이트웨이를 이미 포함 시켰다면 처리율 제한 기능또한 게이트웨이에 포함시켜야 한다.

처리율 제한 서비스를 직접 만드는 데는 시간이 걸리니 상용 API 게이트웨이를 쓰는 것이 바람직할 것이다

처리율 제한 알고리즘

- 토큰버킷
- 누출버킷
- 고정 윈도 카운터
- 이동윈도 로그
- 이동 윈도 카운터

토큰 버킷 알고리즘

- 아마존과 스트라이프가 API 요청을 통제하기 위해 이 알고리즘을 사용한다.

동작원리

- 토큰 버킷은 지정된 용량을 갖는 컨테이너이다. 이 버킷에는 사전 설정된 양의 토큰이 주기적으로 채워진다. 토큰이 찬 버킷에는 더이상 추가되지 않는다. 요청이 도착하면 버

킷에 충분한 토큰이 있는 지 검사하여 있는 경우에는 토큰을 하나 꺼내 요청을 처리하고 충분한 토큰이 없는 경우 해당 요청은 버려진다.

필요한 인자 :

- 버킷 크기 : 버킷에 담을 수 있는 토큰의 최대개수
- 토큰 공급률 : 초당 몇개의 토큰이 버킷에 공급되는 가

버킷은 몇개 사용해야하나?

- API 엔드포인트마다 별도의 버킷을 둔다. 예를 들어 사용자마다 하루에 한번만 포스팅을 할 수 있고, 친구는 150명까지 추가할 수 있고 좋아요 버튼은 다섯번까지만 누를 수 있다면 사용자마다 3개의 버킷을 두어여한다
- IP 주소별로 처리율 제한을 적용해야한다면 IP 주소마다 버킷을 하나씩 할당해야한다.
- 시스템의 처리율을 초당 10000개 요청으로 제한하고 싶다면, 모든 요청이 하나의 버킷을 공유하도록 해야할 것이다.

장점

- 구현이 쉽다
- 메모리 사용측면에서도 효율적이다
- 짧은 시간에 집중되는 트래픽도 처리 가능하다

단점

- 버킷 크기와 토큰 공급률 이라는 두개 인자를 갖고 있는데 이값을 적절하게 튜닝하는 것은 까다로운 일이다

누출 버킷 알고리즘

토큰 버킷 알고리즘과 비슷하지만 버킷대신 FIFO 큐를 사용해 요청 처리율을 고정시킨다

동작 원리

- 요청은 큐에 빈자리가 있는지 확인하고, 빈자리가 있으면 큐에 요청을 추가한다.
- 빈자리가 없으면 버린다
- 지정된 시간마다 큐에서 요청을 꺼내 처리한다

인자(parameter)

- **버킷 크기** : 큐의 사이즈
- **처리율(outflow rate)** : 일정 시간동안 몇 개의 요청을 처리할 것인가, 보통 초단위로 한다.

장점

- 큐의 크기가 제한되어 있어 메모리 사용이 효율적이다.
- 처리율이 고정되어 있어 요청을 안정적으로 처리할 수 있다.

단점

- 트래픽 버스트가 발생할 경우 최신 요청이 버려질 수 있다.
- 튜닝하기 까다롭다.

고정 윈도 카운터 알고리즘

동작 원리

- 시간(timeline)을 고정된 크기(window)로 나누고 윈도우마다 카운터(counter)를 0으로 초기화한다.
- 요청이 들어오면 윈도우의 카운터는 1씩 증가한다.
- 카운터의 값이 임계치에 도달하면 다음 윈도우가 열릴 때까지 새로운 요청을 버린다.

장점

- 메모리 효율이 좋다
- 이해하기 쉽다
- 특정한 트래픽 패턴을 처리하기에 적합하다.

단점

- 윈도우 경계 부분에 요청이 몰린다면 기대했던 시스템의 처리한도보다 많은 양의 요청을 처리하게 된다.

이동 윈도우 로깅 알고리즘

동작 원리

- 모든 요청의 타임스탬프를 로그에 추가한다. 버려진 요청도 로그에 남긴다.
- 로그를 살펴 일정 시간동안 들어온 요청이 임계치보다 높으면 앞으로 들어올 요청은 버린다.
- 로그는 레디스의 정렬 집합(sorted set)과 같은 캐시에 저장한다.

장점

- 매우 세밀하게 시스템 처리율을 조절한다.

단점

- 거부된 요청의 타임스탬프도 저장하기 때문에 메모리를 많이 쓴다.

이동 윈도 카운터 알고리즘

동작원리

- 고정 윈도우를 둔다.
- 다만, 현재 시점의 요청의 개수를 계산하는 방식을 `현재 요청 count + 이전 요청 count * 직전 1분(초)가 겹치는 비율` 으로 계산한다.
- 정확하지 않지만 내림해서 사용한다. - 유의미하다

장점

- 이전 시간대의 평균 처리율에 따라 현재 윈도우 상태를 계산하므로 짧은 시간에 몰리는 트래픽에도 잘 대응함.
- 메모리 효율이 좋음

단점

- 균등하게 분포되어 있다고 가정한 상태에서 추정치를 계산하기에 다소 느슨함 - 고정 윈도우 폭과 같은 문제가 발생할 수 있음
- 클라우드플레어에서 실시했던 실험을 기준으론 0.003% 불과함.

개략적인 아키텍처

- 카운터는 분산 시스템 카운터를 구현해야하므로 외부저장소에 저장해야하고, 디스크 기반 스토어보단 인메모리 기반 스토어가 처리량 측면에서 낫다
- 다양한 기능을 제공하는 redis가 좋다

3단계 상세설계

- 처리율 제한 규칙은 어떻게 만들어지고 저장되는가?
- 처리가 제한된 요청들은 어떻게 처리되는가?

처리율 제한 규칙은 lyft의 경우에는 오픈소스를 사용하는데 보통의 경우 설정파일로 저장해서 사용한다고 소개한다.

처리율 한도 초과 트래픽 처리

- 즉시적으로 429응답으로 내보내거나, 큐에 담아 나중에 처리할 수도 있다.
- 즉시 응답을 보낼 때에는 헤더에 여러 정보를 담아서 어떠한 제한 장치에 걸렸는지 체크할 수 있다.

다이어그램 같은 경우에는 미들웨어가 있고 위에 설명한 내용을 그대로 표현한 형태라 굳이 메모하지 않는다.

분산환경에서의 처리율 제한 장치 구현

경쟁조건

- 동작 과정 :

1. 레디스에서 카운터의 값을 읽는다
2. 카운터 + 1 의 값이 임계치를 넘는지 본다
3. 넘지 않는다면 레디스에 보관된 카운터 값을 1만큼 증가시킨다
    - 동시에 값을 읽어와 증가하고 저장하는 로직 (Read-Modify-Write)인 경우에는 Lost update가 발생할 수 있기 때문이다.
    - 락은 성능을 떨어트리고 분산환경에 적합하지 않다.
    - 루아스크립트(luascript)나 정렬집합(sorted set)과 같은 자료구조를 쓸 수 있다.

동기화 이슈

- 위에 언급한 문제로 redis(외부저장소)를 써야한다.
- 고정 세션은 유연하지도 않고 확장가능하지도 않다. 레디스를 사용하자(중앙집중형 데이터 저장소인)

성능 최적화

- multi region 서비스라면 어떻게 접근해야할까?
    - 우선 굉장히 먼 사용자를 위해서 사용자를 기준으로 가장 가까운 region(edge server) 서버를 두고 그곳으로 요청을 보내도록 하여 지연시간을 최소화할 수 있다.

모니터링

- 이러한 제한기가 올바르게 (의도된 대로) 작동하고 있는지 체크하기 위해서는 모니터링을 통해 확인해야한다.

4단계 마무리

공부하면 좋은 부분

- 경성 또는 연성 처리율 제한
    - 경성 처리율 제한 : 요청 개수는 임계치 절대 넘을 수 없다
    - 연성 처리율 제한 : 요청 개수는 잠시동안 임계치 넘을 수 있다
- 다양한 계층에서의 처리율 제한

    - 애플리케이션 (HTTP) 계층 이외에도 다른 계층에서도 처리율을 어떻게 제한하는지

- 처리율 제한을 회피하는 방법
    - 클라이언트 측 캐시를 사용하여 API 호출 횟수를 줄인다
    - 처리율 제한의 임계치 이해하여 메시지 빈도 조절
    - 예외, 에러처리 코드 도입하여 예외상황에서 안전하게 복구될 수 있게 함
    - 재시도 로직 구현 시 백오프 (알고리즘 또는 프로세스가 다른 조치를 취하지 않은 상황) 시간은 충분하게한다.