가상 면접 사례로 배우는

대규모 시스템 설계 기초

이정민

4. 처리율 제한 장치의 설계

개요

■ 처리율 제한 장치란?

- 클라이언트 또는 서비스가 보내는 트래픽의 처리율을 제어하기 위한 장치
- **API 요청 횟수 임계치 초과 시 호출 처리 중단시킴**
  - 사용자는 초당 2회 이상 새글을 올릴 수 없다.
  - 동일 IP에서 하루에 10개 이상 계정을 생성할 수 없다.
  - 동일 디바이스로 주당 5회 이상 리워드를 요청 할 수 없다.

■ 처리율 제한 장치를 사용하는 이유

- **DoS 공격에 의한 자원 고갈 방지**
  - 임계치 초과 요청에 대해서 처리를 중단함
- **비용 절감**
  - 추가 요청 처리 제한으로 서버 자원 절감, 우선 순위가 높은 API에 자원 할당
  - 제 3자(third-party) API 사용료 지불 시 과금 비용 절감
- **서버 과부하 방지**
  - 봇(bot) 혹은 사용자의 잘못된 이용 패턴으로 유바로딘 트래픽을 걸러냄

1단계) 문제 이해 및 설계 범위 확장

■ 요구사항 인터뷰

- **처리율 제한 장치의 위치 : 클라이언트 or 서버측?**
  ⇒ 서버측
- **API 호출 제어 기준 : IP or 사용자 ID?**
  ⇒ 다양한 형태의 제어규칙을 정의할 수 있고, 유연해야 함
- **시스템 규모 : 스타트업 or 사용자 많은 큰 기업?**
  ⇒ 대규모 요청을 처리 할 수 있어야 함
- **분산 환경 여부?** ⇒ 분산환경에서 작동해야 함
- **독립 서비스 or 애플리케이션 코드?** ⇒ 알아서
- **사용자 요청 처리율 제한에 걸러진 경우 사용자에게 알릴 필요성?** ⇒ 있음

■ 요구사항

- **설정된 처리율 초과하는 요청은 정확하게 제한함**
- **낮은 응답시간**
  - HTTP 응답시간에 나쁜 영향을 주면 안됨
- **가능한 적은 메모를 사용해야함**
- **분산형 처리율 제한**
  - 하나의 처리율 제한 장치를 여러 서버, 프로세스가 공유
- **예외처리**
  - 요청 제한 되었을 때 그 내용을 사용자에게 제공해야함
- **높은 결함 감내성(fault tolerance)**
  - 제한 장치에 장애가 생기더라도 전체 시스템에 영향 X

2단계) 개략적 설계안 제시 및 동의 구하기

- 처리율 제한 장치는 어디에 둘 것인가?
  - **클라이언트 측 : 처리율 제한 안정적 처리불가**
    - 쉬운 위변조, 모든 클라이언트의 구현 통제가 어려움
  - **서버 측**
    - API 서버
    - 미들웨어 : API 게이트웨이 (처리율 제한, SSL종단, 사용자인증, IP 허용목록 관리 등)
  - **고려사항**
    - 사용 기술 스택(프로그래밍 언어, 캐시 서비스 등) 점검
    - 적절한 처리율 제한 알고리즘 선택 (상용 솔루션 사용시 제한됨)
    - API 게이트웨이를 사용하는 경우 여기에 포함 시킬 수 있음
    - 직접 개발 보다 사용 제품을 쓰는 것이 바람직 할 수 있음

- 처리율 제한 알고리즘
  - **토큰 버킷(token bucket)**
  - **누출 버킷(leaky bucket)**
  - **고정 윈도 카운터(fixed window counter)**
  - **이동 윈도 로그(sliding window log)**
  - **이동 윈도 카운터(sliding window counter)**

[참고] 처리율 제한 알고리즘 상세 (1/5)

■ 토큰 버킷 알고리즘

- **동작원리**
  - 토큰 버킷은 지정된 용량을 갖는 컨데이너로, 사전 설정된 양의 토큰이 주기적으로 채워짐 (꽉차면 추가X)
  - 각 요청은 처리될 때 마다 하나의 토큰을 사용함
  - 요청이 오면 버킷에서 토큰을 꺼내 사용함
  - 토큰이 없는 경우 요청은 버려짐
  - API 엔드포인트 마다 별도의 버킷을 사용 (사용자 or IP)
  - 시스템의 처리율을 제한하고 싶다면 모든 요청이 하나의 버킷을 공유해야함
  - 파라미터 : 버킷 크기, 토큰 공급률(refill rate)
- **장점**
  - 구현이 쉽고, 메모리 사용이 효율적임
  - 짧은 시간에 집중되는 트래픽 처리 가능
- **단점**
  - 파라미터 튜닝이 까다로움

[참고] 처리율 제한 알고리즘 상세 (2/5)

■ 누출 버킷 알고리즘

○ **동작원리**

- 토큰 버킷 알고리즘과 유사하나 요청 처리율이 고정됨
- 요청이 오면 큐를 확인하여 빈자리가 있는 경우 추가
- 큐가 가득 차 있으면 새 요청은 버림
- 지정된 시간마다 큐에서 요청을 꺼내 처리
- 파라미터 : 버킷 크기, 처리율(outflow rate)

○ **장점**

- 큐의 크기가 제한되어 메모리 사용이 효율적임
- 고정된 처리율을 갖고 있기 때문에 안정적 출력이 필요한 경우 적합

○ **단점**

- 단시간에 많은 트래픽이 몰리는 경우 요청을 제때 처리하지 못하면 최신 요청이 버려짐
- 파라미터 튜닝이 까다로울 수 있음

[참고] 처리율 제한 알고리즘 상세 (3/5)

■ 고정 윈도 카운터 알고리즘

○ **동작원리**

- 타임라인을 고정된 간격의 윈도로 나누고 윈도마다 카운터를 붙임
- 요청 접 수 시 카운터의 값은 1씩 증가함
- 카운터의 값이 임계치에 도달하면 새로운 요청은 신규 윈도가 열릴 때 까지 버려짐

○ **장점**

- 메모리 효율이 좋고 이해하기 쉬움
- 윈도가 닫히는 시점에 카운터를 초기화하는 방식은 특정한 트래픽 처리하기에 적합함

○ **단점**

- 윈도 경계 부근에서 일시적으로 많은 트래픽이 몰리는 경우 한도 보다 많은 양의 요청을 처리하게 됨

[참고] 처리율 제한 알고리즘 상세 (4/5)

■ 이동 윈도 로깅 알고리즘

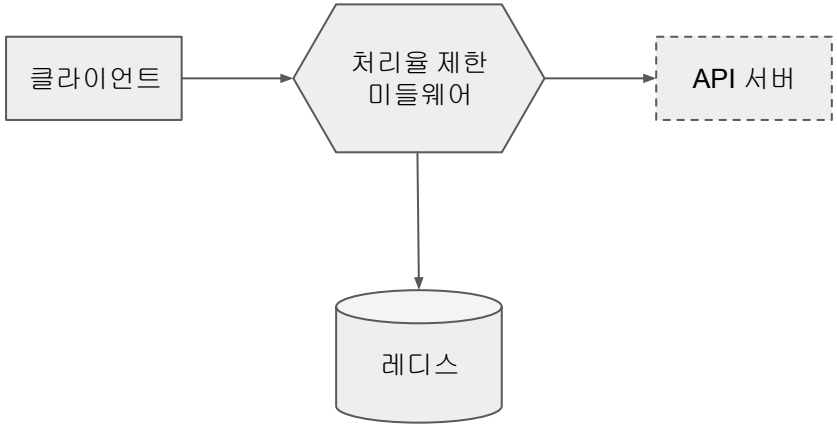
- **동작원리**
  - 요청의 타임스탬프를 추적함
  - 새 요청이 오면 만료된 타임스탬프는 제거함 (현재 윈도의 시작시점 보다 오래된 타임스탬프)
  - 새 요청의 타임스탬프를 로그에 추가함
  - 로그의 크기가 허용치보다 같거나 작으면 요청을 전달, 그렇지 않으면 처리를 거부
- **장점**
  - 허용되는 요청의 개수가 처리율 한도를 넘지 않음
- **단점**
  - 다량의 메모리 사용 (거부된 요청의 타임스탬프도 저장)

[참고] 처리율 제한 알고리즘 상세 (5/5)

- 이동 윈도 카운터 알고리즘
  - **동작원리**
    - 고정 윈도 카운터 + 이동 윈도 로깅 알고리즘 결합
  - **장점**
    - 짧은 시간에 몰리는 트래픽에 잘 대응함
    - 메모리 효율이 좋음
  - **단점**
    - 추정치를 기반으로 한도 보다 더 허용되거나 버려질 수 있음

2단계) 개략적 설계안 제시 및 동의 구하기

- 개략적인 아키텍처
  - **요청 접수 카운터를 추적 대상별로 관리**
    - 사용자, IP, API 엔드포인트, 서비스 단위?
    - 카운터 값이 한도를 넘어서면 한도를 넘은 요청은 거부
  - **카운터는 어디 보관할 것인가?**
    - 데이터베이스 vs **메모리 캐시** (빠르고 TTL 지원)
    - 레디스(Redis)는 메모리 기반 저장장치로서 처리율 제한 장치 구현 시 많이 사용함
  - **동작 원리**
    - 클라이언트가 처리율 제한 미들웨어에게 요청을 보냄
    - 처리율 제한 미들어는 지정 버킷에서 카운터를 가져와서 한도에 도달 했는지 여부를 검사
    - 한도 도달 시 요청 거부, 그렇지 않다면 요청은 API에 전달. 미들웨어는 카운터의 값을 증가 시킨 후 저장



3단계) 상세 설계 (1/3)

■ 처리율 제한 규칙

- **보통 설정 파일 형태로 디스크 저장**
- **예시)**

```
domain: messaing
descriptors:
  - key: message_type
    value: marketing
    rate_limit:
        unit: day
        requests_per_units: 5
```

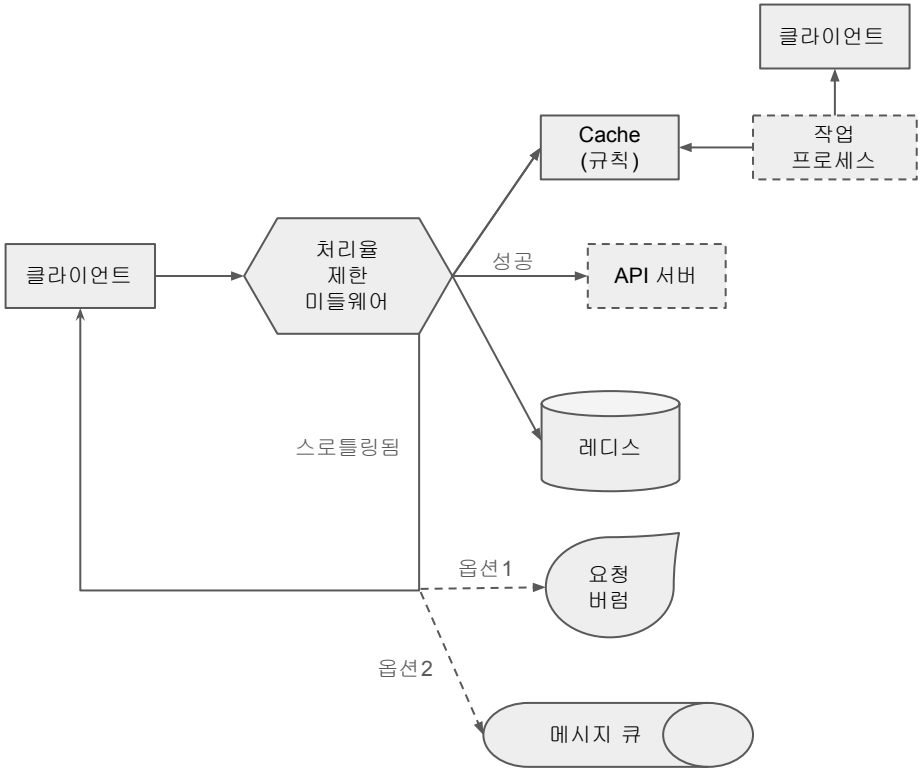
```
domain: auth
descriptors:
  - key: auth_type
    value: login
    rate_limit:
        unit: minute
        requests_per_units: 5
```

■ 처리율 한도 초과 트래픽의 처리

- **한도 제한 시 API는 HTTP 429 응답 전송**
  - 한도 제한 메시지를 보관하여 나중에 처리 할 수도 있음
- **HTTP 응답 헤더 정보 (예시)**
  - X-Ratelimit-Remaining: 윈도 내에 남은 처리 가능 요청수
  - X-Ratelimit-Limit: 매 윈도마다 클라이언트가 전송할 수 있는 요청 수
  - X-Ratelimit-Retry-After: 한도 제한에 걸리지 않으려면 몇 초 뒤에 다시 보내야 하는 지 알림

3단계) 상세 설계 (2/3)

- 상세 설계
  - **동작원리**
    - 처리율 제한 규칙은 디스크에 보관
    - 클라이언트 요청은 미들웨어가 먼저 처리
    - 미들웨어는 제한 규칙을 캐시에서 가져옴. 카운터 및 마지막 요청의 타임스탬프를 레디스 캐시에서 가져옴
    - 해당 요청이 처리율 제한에 걸리지 않은 경우 응답
    - 해당 요청이 처리율 제한에 걸린다면 429 too many request 에러를 클라이언트에 보냄.
    - 해당 요청은 버리거나 메시지 큐에 보관할 수도 있음



3단계) 상세 설계 (3/3)

- 분산 환경에서의 처리율 제한 장치의 구현
  - **경쟁 조건**
    - 레디스에서 카운터의 값을 읽음 (couter)
    - counter + 1의 값이 임계치를 넘는지 확인
    - 넘지 않는다면 레디스에 보관된 카운터 값을 1 증가
    - 병행성이 심한 환경에서 경쟁 조건 이슈가 발생할 수 있음 (요청1이 미완료 상태에서 요청2 병행 실행)
    - 해결책 : 락 (루아 스크립트, Redis 정렬집합)
  - **동기화 이슈**
    - 처리율 제한 장치가 여러대인 경우 동기화 필요
    - 고정 세션 (동일 클라이언트 요청은 같은 장치)
    - 중앙 집중형 데이터 저장소 (레디스)
  - **성능 최적화**
    - 에지서버 - 트래픽을 가장 가까운 에지 서버로 전달
    - 최종 일관성 모델 사용
  - **모니터링**
    - 채택된 처리율 제한 알고리즘이 효과적인가?
    - 정의한 처리율 제한 규칙이 효과적인가?

4단계) 마무리

■ 추가 고려사항

- **경성(hard) or 연성(soft) 처리율 제한**
- **다양한 계층에서의 처리율 제한**
  - 애플리케이션 계층 외 물리, 데이터링크, 네트워크, 전송, 세션, 프리젠테이션 계층에서의 처리
- **처리율 제한 회피 방법. 클라이언트 설계 방안**
  - 클라이언트 측 캐시를 사용하여 API 호출 회수를 줄임
  - 짧은 시간 너무 많은 메시지를 보내지 않도록 함
  - 예외나 에러 처리를 통해 우아하게(gracefully) 복구
  - 재시도(retry) 로직 구현 시 충분한 백오프(back-off) 시간을 둠