4장

1단계 문제 이해 및 설계 범위 확정

2단계 개략적 설계안 제시 및 동의 구하기

처리율 제한 알고리즘

토큰 버킷

필요 인자 : 버킷 크기, 토큰 공급률

사전에 설정된 양 만큼 토큰이 채워진다. 버킷이 가득 차면 추가 된 토큰은 버려진다. 요청이 도착한 경우에는 충분한 토큰이 없는 경우 요청을 버린다.

버킷은 엔드포인트마다 배치한다.

ex) IP 주소별 처리율 제한하는 경우 IP 별로 버킷 배치. 시스템의 처리율을 제한하고 싶다면 모든 요청이 하나의 버킷 공유하도록 설계.

구현이 쉽고 메모리 사용 측면에서 효율적이나 버킷 크기와 공급률을 적절하게 튜닝하는 것은 까다롭다.

누출 버킷

필요 인자 : 버킷 크기, 처리율

FIFO 큐로 구현되는 알고리즘

1. 요청이 도착하면 큐가 가득 차 있는지 확인
2. 빈 자리가 있으면 큐에 요청 추가
3. 가득 차 있으면 새 요청 버림
4. 지정된 시간마다 큐에서 요청을 꺼내 처리

메모리 사용량 측면에서 효율적이고 안정적인 출력이 필요한 경우에 적합하다. 단시간에 많은 트래픽이 몰리는 경우 최신 요청들이 버려지는 상황이 발생.

고정 윈도 카운터

1. 타임라인을 고정된 간격의 윈도로 나누고, 각 윈도마다 카운터를 붙인다.
2. 요청이 접수될 때마다 이 카운터의 값은 1씩 증가한다.
3. 이 카운터의 값이 사전에 설정된 임계치에 도달하면 새로운 요청은 새 윈도가 열릴 때까지 버려진다.

메모리 효율이 좋고 이해하기 쉽지만 일시적으로 많은 트래픽이 몰리는 경우 기대했던 시스템의 처리 한도보다 많은 양의 요청을 처리하게 된다.

이동 윈도 로깅

1. 요청의 타임스탬프를 추적한다.
2. 새 요처잉 오면 만료된 타임스탬프는 제거한다.
3. 새 요청의 타임스탬프를 로그에 추가한다.
4. 로그의 크기가 허용치보다 같거나 작으면 요청을 시스템에 전달한다. 그렇지 않은 경우 처리를 거부한다.

메커니즘이 정교해서 시스템의 처리율 한도를 넘지 않는다. 하지만 다량의 메모리를 사용한다는 단점이 있다.

이동 윈도 카운터

고정 윈도 카운터 알고리즘과 이동 윈도 로킹 알고리즘을 결합.

요청 개수 판별 수식 : 현재 1분 간의 요청 수 + 직전 1분 간의 요청 수 * 이동 윈도와 직전 1분이 겹치는 비율

이전 시간대의 평균 처리율에 따라 현재 윈도의 상태를 계산하므로 짧은 시간에 몰리는 트래픽에도 대응하기 좋다.

개략적 아키텍처

카운터를 메모리에 보관할 때 쓰이는 명령어

- INCR : 메모리에 저장된 카운터의 값을 1 증가시킴
- EXPIRE : 카운터에 타임아웃 값을 설정함. 설정된 시간이 지나면 카운터는 자동으로 삭제된다.

3단계 상세 설계

처리율 한도 초과 트래픽의 처리

HTTP 응답 헤더를 통해 처리율 제한에 걸리고 있는지 여부를 클라이언트에게 전달한다.

분산 환경에서의 처리율 제한 장치의 구현

경쟁 조건

a. 값이 동기화 되지 않는 경우를 고려해 락(lock)을 사용한다.

b. 다만 락(lock)의 경우 성능이 떨어지므로 루아 스크립트나 레디스 자료구조를 사용한다.

동기화 이슈

a. 고정 세션을 활용하여 같은 클라이언트로의 요청은 항상 같은 처리율 제한 장치로 보낸다.

b. 중앙 집중형 데이터 저장소를 사용한다. (ex. 레디스)

성능 최적화

a. 물리적으로 데이터 센터와의 거리를 좁힌다.

b. 장치 간 데이터 동기화에서 일관성 모델을 사용한다.

모니터링

a. 채택된 처리율 제한 알고리즘이 효과적인지 판단.

b. 정의한 처리율 제한 규칙이 효과적인지 판단.

4단계 마무리

추가적으로 언급하면 좋을 내용

- 경성 또는 연성 처리율 제한
- 다양한 계층에서의 처리율 제한
- 처리율 제한을 회피하는 방법