3장 시스템 설계 면접 공략법

효과적인 면접을 위한 4단계 접근법

1단계 - 문제 이해 및 설계 범위 확장

- 문제를 답할 때에 생각 없이바로 답을 내지 말고 깊이 생각하고 질문하여 요구 사항과 가정들을 분명히 하여야 한다.
- 올바른 질문을 하고, 적절한 가정을 하며, 시스템 구축에 필요한 정보를 모아야 한다. 아래와 같은 질문들을 해볼 수 있다.
    - 구체적으로 어떤 기능을 만들어야 하나?
    - 제품 사용자 수는 얼마나 되나?
    - 회사의 규모는 얼마나 빨리 커지리라 예상하나?
    - 회사가 주로 사용하는기술 스택은 무엇인가? 설계를 단순화하기 위해 활용할 수 있는 기존 서비스로는 어떤 것들이 있는가?

2단계 - 개략적인 설계안 제시 및 동의 구하기

- 설계안에 대한 최초 청사진을 제시하고 의견을 구하라. 면접관을 마치 팀원인 것처럼 대하라.
- 클라이언트, API, 웹 서버, 데이터 저장소, 캐시, CDN, 메시지 큐 등의 핵심 컴포넌트를 포함하는 다이어그램을 직접 그려라.
- 최초 설계안이 시스템 규모 관련 제약사항을 만족하는지 개략적으로 계산해보고 그 과정은 소리 내어 설명하라. 이런 추정이 필요한지는 면접관한테 미리 물어보자.
- 가능하면 시스템의 구체적인 사용 사례도 몇 가지 살펴보자. 계략적인 설계안을 잡아나가고 미처 고려하지 못한 에지 케이스를 찾는데 도움이 된다.
- API 엔드포인트나 데이터베이스 스키마도 보여야 하는지는 질문에 따라 다르다. 면접관의 의견을 물어보자.

예제 - 뉴스 피드 시스템 설계

이 설계는 두 가지 처리 플로 (flow)로 나눠 생각할 수 있다. 피드 발행 (feed publishing)과 피드 생성 (feed building)이다.

- 피드 발행 : 사용자가 포스트를 올리면 관련된 데이터가 캐시/데이터베이스에 기록되고, 해당 사용자의 친구 뉴스 피드에 뜨게 된다.
- 피드 생성 : 어떤 사용자의 뉴스 피드는 해당 사용자 친구들의 포스트를 시간 역순으로 정렬하여 만든다.

3단계 - 상세 설계

- 이 단계에 왔으면 면접관과 다음 목표를 달성했을 것이다.
    - 시스템에서 전반적으로 달성해야 할 목표와 기능 범위 확인
    - 전체 설계의 개략적 청사진 마련
    - 해당 청사진에 대한 면접관의 의견 청취
    - 상세 설계에서 집중해야 할 영역들 확인
- 이제 해야할 것은 설계 대상 컴포넌트 사이의 우선순위를 정하는 것이다. 대부분의 경우 면접관은 특정 시스템 컴포넌트들의 세부 사항을 깊이 있게 설명할 것을 원한다.
    - 단축 URL 생성기 설계에 관한 문제라면 그 해시 함수의 설계를 구체적으로 설명하는 것을 듣고 싶을 것이다
    - 채팅 시스템에 관한 문제라면 어떻게 지연시간 (latency)를 줄이고 사용자의 온/오프라인 상태를 표시할 것인지 듣고자 할 것이다.
- 면접 시에는 시간 관리에도 신경써야 한다.
    - 불필요한 세부사항에 시간을 쓰지 말자.
        - 페이스북에서 뉴스 피드의 순위를 매기는 데 사용되는 EdgeRank 알고리즘에 대해 이야기하는 것은 시간이 너무 많이 들고, 규모 확장 가능한 시스템을 설계할 수 있다는 능력을 보이는 데에 도움이 되지 않는다.

예제 - 뉴스 피드 시스템의 개략적 설계 마친 상황

다음 두 가지를 더 깊이 탐구해야 한다.

1. 피드 발행
2. 뉴스 피드 가져오기

4단계 - 마무리

마지막 단계에서 면접관은 설계 결과물에 관련된 몇 가지 후속 질문을 던질 수도 있고 스스로 추가 논의를 진행할 수도 있다.

- 면접관이 시스템 병목 구간, 혹은 개선 가능 지점을 찾아내라고 질문하면 여러분의 설계가 완벽하거나 개선할 점이 없다는 답은 하지 말자. 개선할 점은 언제나 존재하고 여기서 여러분의 비판적 사고 능력을 보일 수 있다.
- 여러분이 만든 설계를 다시 요약해주는 것도 도움이 될 수 있다. 여러 해결책을 제시한 경우에는 특히 그렇다. 면접관의 기억을 환기시켜주는 효과가 있기 때문이다.
- 오류가 발생하면 무슨 일이 생기는지 (서버 오류, 네트워크 장애 등) 따지보면 흥미로울 것이다.
- 운영 이슈도 논의할 가치가 있다. 메트릭은 어떻게 수집하고 모니터링 할 것인가? 로그는? 시스템은 어떻게 배포해 (roll-out) 나갈 것인가?
- 미래에 닥칠 규모 확장 요구에 어떻게 대처할 것인지도 흥미로운 주제다. 백만 사용자를 충분히감당할 수 있는 서비스가 천만 사용자를 감당하려면 어떻게 해야 하는가?
- 시간이 좀 남았으면 필요하지만 다루지 못한 세부 개선사항들을 제안할 수 있다.

해야 할 것

- 질문을 통해 확인하라. 스스로 내린 가정이 옳다 믿고 진행하지 말자.
- 문제의 요구사항을 이해하라.
- 정답이나 최선의 답안 같은 것을 업다는 점을 명심하라. 스타트업을 위한 설계안과 수백만 사용자를 지원하는 중견 기업의 설계안이 같을리 없다.
- 면접관이 여러분의 사고 흐름을 이해할 수 있도록 하라. 면접관과 소통하라.
- 가능하다면 여러 해법을 함께 제시하라.
- 개략적인 설계에 면접관이 동의하면, 각 컴포넌트의 세부사항을 설명하기 시작하라. 가장 중요한 컴포넌트부터 설명해라.
- 면접관의 아이디어를 이끌어내라. 좋은 면접관은 여러분과 같은 팀원처럼 협력한다.
- 포기하지 말라.

하지 말아야 할 것

- 전형적인 면접 문제들에도 대비하지 않은 상태에서 면접장에 가지 말라.
- 요구사항이나 가정들은 분명히 하지 않은 상태에서 설계를 제시하지 말라.
- 처음부터 특정 컴포넌트의 세부사항을 너무 깊이 설명하지 말라. 개략적인 설명 후 세부적으로 나아가라.

- 진행 중 막혔다면, 힌트를 청해보자.
- 소통을 주저말라. 침묵 속에서 설계를 진행하지 말자.
- 설계안을 내놓는 순간 면접이 끝났다고 생각하지 말라. 면접관이 끝났다고 말하기 전까지는 끝난 것이 아니다. 의견을 일찍, 그리고 자주 구하라.

시간 배분

시스템 설계 면접은 매우 광범위한 영역을 다루므로 시간이 부족할 수 있다. 45분의 시간이 주어진다고 가정하고 각 단계에 어느 정도 시간을 쓰는 것이 좋을지 정리해봤다.

1단계 - 문제 이해 및 설계 범위 확장 : 10분

2단계 - 개략적 설계안제시 및 동의 구하기 : 10 ~ 15분

3단계 - 상세 설계 : 10 ~ 25분

4단계 - 마무리 : 3 ~ 5분

4장 처리율 제한 장치의 설계

네트워크 시스템에서 처리율 제한 장치 (rate limiter) 는 클라이언트 또는 서비스가 보내는 트래픽의 처리율 (rate) 을 제어하기 위한 장치이다. HTTP를 예로 들면 이 장치는 특정 기간 내에 전송되는 클라이언트의 요청 횟수를 제한한다. API 요청 횟수가 제한 장치에 정의된 임계치 (threshold) 를 넘어서면 추가로 도달한 모든 호출은 처리가 중단 (block) 된다.

API에 처리율 제한 장치를 두면 좋은 점

- DoS 공격에 의한 자원 고갈을 방지할 수 있다. 대형 IT 기업들이 공개한 대부분의 API는 어떤 형태로든 처리율 제한 장치를 갖고 있어 추가 요청에 대해서는 처리를 중단한다.
- 비용을 절감한다. 추가 요청에 대한 처리를 제한하면 서버를 많이 두지 않아도 되고, 우선 순위가 높은 API에 더 많은 자원을 할당할 수 있다. 특히 제3자 API에 사용료를 지불하는 회사들에게는 아주 중요하다.
- 서버 과부하를 막는다. 봇(bot) 에서 오는 트래픽이나 사용자의 잘못된 이용 패턴으로 유발된 트래픽을 걸러내는데 처리율 제한 장치를 활요할 수 있다.

1단계 - 문제 이해 및 설계 범위 확정

처리율 제한 장치를 구현하는 데는 여러 가지 알고리즘을 사용할 수 있는데, 각가 고유한 장단점을 가지고 있다. 면접관과 소통하면 어떤 제한 장치를 구현해야 하는지 분명히 알 수 있

다.

다음은 면접관과의 대화를 통하여 시스템 요구사항을 요약한 것이다.

- 설정된 처리율을 초과하는 요청은 정확하게 제한한다.
- 낮은 응답시간
  - 이 처리율 제한 장치는 HTTP 응답시간에 나쁜 영향을 주어서는 곤란하다.
- 가능한 한 적은 메모리를 써야 한다.
- 분산형 처리율 제한 (distributed rate limiting)
  - 하나의 처리율 제한 장치를 여러 서버나 프로세스에서 공유할 수 있어야 한다.
- 예외 처리
  - 요청이 제한되었을 때는 그 사실을 사용자에게 분명하게 보여주어야 한다.
- 높은 결함 감내성 (fault tolerance)
  - 제한 장치에 장애가 생기더라도 전체 시스템에 영향을 주어서는 안된다.

2단계 - 개략적 설계안 제시 및 동의 구하기

처리율 제한 장치는 어디에 둘 것인가?

- 클라이언트 측에 두면 요청을 쉽게 위변조 할 수 있으므로 서버 측에 두어야 한다.
- API 서버에 두는 대신 처리율 제한 미들웨어를 만들어 API 서버로 가는 요청을 통제한다.
  - 추가 요청은 미들웨어에 의해 가로막히고 클라이언트로 HTTP 상태코드 429가 반환된다.
- 클라우드 마이크로서비스의 경우 API 게이트웨이에 구현한다.
  - API 게이트웨이는 처리율 제한, SSL 종단 (termination), 사용자 인증 (authentication), IP 허용 목록 (whitelist) 관리 등을 지원하는 완전 위탁관리형 서비스이다. 즉 클라우드 업체가 유지 보수를 담당하는 서비스이다.
- 어디에 둘지에 대한 지침
  - 프로그래밍 언어, 캐시 서비스 등 현재 사용하고 있는 기술 스택을 점검하라. 현재 사용하는 언어가 서버 측 구현을 지원하기에 효율이 충분한지 확인하라.

- 사업 필요에 맞는 처리율 제한 알고리즘을 찾아라. third party 에서 제공하는 게이트웨이 사용시에는 선택에 제한이 있을 수 있다.
- 설계가 마이크로서비스에 기반하고 있고, 사용자 인증이나 IP 허용목록 관리 등을 처리하기 위해 API 게이트웨이를 이미 포함시켰다면 처리율 제한 기능도 게이트웨이에 포함시켜야 할 수 있다.
- 처리율 제한 서비스를 직접 만드는 데 시간이 든다. 구현하기에 충분한 인력이 없다면 상용 API 게이트웨이를 쓰는 것이 바람직하다.

처리율 제한 알고리즘

- **토큰 버킷 알고리즘 (token bucket)**

그림 4-5

- 간단하고 세간의 이해도도 높은 편이라 보편적으로 널리 이용되는 알고리즘.
- 동작 원리
  - 토큰 버킷은 지정된 용량을 갖는 컨테이너이다. 버킷에 사전 설정된 양의 토큰이 주기적으로 채워진다. 토큰이 찬 버킷에는 더 이상 추가되지 않는다.

        - 요청이 도착하면 버킷에 충분한 토큰이 있는지 검사하여 있는 경우, 토큰을 하나 꺼내 요청을 처리한다.
        - 버킷이 다 찼다면 해당 요청은 버려진다 (dropped)
    - 필요한 인자
        - 버킷 크기 : 버킷에 담을 수 있는 토큰의 최대 개수
        - 토큰 공급률 (refill rate) : 초당 몇 개의 토큰이 버킷에 공급되는가
    - 버킷은 몇 개를 사용해야 하나?
        - 통상적으로 API 엔드포인트마다 별도의 버킷을 둔다.
        - IP 주소별로 처리율을 제한해야 한다면 IP 주소마다 버킷을 하나씩 할당한다.
        - 시스템의 처리율을 초당 10,000개로 제한하고 싶다면, 모든 요청이 하나의 버킷을 공유하도록 해야한다.
    - 장점
        - 구현이 쉽다.
        - 메모리 사용 측면에서도 효율적이다.
        - 짧은 시간에 집중되는 트래픽 (burst traffic)도 처리 가능하다. 버킷에 남는 토큰이 있기만 하면 요청은 시스템에 전달될 것이다.
- **누출 버킷 알고리즘 (leaky bucket)**

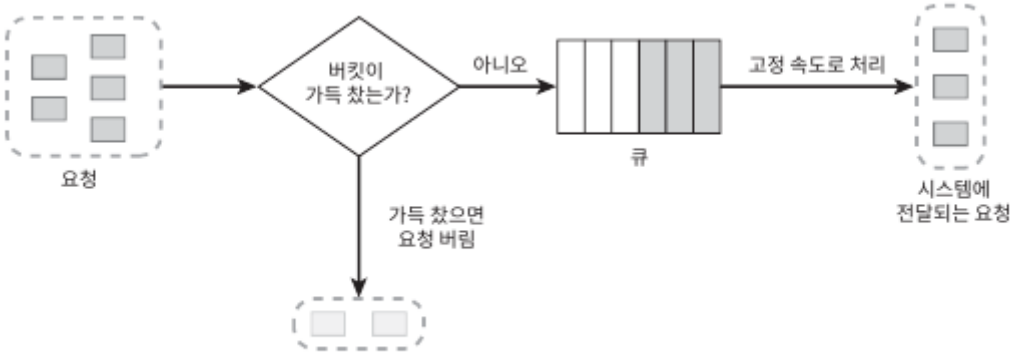


그림 4-7

    - 토큰 버킷과 비슷하지만 요청 처리율이 고정되어있다는 점이 다르다. 보통 FIFO 로 구현.
    - 동작 원리
        - 요청이 도착하면 큐가 가득찬지 본다. 아니라면 큐에 요청 추가.

        - 큐가 가득 찼다면 새 요청은 버린다.
        - 지정된 시간마다 큐에서 요청을 꺼내어 처리한다.
    - 필요한 인자
        - 버킷 크기 : 큐 사이즈와 같은 값.
        - 처리율 (outflow rate) : 지정된 시간당 몇 개를 처리할지 지정하는 값. 보통 초 단위.
    - 장점
        - 큐의 크기가 제한되어 있으 메모리 사용량 측면에서 효율적이다.
        - 고정된 처리율을 갖고 있기 때문에 안정적 출력 (stable outflow rate)이 필요한 경우 적합하다.
    - 단점
        - 단시간에 많은 트래픽이 몰리는 경우 큐에는 오랜 요청이 쌓이고 그 요청을 제때 처리 못하면 최신 요청들은 버려진다.
        - 두 개의 인자를 갖고 있는데, 이들을 올바르게 튜닝하기 어려울 수 있다.
- **고정 윈도 카운터 알고리즘 (fixed window counter)**

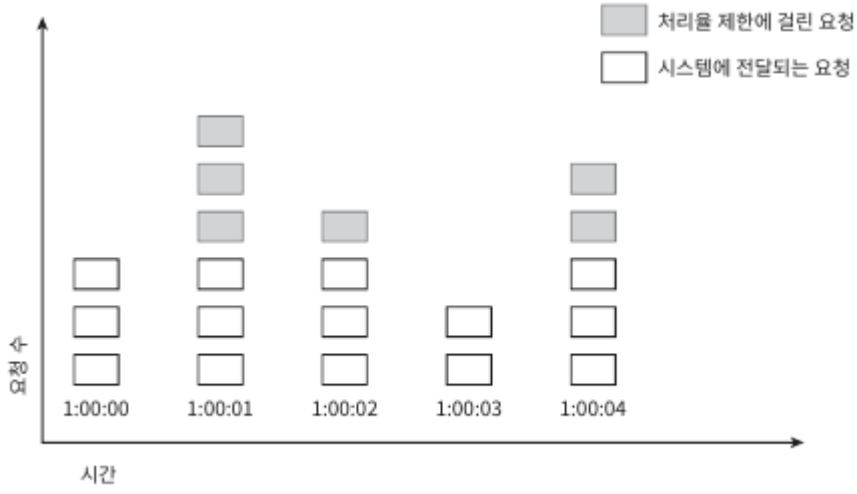
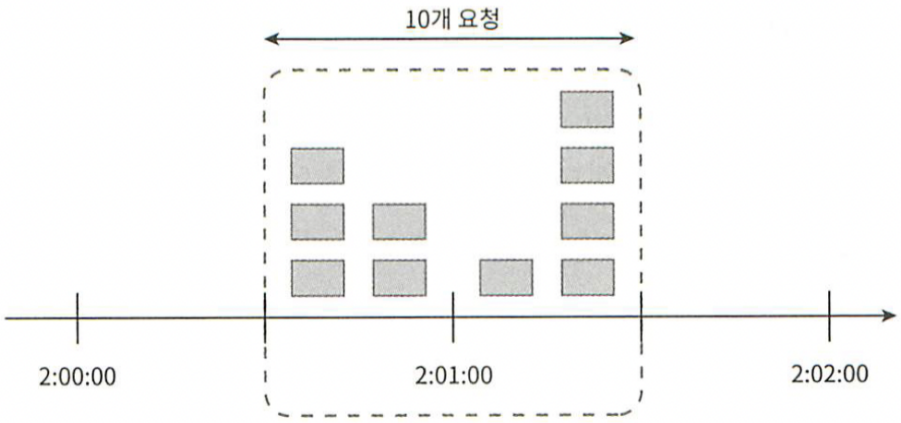


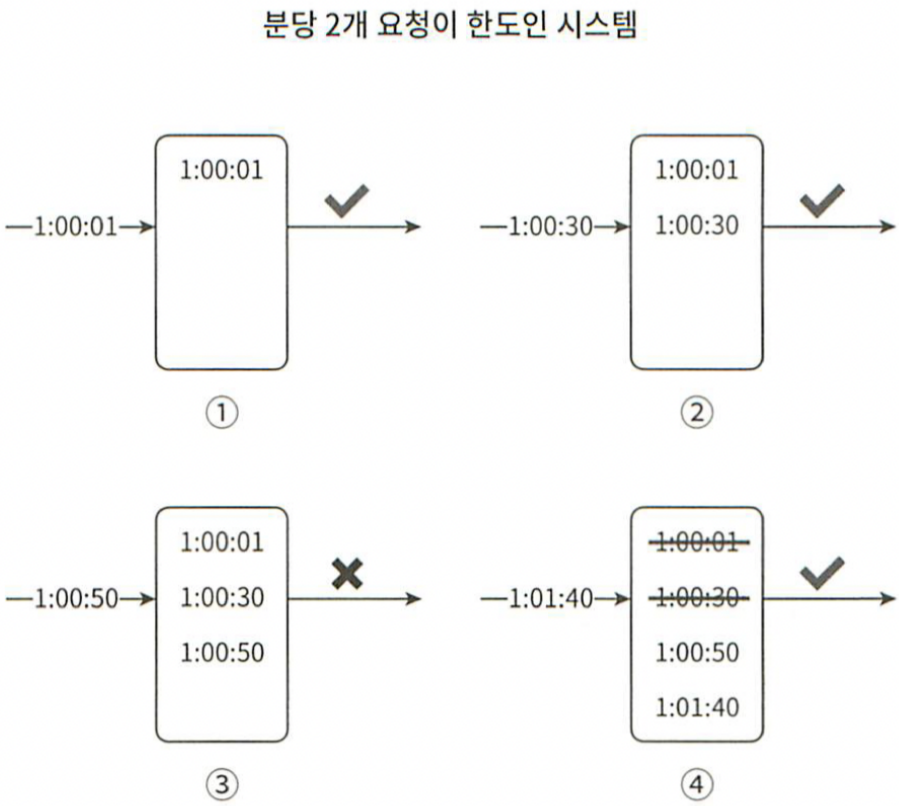
그림 4-8

    - 동작 원리
        - 타임라인을 고정된 간격의 윈도 (window) 로 나누고, 각 윈도마다 카운터 (counter) 를 붙인다.
        - 요청이 접수될 때마다 이 카운터 값은 1씩 증가한다.

        - 이 카운터 값이 사전에 설정된 임계치 (threshold) 에 도달하면 새로운 요청은 새 윈도가 열릴 때까지 버려진다.
    - 문제점
        - 윈도의 경계 부근에 순간적으로 많은 트래픽이 집중되면 윈도에 할당된 양보다 더 많은 요청이 처리될 수 있다.



        위의 시스템은 분당 최대 5개의 요청만 허용을 하고 카운터는 매분마다 초기화 된다. 그러나 2:00:30부터 2:01:30 사이의 1분 동안에는 허용 한도의 두 배인 10개의 요청을 처리하였다.
    - 장점
        - 메모리 효율이 좋다.
        - 이해가 쉽다.
        - 윈도가 닫히는 시점에서 카운터를 초기화하는 방식은 특정 트래픽 패턴을 처리하기에 적합하다.
    - 단점
        - 위에서 언급하였던 문제점의 사례처럼 처리 한도보다 많은 요청을 처리하게 될 수 있다.
- **이동 윈도 로깅 알고리즘 (sliding window log)**



- 동작 원리
  - 요청의 타임스탬프 (timestamp) 를 추적한다. 타임스탬프 데이터는 보통 레디스(Redis) 의 정렬 집합 (sorted set) 같은 캐시에 보관한다.
  - 새 요청이 오면 만료된 타임스탬프는 제거한다. 그 값이 현재 윈도의 시작 시점보다 오래되면 만료된 것이다.
  - 새 요청의 타임스탬프를 로그에 추가한다.
  - 로그의 크기가 허용치보다 같거나 작으면 요청을 시스템에 전달한다. 그렇지 않으면 처리를 거부한다.
- 장점
  - 처리율 제한 메커니즘이 아주 정교하다. 어느 순간에 윈도를 보더라도, 허용되는 요청의 개수는 시스템의 처리율 한도를 넘지 않는다.
- 단점
  - 거부된 요청의 타임스탬프도 보관하므로 다량의 메모리를 사용한다.

- **이동 윈도 카운터 알고리즘 (sliding window counter)**

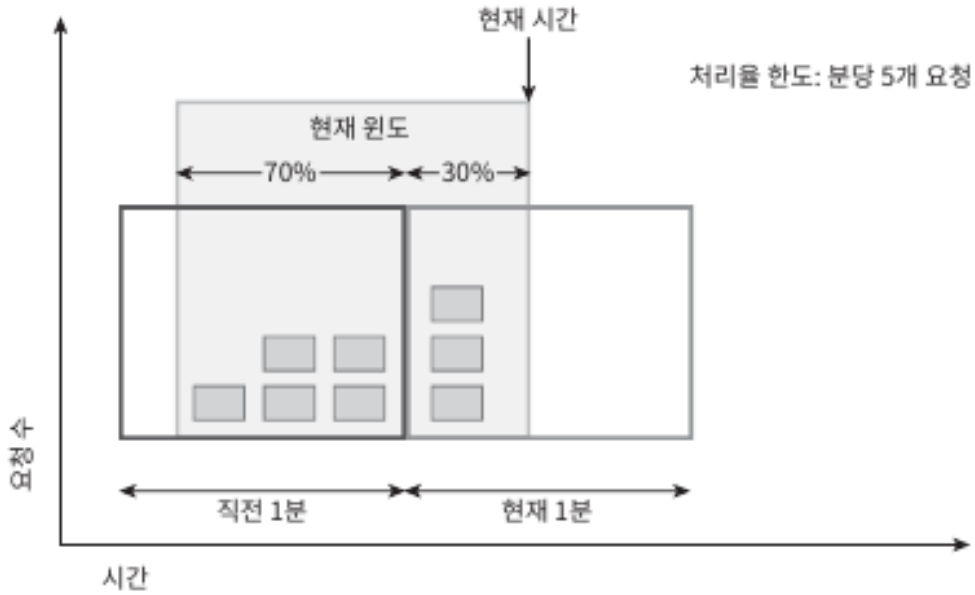


그림 4-11

- 고정 윈도 카운터 알고리즘과 이동 윈도 로깅 알고리즘을 결합한 것이다. 두 가지 방식으로 구현 가능하다.
- 현재 윈도우의 요청 개수
  - 현재 1분간의 요청 수 + 직전 1분간의 요청 수 x 이동 윈도와 직전 1분이 겹치는 비율
- 장점
  - 이전 시간대의 평균 처리율에 따라 현재 윈도 상태 계산하므로 짧은 시간에 몰리는 트래픽도 잘 대응.
  - 메모리 효율이 좋다.
- 단점
  - 직전 시간대에 도착한 요청이 균등하게 분포되어 있다고 가정하고 추정치 계산하므로 다소 느슨함. 하지만 실제 실험에서는 버려진 요청이 0.003%에 불과했으므로 심각X.

개략적인 아키텍처

- 카운터를 어디에 보관할 것인가?
  - DB는 디스크 접근 때문에 느리니까 안된다.
  - 메모리 상에서 동작하는 캐시가 좋다. 빠르고 시간 기반 만료 정책을 지원하므로.

- Redis 는 처리율 제한 장치를 구현할 때 자주 사용되는 메모리 기반 저장 장치이다. 두 가지 명령어를 지원한다.
  - INCR : 메모리에 저장된 카운터의 값을 1 증가시킴.
  - EXPIRE : 카운터에 타임아웃 값 설정. 설정 시간 지나면 카운터 자동 삭제됨.

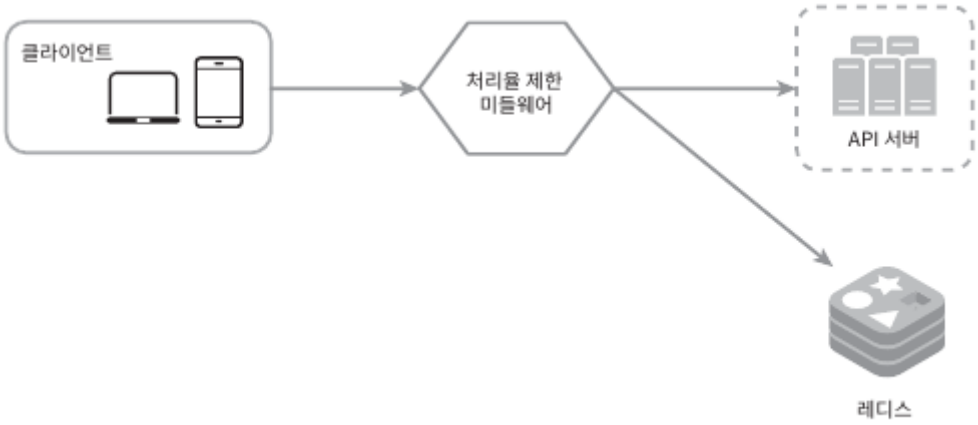
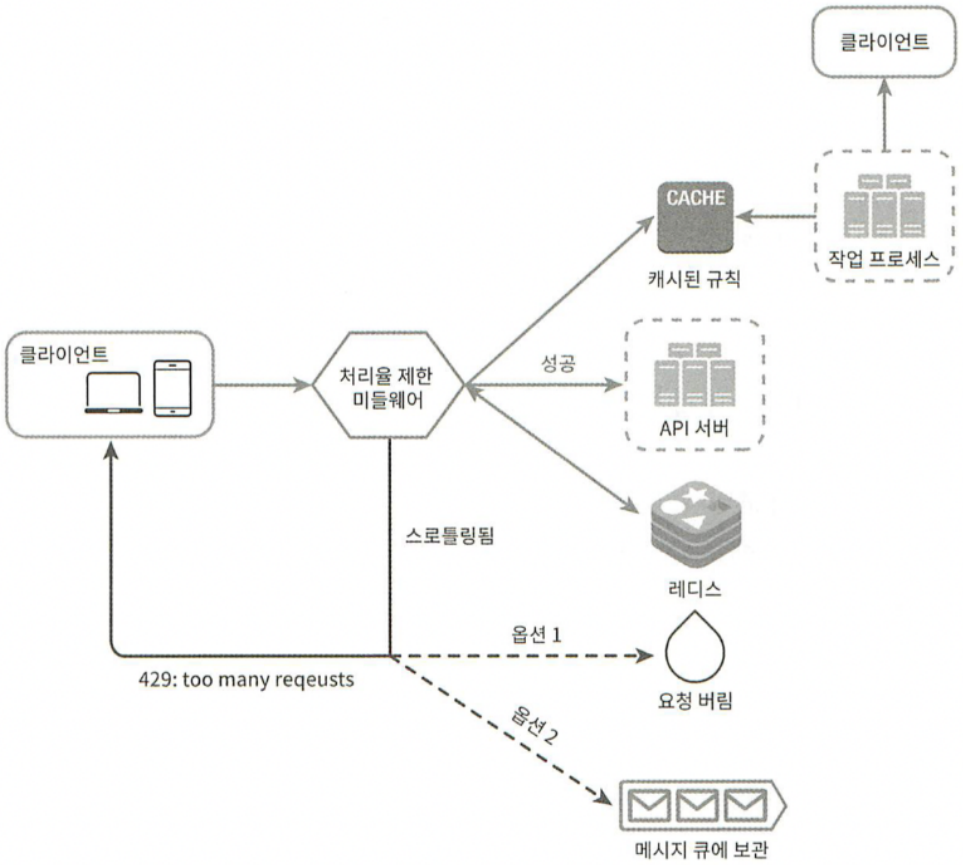


그림 4-12

- 처리율 제한 장치 동작 구조
  - 클라이언트가 처리율 제한 미들웨어에게 요청을 보냄.
  - 미들웨어는 레디스의 지정 버킷에서 카운터를 가져와서 한도 도달 여부 검사. 한도 도달시 요청 거부.
  - 도달하지 않았으면 요청이 API 서버로 전달됨. 미들웨어는 카운터 값 증가시키고 다시 레디스에 저장함.

3단계 - 상세 설계

- **처리율 한도 초과 트래픽 처리 (HTTP 헤더)**
  - X-Ratelimit-Remaining : 윈도 내에 남은 처리 가능 요청의 수.
  - X-Ratelimit-Limit : 매 윈도마다 클라이언트가 전송할 수 있는 요청의 수.
  - X-Ratelimit-Retry-After : 한도 제한에 안 걸리려면 몇 초 뒤에 요청 보내야 하는지 알림. 사용자가 너무 많은 요청 보내면 429 오류와 함께 보냄.
- **상세 설계**



    - 클라이언트가 서버로 요청 보내면 처리율 제한 미들웨어에 도착.
    - 미들 웨어는 제한 규칙을 가져오고 카운터 및 마지막 타임스탬프를 레디스 캐시에서 가져옴.
        - 요청이 제한 걸리지 않으면 서버로 전송.
        - 제한 걸리면 429 에러를 클라이언트에 보냄. 그 요청은 버리거나 메시지 큐에 보관.

- **분산 환경에서의 처리율 제한 장치 구현**
    - 경쟁 조건 (race condition)
        - 레디스에서 카운터 값을 읽은 뒤 +1 값이 넘지 않으면 더하고 저장.
        - 여러 스레드가 병렬로 카운터 값 읽고 업데이트 하는 과정에서 문제 발생.
        - 해결책 : 락 (시스템 성능 떨어뜨림),루아스크립트, 정렬집합 (레디스 자료구조).
    - 동기화 이슈
        - 웹은 무상태 계층이므로 처리율 제한 장치는 요청을 받지 않은 클라이언트의 상태 모름.

        - 고정세션 (sticky session) 을 활용하여 해결할 수 있다. 하지만 규모 확장 가능성 떨어뜨림.
- **성능 최적화**
    - 데이터 센터 위치를 고려하여 latency 관리
    - 제한 장치 간 데이터 동기화시 최종 일관성 모델 (eventual consistency model) 사용.
- **모니터링**
    - 채택된 처리율 제한 알고리즘이 효과적인지.
    - 정의한 처리율 제한 규칙이 효과적인지.

4단계 - 마무리

- **추가적으로 언급하면 좋은 것들**
    - 경성 (hard) 또는 연성 (soft) 처리율 제한
        - 경성 : 요청 개수가 임계치를 절대 넘을 수 없음.
        - 연성 : 요청 개수가 잠시 동안 임계치 넘을 수 있음.
    - 다양한 계층에서의 처리율 제한
        - 애플리케이션 계층 (7계층) 외에됴 Iptables 를 이용하여 IP 주소에 처리율 제한을 적용하여 3계층에서 처리율 제한 가능.
    - 처리율 제한을 회피하는 방법
        - 클라이언트 측 캐시를 사용하여 API 호출 줄이기.
        - 처리율 제한의 임계치에 맞게 짧은 시간 동안 너무 많은 메시지 보내지 않기.
        - 예외나 에러 처리하는 코드를 도입하여 클라이언트가 유연하게 복구될 수 있도록.
        - 재시도 (retry) 로직을 구현할 때에는 충분히 백오프 (back-off) 시간 두기.