11장 - 뉴스 피드 시스템 설계

- 뉴스 피드
  - ex) 페이스북

    홈 페이지 중앙에 지속적으로 업데이트되는 스토리들
    - 사용자 상태 정보 업데이트
    - 사진, 비디오, 링크 앱 활동(app activity),
    - 페이스북에서 팔로하는 사람들, 페이지, 좋아요(likes)
    - 등
  - 유사한 서비스 : 페이스북 뉴스 피드, 인스타그램 피드, 트위터 타임라인

1단계 : 문제 이해 및 설계 범위 확정

지원자: 모바일 앱을 위한 시스템인가요? 아니면 웹? 둘 다 지원해야 합니까?

면접관: 둘 다 지원해야 합니다.

지원자: 중요한 기능으로는 어떤 것이 있을까요?

면접관: 사용자는 뉴스 피드 페이지에 새로운 스토리를 올릴 수 있어야 하고,
친구들이 올리는 스토리를 볼 수도 있어야 합니다.

지원자: 뉴스피드에는 어떤 순서로 스토리가 표시되어야 하나요?
최신 포스트가 위에 오도록 해야 하나요?
아니면 토픽 점수 같은 다른 기준이 있습니까?
ex) 가까운 친구의 포스트는 좀 더 위에 배치해야 된다거나 하는?

면접관: 단순히 시간 흐름 역순(reverse chronological order)으로 표시된다고 가정합니

지원자: 한 명의 사용자는 최대 몇 명의 친구를 가질 수 있습니까?

면접관: 5,000명 입니다.

지원자: 트래픽 규모는 어느 정도됩니까?

면접관: 매일 천만 명이 방문한다고 가정합시다. (10million DAU)

지원자: 피드에 이미지나 비디오 스토리도 올라올 수 있습니까?

면접관: 스토리에는 이미지나 비디오 등의 미디어 파일이 포함될 수 있습니다.

2단계 : 개략적 설계안 제시 및 동의 구하기

- `피드 발행(feed publishing)`과 `뉴스 피드 생성(news feed building)`의 두 부분으로 나눠 설계
  - **피드 발행**
    - 사용자가 스토리를 포스팅하면 해당 데이터를 캐시와 데이터베이스에 기록
    - 새 포스팅은 친구의 뉴스피드에도 전송
  - **뉴스 피드 생성**
    - 모든 친구의 포스팅을 시간 흐름 역순으로 모아서 만든다고 가정 (지면 관계상 다른 조건 생략)

뉴스 피드 API

- HTTP 프로토콜 기반으로 클라이언트가 서버와 통신하기 위해 사용하는 수단
  - 상태 정보를 업데이트하거나, 뉴스 피드를 가져오거나, 친구를 추가하는 등 다양한 작업을 수행
- 가장 중요한 두 가지 API는 다음과 같음
  - `피드 발행 API`, `피드 읽기 API`

피드 발행 API

- 새 스토리를 포스팅하기 위한 API
- `HTTP POST` 형태로 요청하면 됨

- ex) `POST /v1/me/feed`

인자:

- 바디(body) : 포스팅 내용에 해당
- Authorization 헤더 : API 호출을 인증하기 위해 사용

피드 읽기 API

- 뉴스 피드를 가져오는 API
- ex) `GET /v1/me/feed`

인자:

- Authorization 헤더 : API 호출을 인증하기 위해 사용

피드 발행

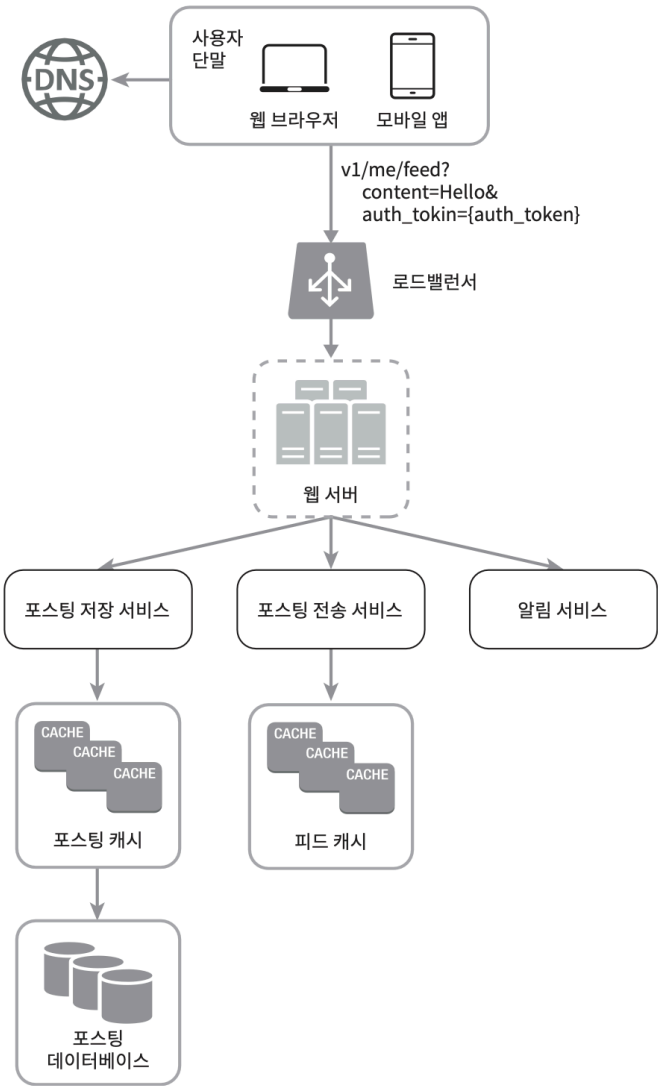

그림 11-2

- **사용자**
  - 모바일 앱이나 브라우저에서 새 포스팅을 올리는 주체. `POST /v1/me/feed API`를 사용
- **로드 밸런서(load balancer)**
  - 트래픽을 웹 서버들로 분산
- **웹 서버**
  - HTTP 요청을 내부 서비스로 중계하는 역할 담당
- **포스팅 저장 서비스(post service)**
  - 새 포스팅을 데이터베이스와 캐시에 저장
- **포스팅 전송 서비스(fanout service)**
  - 새 포스팅을 친구의 뉴스 피드에 푸시(push)한다.

- 뉴스 피드 데이터는 캐시에 보관하여 빠르게 읽어갈 수 있도록 한다.

- **알림 서비스(notification service)**
  - 친구들에게 새 포스팅이 올라왔음을 알리거나, 푸시 알림을 보내는 역할 담당

뉴스 피드 생성

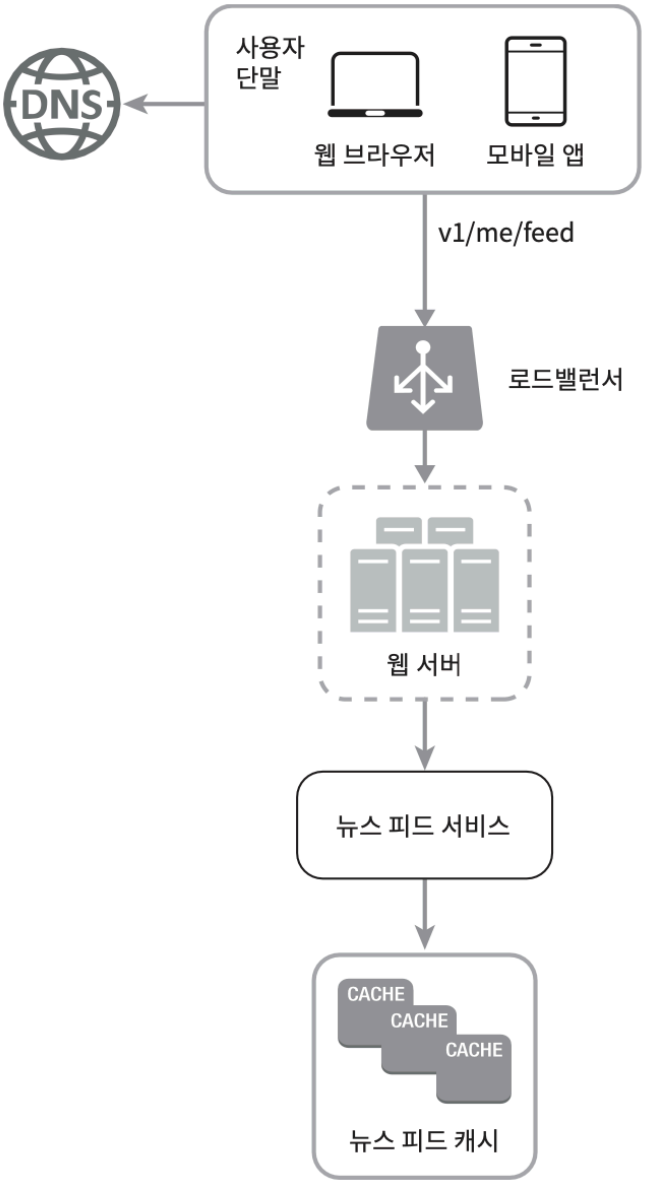

그림 11-3

- **사용자**
  - 뉴스 피드를 읽는 주체. `GET /v1/me/feed`를 이용

- **로드 밸런서**
  - 트래픽을 웹 서버들로 분산
- **웹 서버**
  - 트래픽을 뉴스 피드 서비스로 보냄
- **뉴스 피드 서비스(news feed service)**
  - 캐시에서 뉴스 피드를 가져오는 서비스
- **뉴스 피드 캐시(news feed cache)**
  - 뉴스 피드를 랜더링할 때 필요한 피드 ID를 보관

3단계 : 상세 설계

피드 발행 흐름 상세 설계

- `웹 서버`와 `포스팅 전송 서비스(fanout service)`에 초점을 맞춤

웹 서버

- `클라이언트 처리`를 포함한, `인증`, `처리율 제한` 등의 다양한 기능 수행
  - 올바른 인증 토큰을 Authorization 헤더에 넣고 API를 호출하는 사용자만 포스팅할 수 있어야 함
  - 특정 기간 동안 한 사용자가 올릴 수 있는 포스팅 수 제한 (스팸, 유해한 콘텐츠 방지)

포스팅 전송(팬아웃) 서비스

- 어떤 사용자의 새 포스팅을 그 사용자와 친구 관계에 있는 모든 사용자에게 전달하는 과정
- 팬 아웃은 다음의 두 가지 모델이 존재함
  - **쓰기 시점에 팬아웃(fanout-on-write, push 모델이라고도 함)**

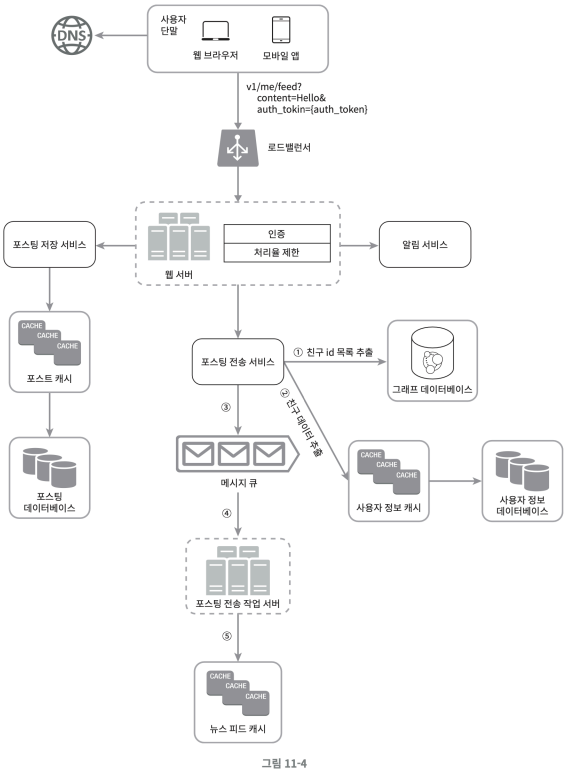

그림 11-4

- **읽기 시점에 팬아웃(fanout-on-read, pull 모델이라고도 함)**

쓰기 시점에 팬아웃하는 모델

- 새로운 포스팅을 **기록하는 시점**에 뉴스 피드를 갱신하게 됨
- 포스팅이 완료되면 바로 해당 사용자의 캐시에 해당 포스팅을 기록

장점

- 뉴스 피드가 실시간으로 갱신되며 친구 목록에 있는 사용자에게 즉시 전송
- 새 포스팅이 **기록되는 순간에 뉴스 피드가 이미 갱신**되므로(pre-computed) 뉴스 피드를 읽는 시간이 짧음

단점

- **친구가 많은 사용자의 경우(인플루언서 같은)** 친구 목록을 가져오고,
  그 목록에 있는 사용자 모두의 뉴스 피드를 갱신하는데 많은 시간이 소요될 수 있음 (**핫키, hotkey 문제 발생**)

- 서비스를 자주 이용하지 않는 사용자의 피드까지 갱신해야 하므로 컴퓨팅 자원 낭비

읽기 시점에 팬아웃 하는 모델

- 피드를 **읽어야 하는 시점**에 뉴드 피드를 갱신 - **요청 기반(on-demand) 모델**
- 사용자가 본인 홈페이지나 타임 라인을 로딩하는 시점에 새로운 포스트를 가져오게 됨

장점

- 비활성화된 사용자 또는 서비스에 거의 로그인하지 않은 사용자의 경우 이 모델이 유리
  - 로그인하기까지는 어떤 컴퓨팅의 자원도 소모하지 않음
- 데이터를 친구 각각에 푸시하는 작업이 필요 없으므로 핫키 문제도 생기지 않음

단점

- 뉴스 피드를 읽는 데 많은 시간이 소요될 수 있음

두 가시 방식을 결합

- 뉴시 피드를 빠르게 가져오는 것은 매우 중요하므로 **대부분의 사용자는 푸시 모델** 사용
- **친구**나 **팔로어(follower)**가 **아주 많은 사용자의 경우**에는
  팔로어로 하여금 해당 사용자의 포스팅을 필요할 때 가져가도록 하는
  풀 모델을 사용하여 시스템 과부하 방지

> 안정 해시를 통해 요청과 데이터를 보다 보르게 분산하여 핫키 문제를 줄여보자.

- 위 그림 11.4에서 팬아웃 서비스에 관한 부분만 보면 다음과 같다.

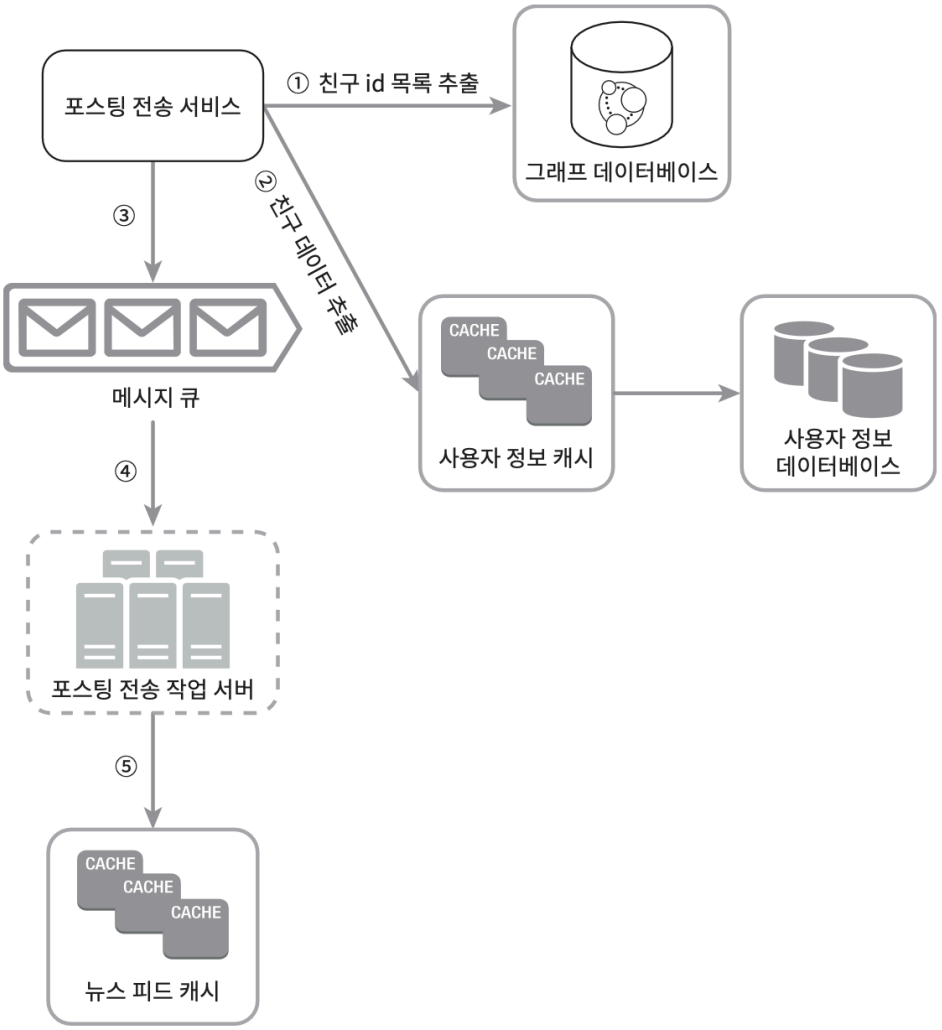

그림 11-5

팬아웃 서비스의 동작 과정

1. **그래프 데이터베이스에서 친구 ID 목록을 가져옴**
   - 그래프 데이터베이스는 친구 관계나 친구 추천을 관리하기 적합 ([2]를 읽어보자)
2. **사용자 정보 캐시에서 친구들의 정보를 가져옴**
   - 이후 사용자 설정에 따라 친구 가운데 일부를 걸러낸다.

     ex) 친구 중 누군가는 **피드 업데이트를 무시**할 수 있음 (친구는 유지되어도 새 스토리는 피드에 안 보임
     새로 포스팅된 스토리가
     일부 사용자에게만 공유되도록 설정된 경우에도 비슷한 일이 벌어짐
3. **친구 목록과 새 스토리의 포스팅 ID를 메시지 큐에 넣음**

4. **팬아웃 작업 서버가 메시지 큐에서 데이터를 꺼내 뉴스 피드데이터를 뉴스 피드 캐시에 넣음**
    - 뉴스 피드 캐시는 `<포스팅 ID, 사용자 ID>`의 순서쌍을 보관하는 매핑 테이블

post_id	user_id
post_id	user_id
post_id	user_id
post_id	user_id
post_id	user_id
post_id	user_id
post_id	user_id
post_id	user_id

그림 11-6

뉴스 피드 캐시의 매핑 테이블

        - 새로운 포스팅이 만들어질 때마다 위 그림처럼 레코드가 추가
        - **ID만 보관하는 이유**는 포스팅 정보 전부를 이 테이블에 저장할 경우 메모리 요구량이 너무 늘어남
        - 또는 **포스팅 전부를 메모리**에 올리는 방식으로 **캐시 크기에 제한을 거는 방식**도 있음
            - 어떤 사용자가 뉴스 피드에 올라온 수천 개의 스토리를 전부 읽는 경우는 극히 적음
            - 즉, 사용자는 `최신 스토리`를 원하고 이럴 경우 **캐시 미스(cache miss)의 확률은 지극히 적음**

피드 읽기 흐름 상세 설계

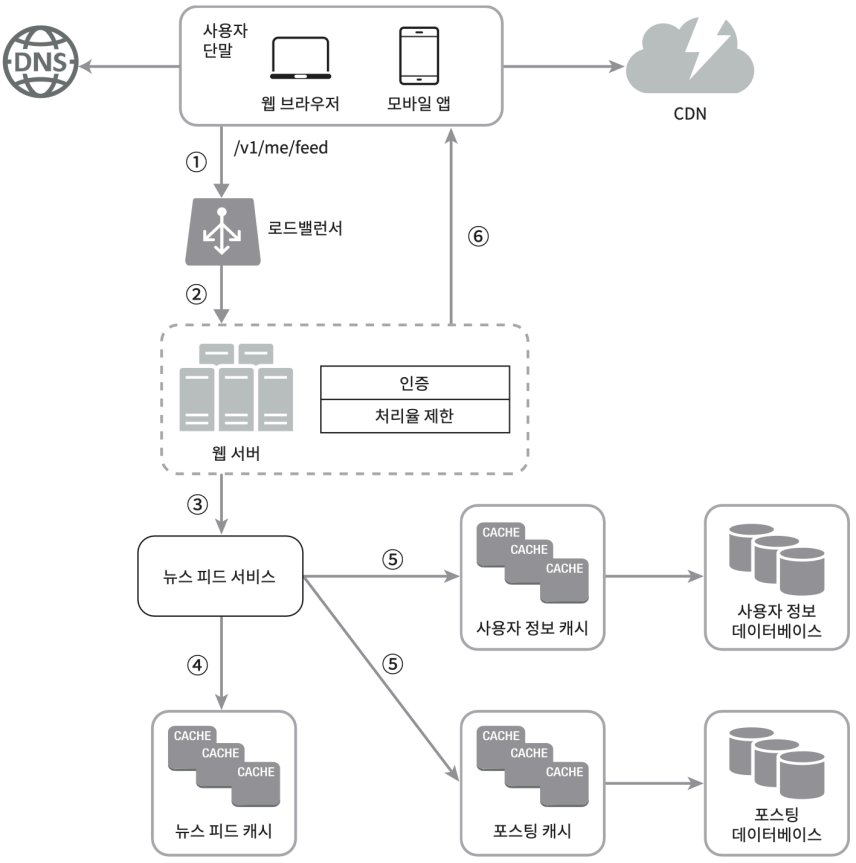

그림 11-7

뉴스 피드를 읽는 과정의 상세 설계안

- 이미지나 비디오 같은 미디어 콘텐츠는 CDN에 저장

클라이언트가 뉴스 피드 읽어가는 단계

1. 사용자가 뉴스 피드를 읽으라는 요청을 보냄. (사용자의 요청은 `/v1/me/feed` 로 전송)
2. 로드 밸런서가 요청을 쉡 서버 가운데 하나로 보냄
3. 웹 서버는 피드를 가져오기 위해 뉴스 피드 서비스를 호출
4. 뉴스 피드 서비스는 뉴스 피드 캐시에서 포스팅 ID 목록을 가져옴
5. 뉴스 피드에 표시할 사용자 이름, 사용자 사진, 포스팅 콘텐츠, 이미지 등을 사용자 캐시와 포스팅 캐시에서 가져와 완전한 뉴스 피드를 만듦
6. 생성된 뉴스 피드를 JSON 형태로 클라이언트에게 보냄. (클라이언트는 해당 피드 랜더링)

캐시 구조

- 캐시 자체는 **뉴스 피드 시스템의 핵심 컴포넌트**

그림 11-8

본 설계안의 경우 캐시를 다섯 계층으로 사용 가능

- 뉴스 피드 : 뉴스 피드의 ID 보관
- 콘텐츠 : 포스팅 데이터를 보관. (인기 콘텐츠는 따로 보관)
- 소셜 그래프 : 사용자 간 관계 정보 보관
- 행동(action) : 포스팅에 대한 사용자의 행위에 관한 정보 보관. (좋아요 , 답글 등등)
- 횟수(counter) : 좋아요 횟수, 응답 수, 팔로어 수, 팔로잉 수 등의 정보 보관

4단계 : 마무리

- 뉴스 피드 시스템을 뉴스 피드 발행 과 생성 두 부분을 구성
- 이 문제도 정답은 없음. 회사마다 독특한 제약 또는 요구조건이 있을 경우 언제든지 변경될 수 있음
    - 이때 설계를 진행하고 기술을 선택할 때
      선택하게 된
      배경에 어떤 타협적 결정등(trade-off)가 있었는지 이해하고, **설명할 수 있어야 함**
- 설계를 마친 후 시간이 남았다면 면접관과 규모 확장성 이슈를 논의하는 것도 좋음

다루면 좋을 만한 주제

데이터베이스 규모 확장

- 수직적 규모 확장 vs 수평적 규모 확장
- SQL vs NoSQL
- 주-부(master-slave) 다중화
- 복제본(replica)에 대한 읽기 연산
- 일관성 모델(consistency model)
- 데이터베이스 샤딩(sharding)

추가로 논의할 만한 주제

- 웹 계층(web tier)을 무상태로 운영하기
- 가능한 한 많은 데이터를 캐시할 방법
- 여러 데이터 센터를 지원할 방법
- 메시지 큐를 사용하여 컴포넌트 사이의 결합도 낮추기
- 핵심 메트릭(key metric)에 대한 모니터링
    - ex) 트래픽이 몰리는 시간대의 QPS, 사용자가 뉴스 피드를 새로고침(refresh) 할 때의 지연시간 등

추가) 면접에서 사용할 수 있는 유용한 규칙

📌 읽기가 많은 시스템의 경우- 캐시 사용을 고려

📌 쓰기가 많은 시스템의 경우 - 비동기 처리를 위해 Message Queue 사용

📌 낮은 지연시간을 요구하는 경우 - 캐시 및 CDN 사용을 고려

📌 ACID 원칙이 필요한 경우 - RDBMS/SQL DB로 이동

📌 비정형 데이터의 경우 - NoSQL DB로 이동

📌 복잡한 데이터(비디오, 이미지, 파일)의 경우 - Blob/Object 저장소로 이동

📌 복잡한 사전 계산이 필요한 경우 - Message Queue & 캐시를 이용

📌 대용량 데이터 검색 - 검색 인덱스, 트리 또는 검색 엔진을 고려

📌 SQL 데이터베이스 확장 - 데이터베이스 샤딩을 구현

📌 고가용성, 성능 및 처리량 - 로드 밸런서를 사용

📌 글로벌 데이터 전송 - CDN 사용을 고려

📌 그래프 데이터(노드, 엣지 및 관계가 있는 데이터) - 그래프 DB를 사용

📌 다양한 컴포넌트 확장 - 수평 확장을 구현

📌 고성능 데이터베이스 쿼리 - DB 인덱스 사용

📌 일괄 작업 처리 - 배치 및 Message Queue 사용

📌 서버 부하 관리 및 DOS 공격 방지 - Rate Limiter 사용

📌 마이크로서비스 아키텍처 - API Gateway 사용

📌 Single Point of Failure - 이중화 구현

📌 내결함성 및 내구성 - 데이터 복제 구현

📌 사용자 간의 빠른 통신 - Websocket 사용

📌 분산 시스템의 장애 감지 - Heartbeat 구현

📌 데이터 무결성 - Checksum Algorithm 사용

📌 효율적인 서버 확장 - Hashing 구현

📌 분산형 데이터 전송 - Gossip 프로토콜 고려

📌 위치 기반 기능 - Quadtree, Geohash 등을 사용

📌 특정 기술 이름 피하기 - 일반적인 용어 사용

📌 고가용성과 일관성 트레이드 오프 - 궁극적인 일관성

📌 IP 확인 및 도메인 이름 쿼리 - DNS 언급

📌 네트워크 요청에서 대용량 데이터 처리 - 페이지네이션 구현

📌 캐시 제거 정책 - LRU 캐시

📌 트래픽 급증 처리 - 오토스케일링 구현하여 리소스를 동적으로 관리

📌 분석이 필요할 때 - data lake 또는 전용 DB 사용 고려

📌 대규모 동시 연결 처리 - connection pooling을 사용하고 Protobuf를 사용하여 데이터 페이로드 최소화

출처 - https://ducktopia.tistory.com/112

▼ Chapter 11: DESIGN A NEWS FEED SYSTEM

1. How News Feed Works
2. Friend of friend recommendations Neo4j and SQL server
    - 25.02.20 - 여기로 변경