1️⃣

230913 1. 사용자 수에 따른 규모 확장성

1. 사용자 수에 따른 규모 확장성

> 💡 한 명의 사용자를 지원하는 시스템에서 시작하여, 최종적으로는 몇백만 사용자를 지원하는 시스템을 설계해 볼 것이다. 규모 확장성과 관련된 설계 문제를 푸는 데 쓰일 유용한 지식들을 마스터할 수 있을 것이다.

▼ 단일 서버

모든 컴포넌트(웹 앱, 데이터베이스, 캐시)가 단 한대의 서버에서 실행되는 간단한 시스템

사용자의 요청이 처리되는 과정

1. 사용자가 도메인을 이용하여 웹사이트 접속
    - 도메인 이름을 도메인 이름 서비스(DNS)에 질의하여 IP 주소로 변환하는 과정 필요
    - DNS는 제3 사업자가 제공하는 유료 서비스: 시스템의 일부가 아님
2. DNS 조회 결과로 IP 주소 반환
3. 해당 IP 주소로 HTTP(HyperText Transfer Protocol) 요청 전달
4. 요청을 받은웹 서버는 HTML 페이지나 JSON 형태의 응답을 반환

요청이 어디로부터 오는지

1. 웹 어플리케이션

- 비지니스 로직, 데이터 저장 등을 처리하기 위해서는 서버 구현용 언어(자바, 파이썬, ..)를 사용
- 프레젠테이션용으로는 클라이언트 구현용 언어(HTML, 자바스크립트, ..)를 사용

2. 모바일 앱
    - 모바일 앱과 웹 서버 간 통신을 위해서는 HTTP 프로토콜 이용
    - HTTP 프로토콜을 통해서 반환될 응답 데이터 포맷은 보통 JSON

▼ 데이터베이스

사용자가 늘면 서버 하나로는 충분하지 않음

웹/모바일 트래픽 처리와 데이터베이스로 분리

각각을 독립적으로 확장 가능

▼ 관계형 데이터베이스

관계형 데이터베이스 관리 시스템 / Relational Database Management System(RDBMS)

MySQL, Oracle Database, PostgreSQL

자료를 테이블과 열, 컬럼으로 표현

여러 테이블에 있는 데이터를 그 관계에 따라 join하여 합치기 가능

▼ 비관계형 데이터베이스

NoSQL

CouchDb, Neo4j, Cassandra, HBase, Amazon DynoDB

일반적으로 join을 지원하지 않음

NoSQL의 네 가지 분류

1. 키-값 저장소(key-value store)
2. 그래프 저장소(graph store)
3. 칼럼 저장소(column store)
4. 문서 저장소(document store)

비관계형 데이터베이스를 선택하는 것이 바람직한 경우

1. 아주 낮은 응답 지연시간이 요구됨
2. 다루는 데이터가 비정형이라 관계형 데이터가 아님
3. 데이터를 직력화하거나 역질렬화 할 수 있기만 하면 됨
4. 아주 많은 양의 데이터를 저장할 필요가 있음

▼ 수직적 규모 확장 vs 수평적 규모 확장

▼ 수직적 규모 확장

Scale Up / Vertical Scaling

서버에 고사양 자원을 추가하는 행위

한 대의 서버에 CPU나 메모리를 무한대로 증설할 방법이 없기 때문에 한계가 있음

장애에 대한 자동복구 방안이나 다중화 방안을 제시하지 않음

서버에 장애가 발생하면 웹사이트/앱이 완전히 중단됨

▼ 수평적 규모 확장

Scale Out

더 많은 서버를 추가하여 성능을 개선하는 행위

▼ 로드밸런서

많은 접속자가 발생하여 응답 속도가 느려지거나 서버 접속이 불가능한 상황을 최소화하기 위한 방안

부하 분산 집합에 속한 웹 서버들에게 트래픽 부하를 고르게 분산하는 역할

- 사용자는 로드 밸런서의 공개 IP 주소로 접속
  - 웹 서버가 클라이언트의 접속을 직접 처리하지 않음
- 더 나은 보안을 위해 서버 간 통신에는 사설 IP 주소가 이용됨
  - 같은 네트워크에 속한 서버 사이의 통신에만 쓰일 수 있음
  - 인터넷을 통해서는 접속 불가
  - 웹 서버와 통신하기 위해 사설 IP 이용

부하 분산 집합에 또 하나의 웹 서버를 추가하면 장애를 복구하지 못하는 문제 해소 → 가용성 향상

1. 서버 1이 다운될 경우
    - 모든 트래픽은 서버 2로 전송(웹 사이트 전체가 다운되는 일 방지)
2. 웹 사이트로 유입되는 트래픽이 가파르게 증가할 경우
    - 두 대의 서버로 트래픽을 감당할 수 없는 시점이 오면 서버를 추가하기만 하면 됨
    - 로드밸런스가 자동적으로 트래픽 분산

▼ 데이터베이스 다중화

많은 데이터베이스 관리 시스템이 다중화를 지원

서버 사이에 주(master)-부(slave) 관계를 설정하고 데이터 원본은 주, 데이터 사본은 부에 저장

주 데이터베이스

- 쓰기 연산 가능
- 데이터베이스를 변경하는 명령어들(insert, delete, update, ..)은 주 데이터베이스로만 전달되어야 함

부 데이터베이스

- 주 데이터베이스로부터 사본을 전달받음
- 쓰기 연산 불가능
- 읽기 연산만 지원

대부분의 애플리케이션은 읽기 연산의 비중이 쓰기 연산보다 훨씬 높음

→ **부 데이터베이스의 수가 주 데이터베이스의 수보다 많음**

데이터베이스 다중화의 장점

1. 더 나은 성능
    - 모든 데이터 변경은 주 데이터베이스 서버로만 전달

- 읽기 연산은 부 데이터베이스 서버들로 분산
- 병렬로 처리될 수 있는 query 수가 늘어남 → 성능이 좋아짐

2. 안정성
    - 데이터베이스 서버의 일부가 파괴되어도 데이터 보존
    - 지역적으로 떨어진 여러 장소에 다중화도 가능
3. 가용성
    - 데이터를 여러 지역에 복제 가능
    - 하나의 데이터베이스 서버에 장애가 발생하더라도 다른 서버에 있는 데이터를 가져와 계속해서 서비스 가능

부 데이터베이스 서버가 다운될 경우

1. 부 서버가 한대 뿐인데 다운된 경우, 읽기 연산이 모두 주 데이터베이스로 전달
2. 부 서버가 여러 대인데 다운된 경우, 나머지 부 데이터베이스 서버로 전달
3. 새로운 부 데이터베이스 서버가 장애 서버를 대체

주 데이터베이스 서버가 다운될 경우

1. 부 서버가 한대 뿐인데 다운된 경우, 부 데이터베이스 서버가 주 서버로 변경
2. 모든 데이터베이스 연산이 일시적으로 새로운 주 서버에서 실행
3. 새로운 부 서버 추가
4. 부 서버에 보관된 데이터가 최신 상태가 아닐 경우, 없는 데이터를 복구 스크립트를 통해 추가
    - 다중 마스터나 원형 다중화 방식을 도입하면 대처하는 데 도움이 됨(복잡함)

로드밸런서와 데이터베이스 다중화를 모두 고려한 설계의 동작 방식

1. 사용자는 DNS로부터 로드밸런서의 공개 IP 주소 받음
2. 해당 IP 주소를 사용해 로드밸런서 접속
3. HTTP 요청은 서버 1이나 2로 전달

4. 웹 서버는 사용자의 데이터를 부 데이터베이스 서버에서 read
5. 데이터 변경 연산(데이터 추가, 삭제, 갱신, ...)은 주 데이터베이스로 전달

▼ 캐시

값비싼 연산 또는 자주 참조되는 데이터를 메모리 안에 두고, 이후 요청이 빨리 처리될 수 있도록 하는 저장소

웹 페이지를 새로고침 할 때마다 표시할 데이터를 가져오기 위해 한 번 이상의 데이터베이스 호출이 발생

애플리케이션 성능: 데이터베이스를 얼마나 자주 호출하느냐에 크게 좌우

캐시는 이러한 성능 문제를 완화할 수 있음

캐시 계층

데이터가 잠시 보관되는 곳

데이터베이스보다 훨씬 빠름

별도의 캐시 계층을 통해 데이터베이스의 부화를 줄이고, 규모를 독립적으로 확장 가능

주도현 캐시 전략(read-through caching strategy)

요청을 받은 웹 서버는 캐시에 응답이 저장되어 있는지 확인

저장되어 있다면 해당 데이터를 클라이언트에 반환

없다면 데이터베이스 query를 통ㅇ해 데이터를 찾아 캐시에 저장한 뒤 클라이언트에 반환

캐시할 데이터 종류, 크리, 액세스 패턴에 맞는 캐시 전략을 선택

캐시 사용 시 유의할 점

1. 어떤 상황에 바람직한가?
2. 어떤 데이터를 캐시에 두어야 하는가?
    - 데이터를 휘발성 메모리에 두는 것이므로, 영속적으로 보관할 데이터를 캐시에 두는 것은 바람직하지 않음
    - 중요 데이터를 여전히 지속적 저장소에 두어야 함

3. 캐시에 보관된 데이터는 어떻게 만료되는가?
    - 만료 정책을 마련해 두어야 함
    - 만료된 데이터는 캐시에서 삭제
    - 만료 기한이 너무 짧으면 데이터베이스를 자주 읽게 됨
    - 만료 기한이 너무 길면 원본 데이터와 차이가 날 가능성이 높아짐
4. 일관성은 어떻게 유지되는가?
    - 캐시와 저장소 사이에 일관성을 유지하는 것은 어려운 문제지만 반드시 해결해야 함
5. 장애에는 어떻게 대처할 것인가?
    - 캐시 서버를 한 대만 두는 경우 해당 서버는 단일 장애 지점이 될 가능성이 있음
        - 어떤 특정 지점에서의 장애가 전체 시스템의 동장을 중단시켜버릴 수 있는 경우
    - 여러 지역에 걸쳐 캐시 서버를 분산시켜야 함
6. 캐시 메모리는 얼마나 크게 잡을 것인가?
    - 캐시 메모리가 너무 작으면 엑세스 패턴에 따라서는 데이터가 너무 자주 캐시에서 밀려나 성능이 떨어질 수도 있음
    - 캐시 메모리를 과할당하는 것도 이를 해결하기 위한 하나의 방법
7. 데이터 방출 정책은 무엇인가?
    - 캐시가 꽉 차면 추가로 캐시에 데이터를 저장해야 할 경우 기존 데이터를 내보내야 함
    - 보통 LRU(마지막으로 사용된 시점이 가장 오래된 데이터를 방출)을 많이 사용
    - LFU(사용된 빈도가 가장 낮은 데이터를 방출)이나 FIFO(가장 먼저 자장된 데이터를 방출)를 사용하기도 함
    - 상황에 맞게 선택하는 것이 좋음

▼ 콘텐츠 전송 네트워크(CDN)

정적 콘텐츠를 정송하는데 쓰이는, 지리적으로 분산된 서버의 네트워크

이미지, 비디오, CSS, JavaScript 파일 등을 캐시

동적 콘텐츠 캐싱

상대적으로 새로운 개념이라 책에서 다룰 수 있는 범위 밖

요청 경로, 질의 문자열, 쿠키, 요청 헤더 등의 정보에 기반하여 HTML 페이지를 캐싱

CDN의 동작 방법

1. 사용자가 이미지 URL을 통해 특정 이미지에 접근
    - URL의 도메인은 CDN 서비스 사업자가 제공
2. CDN 서버의 캐시에 해당하는 이미지가 없는 경우, 원본 서버에 요청하여 파일 호출
    - 원본 서버는 웹 서버일 수도, Amazon S3 같은 온라인 저장소일 수도
3. 원본 서버가 파일을 CDN 서버에 반환
    - 응답의 HTTP 헤더에는 해당 파일을 얼마나 오래 캐시될 수 있는지를 설명하는 TTT(Time-To-Live) 값이 들어 있음
4. CDN 서버는 파일을 캐시하고 사용자에게 반환
5. 이미지는 TTL에 명시된 시간이 끝날 때까지 캐시
6. 다른 사용자가 같은 이미지에 대한 요청을 CDN 서버에 전송
7. 만료되지 않은 이미지에 대한 요청은 캐시를 통해 처리

CDN 사용시 고려해야 할 상황

1. 비용
    - CDN은 제3 사업자에 의해 운영
    - CDN으로 들어가고 나가는 데이터 전송 양에 따라 요금 지불
    - 자주 사용되지 않는 콘텐츠를 캐싱하는 것은 돈낭비
2. 적절한 만료 시한 설정
    - 시의성이 중요한 콘텐츠의 경우 만료 시점이 중요한
    - 너무 길지면 콘텐츠의 신선도가 떨어짐
    - 너무 짧으면 원본 서버에 빈번히 접속하게 되어 성능 저하
3. CDN 장애에 대한 대처 방안
    - CDN 자체가 죽었을 경우 웹 사이트/애플리케이션이 어떻게 동작해야 하는지 고려

- 문제를 감지하여 원본 서버로부터 직접 콘텐츠를 가져오도록 클라이언트를 구성하는 것도 하나의 방법

4. 콘텐츠 무효화 방법
    - 아직 만료되지 않은 콘텐츠라고 해도 CDN에서 제거 가능
        - CDN 서비스 사업자가 제공하는 API를 이용하여 콘텐츠 무효화
        - 콘텐츠의 다른 버전을 서비스하도록 오브젝트 버저닝(URL 마지막에 버전 정보 인자로 전송) 이용

CDN과 캐시가 추가된 설계

1. 정적 콘텐츠는 더 이상 웹서버를 통해 서비스하지 않음
2. CDN을 통해 제공되기 때문에 더 나은 성능 보장
3. 캐시가 데이터베이스의 부하를 줄여줌

▼ 무상태(stateless) 웹 계층

웹 계층을 수평적으로 확장하기 위해서는 상태 정보를 웹 계층에서 제거해야 함

상태관계형 데이터베이스나 NoSQL 같은 지속성 저장소에 보관하고, 필요할 때 가져오도록 하는 것이 최선

이렇게 구성된 웹 계층 = **무상태 웹 계층**

▼ 상태 정보 의존적인 아키텍처

클라이언트 정보(상태)를 유지하여 요청들 사이에 공유

사용자의 세션 정보나 프로파일 이미지가 특정 서버에 저장

사용자를 인증하기 위해 HTTP 요청은 반드시 정보가 저장된 특정 서버로 전송해야 함

다른 서버로 전송할 경우 인증 실패

같은 사용자의 요청은 항상 같은 서버로 전송되어야 함

로드밸런서가 이를 위해 고정 세션 지원하지만 로드밸런서 입장에서는 부담되는 처리

서버의 장애를 처리하기도 복잡

▼ 무상태 아키텍처

사용자로부터의 HTTP 요청은 아무 웹 서버로 전달 가능

상태 정보가 필요한 경우 공유 저장소(관계형 데이터베이스, 캐시 시스템, NoSQL,) 로부터 데이터 추출

상태 정보와 웹 서버가 물리적으로 분리

단순하고, 안정적이며, 규모 확장이 쉬움

상태 정보가 웹 서버들로부터 제거됨

트래픽 양에 따라 웹 서버를 넣거나 빼기만 하면 자동으로 규모 확장 가능

▼ 데이터 센터

장애가 없는 상황

- 사용자에게 가장 가까운 데이터 센터로 안내됨 = **지리적 라우팅**

geoDNS

사용자의 위치에 따라 도메인 이름을 어떤 IP 주소로 변환할지 결정할 수 있도록 하는 DNS 서비스

데이터 센터 중 하나에 심각한 장애가 발생한 경우

- 모든 트래픽이 장애가 없는 데이터 센터로 전송

다중 데이터센터 아키텍처를 만들 때 해결해야 할 기술적 난제

1. 트래픽 우회
    - 올바른 데이터 센터로 트래픽을 보내는 효과적인 방법을 찾아야 함
2. 데이터 동기화
    - 데이터 센터마다 별도의 데이터베이스를 사용하고 있는 경우, 장애가 자동우로 복구되어 트래픽이 다른 데이터베이스로 우회된다고 해도, 해당 데이터 센터에는 찾는 데이터가 없을 수도 없음
    - 데이터를 여러 데이터 센터에 걸쳐 다중화하는 것이 하나의 방법
3. 테스트와 배포
    - 웹 사이트/애플리케이션을 여러 위치에서 테스트해 보는 것이 중요

- 자동화된 배포 도구는 모든 데이터 센터에 동일한 서비스가 설치되도록 하는 데 중요한 역할

▼ 메세지 큐

메지의 무손실(메세지 큐에 보관된 메세지는 소비자가 꺼낼 때까지 안전히 보관됨)을 보장

비동기 통신을 지원하는 컴포넌트

메세지의 버퍼 역할

메세지 큐 동작 방법

1. 생산자(publisher)라고 불리는 입력 서비스가 메세지를 만들어 메세지 큐에 발행(publish)
2. 연결되어 있는 구독자(subscriber)라고 불리는 서버가 메세지를 받아 그에 맞는 동작 수행

서비스와 서버 간 **결합이 느슨**해짐

규모 확장성이 보장되어야 하는 안정적 애플리케이션을 구성하기 좋음

이미지의 크로핑, 샤프닝, 블러링 등을 지원하는 사진 보정 애플리케이션

시간이 오래 걸릴 수 있는 프로세스 → 비동기로 처리하면 좋음

웹 서버는 사진 보정 작업을 메세지 큐에 넣음

보정 작업 프로세스들이 이 작업을 메세지 큐에서 꺼내 비동기로 처리

▼ 로그, 메트릭 그리고 자동화

사업 규모가 커지면 로그, 메트릭, 자동화와 같은 작업들에 필수적으로 투자해야 함

▼ 로그

에러 로그를 모니터링하는 것은 중요함

시스템의 오류와 문제들을 쉽게 찾아낼 수 있음

로그를 단일 서비스로 모아주는 도구를 활용하면 더 편리하게 검색하고 조회할 수 있음

▼ 메트릭

잘 수집하면 사업 현황에 관한 유용한 정보를 얻을 수 있음

시스템의 현재 상태를 손쉽게 파악할 수 있음

1. 호스트 단위 메트릭
    - CPU, 메모리, 디스크/IO
2. 종합 메트릭
    - 데이터베이스, 캐시 계층의 성능
3. 핵심 비지니스 메트릭
    - 일별 능동 사용자, 수익, 재방문

▼ 자동화

시스템의 생산성을 높일 수 있음

빌드, 테스트, 배포 절차를 자동화하면 개발 생산성을 크게 향상시킬 수 있음

메세지 큐, 로그, 메트릭, 자동화 등을 반영하여 수정한 설계안

1. 메세지 큐는 각 컴포넌트가 보다 느슨한 결합이 될 수 있도록 하고, 결함에 대한 내성을 높임
2. 로그, 모니터링, 메트릭, 자동화 등을 지원하기 위한 장치를 추가

▼ 데이터베이스의 규모 확장

저장할 데이터가 많아지면 데이터베이스에 대한 부하도 증가

데이터베이스 규모 확장이 필수

▼ 수직적 확장

Scale up

기존 서버에 더 많은 혹은 고성능의 자원(CPU, RAM, 디스크, ..)을 증설

심각한 약점

1. 데이터베이스 서버 하드웨어에는 한계가 있음
    - 무한으로 증성 불가
2. SPOF(Single Point of Failure)로 인한 위험성이 큼

3. 비용

▼ 수평적 확장

더 많은 서버를 추가함으로써 성능을 향상

Sharding

대규모 데이터베이스를 샤드(Shard)라고 부르는 작은 단위로 분할하는 기술

모든 샤드는 같은 스키마를 사용

샤드에 보관되는 데이터 사이에는 중복이 없음

어느 샤드에 어떤 데이터가 쌓이는지는 각각 다름

- 특정 값을 해시 함수로 사용하여 데이터가 보관될 샤드를 정하기도 함

샤딩 키(= 파티션 키)를 어떻게 정하느냐가 가장 중요

- 하나 이상의 컬럼으로 구성

Sharding 도입 시 생각해봐야 할 문제

1. 데이터의 재 샤딩
   - 데이터가 너무 많아져서 하나의 샤드로는 더 이상 감당하기 어려울 때
   - 샤드 간 데이터 분포가 균등하지 못하여 어떤 샤드에 할당된 공간 소모가 다른 샤드에 비해 빨리 진행될 때
   - 샤드 키를 계산하는 함수를 변경하고 데이터를 재배치해야 함
2. 유명인사(핫스팟 키) 문제
   - 특정 샤드에 질의가 집중되어 서버에 과부하가 걸리는 문제
   - 집중되는 데이터에 각각 샤드 하나씩 할당 혹은 더 잘게 쪼개기
3. 조인과 비정규화
   - 하나의 데이터베이스를 여러 샤드 서버로 쪼개면 여러 샤드에 걸친 데이터를 조인하기 힘들어짐
   - 데이터베이스를 비정규화하여 하나의 테이블에서 질의가 수행될 수 있도록 하기

▼ 백만 사용자, 그리고 그 이상

시스템의 규모를 확장하는 것은 지속적이고 반복적인 과정

수백만 사용자 이상을 지원하려면 새로운 전략을 도입해야 하고 더 지속적으로 시스템을 가다듬어야 함

시스템 규모 확장을 위해 이번 장에서 살펴본 기법

1. 웹 계층은 무상태 계층으로
2. 모든 계층에 다중화 도입
3. 가능한 한 많은 데이터를 캐시할 것
4. 여러 데이터 센터를 지원할 것
5. 정적 컨텐츠는 CDN을 통해 서비스할 것
6. 데이터 계층은 샤딩을 통해 그 규모를 확장할 것
7. 각 계층은 독립적 서비스로 분할할 것
8. 시스템을 지속적으로 모니터링하고, 자동화 도구들을 활용할 것

▼ 토론

데이터 동기화를 위한 다중 데이터센터 아키텍처 설계: 데이터베이스 복제 vs. 데이터 스트리밍

1. 데이터베이스 복제:
    - 데이터베이스 복제는 다중 데이터센터 간에 데이터를 복사하는 방법 중 하나입니다.
    - 주 데이터베이스에서 변경된 데이터를 보조 복제본 데이터베이스로 복사하여 동기화합니다.
    - 복제된 데이터베이스는 읽기 작업에 사용되거나 장애 복구 시에 대체로 사용됩니다.
    - 복제된 데이터베이스 간에는 일정한 시간이 소요되므로, 데이터 동기화에 약간의 지연이 발생할 수 있습니다.
2. 데이터 스트리밍:
    - 데이터 스트리밍은 데이터를 실시간으로 전송하여 다중 데이터센터 간에 동기화하는 방법입니다.
    - 변경된 데이터를 스트림으로 만들고, 스트림을 다른 데이터센터로 전송하여 동기화합니다.

- 스트리밍 기술은 일반적으로 데이터베이스의 로그를 기반으로 작동하며, 변경 사항을 실시간으로 반영합니다.
- 데이터 스트리밍은 거의 실시간으로 데이터를 동기화할 수 있으므로, 데이터베이스 복제보다 빠른 동기화가 가능합니다.

데이터베이스 복제와 데이터 스트리밍은 모두 다중 데이터센터 환경에서 데이터 동기화를 달성하기 위해 사용되는 방법입니다. 선택할 방법은 요구 사항, 확장성, 지연 시간 등을 고려하여 결정해야 합니다.

1장 훑으면서 대규모 설계를 할때 가장 중요하다고 생각하는 것은?

- 전체적으로 다중어쩌구를 사용하는 것이 좋은 것 같고, 그 중에서 가장 중요한 것을 디비랑 로그가 중요한 것 같다. 직접 느끼는 것이기도 함. 그리고 독립적으로 서비스를 분할하는 것도 중요하다고 생각.