1주차

사용자 수에 따른 규모 확장성

단일 서버

- 모든 컴포넌트가 단 한대의 서버에서 실행되는 간단한 시스템
- 웹, 앱, 데이터베이스, 캐시 등이 전부 서버 한대에서 실행

1. 사용자가 api.mysite.com에 접근하기 위해서 제일 먼저 DNS에 질의한다. DNS의 반복적질의와 재귀적 질의를 거친 후, IP주소를 받는다.
2. IP주소(15.125.23.314)로 요청을 보낸다. (이때 사용하는 것은 HTTP 프로토콜로 요청함)
3. 그 웹 서버에서 HTML 페이지 또는 JSON 형태로 응답한다. (이때 사용하는 것은 HTTP 프로토콜로 응답함)

2번에서 요청은 어디서부터 보내지는 가?

- 웹 애플리케이션 : 비즈니스 로직, 데이터 저장등을 처리하기 위해서는 서버 구현용 언어를 사용하고, 프레젠테이션 용으로는 클라이언트 구현용 언어를 사용한다.
- 모바일 앱 : HTTP 프로토콜을 사용함, 응답 데이터 포맷으로는 보통 JSON을 사용

데이터베이스

- 사용자가 늘면 서버를 여러개 두어야한다.
- 그림 1-3처럼 웹/모바일 트래픽 처리용도로 하나를 두고 데이터베이스 서버를 하나를 두어서 서로 분리시키면 각각을 독립적으로 확장해 나갈 수 있게 된다.

관계형 데이터베이스 vs 비-관계형 데이터베이스

관계형 데이터베이스

- 관계형 데이터베이스(RDBMS)는 MySQL, 오라클, PostgreSQL 등이 대표적이다.
- 관계형 데이터베이스는 자료를 테이블과 열, 칼럼으로 표현한다.
- SQL을 사용하면 여러 테이블에 있는 데이터를 그 관계에 따라 조인하여 합칠 수 있다.

- 장점

1. 데이터의 성능이 일반적으로 좋아 정렬, 탐색, 분류가 빠르다
2. 신뢰성이 높아 데이터의 무결성을 보장한다.
3. 정규화에 따른 갱신 비용을 최소화한다.

- 단점

1. 기존에 작성된 스키마를 수정하기 어렵다.
2. 데이터베이스의 부하를 분석하기 어렵다
3. 빅데이터를 처리하는데 비효율적이다.

비-관계형 데이터베이스

- 비 관계형 데이터베이스(NoSQL)는 CouchDB, Neo4j, Cassandra, HBase, Amazon DynamoDB 등이 대표적
- 키-값 저장소 : 거대한 Map으로 key-value 형식으로 지원
- 그래프저장소
- 칼럼저장소
- 문서 저장소

→ 이렇게 네가지로 분류할 수 있다. 이 비관계형 데이터베이스는 일반적으로 조인 연산을 지원하지 않는다.

- 장점

1. 대용량 데이터 처리를 하는 데 효율적이다.
2. 읽기 작업보다 쓰기 작업이 더 빠르고 관계형 데이터베이스에 비해 쓰기와 읽기 성능이 빠르다.
3. 복잡한 데이터 구조를 표현할 수 있다.

- 단점 : 쿼리 처리시 데이터를 파싱 후 연산을 해야해서 큰 크기의 문서를 다룰 때는 성능이 저하됨.

→ 관계형 데이터베이스는 전통적인 것으로서 많이 쓰이지만 이것의 단점을 보완하기 위해 나온 데이터베이스가 비-관계형데이터베이스이기 때문에 앞으로 쓰임이 많아 질 것이다.

수직적 규모 확장 vs 수평적 규모 확장

- 스케일 업(scale up) : 서버의 용량을 늘리는 방식 → 수직적 규모 확장
- 스케일 아웃(scale out) : 서버의 수를 늘리는 방식 → 수평적 규모 확장

> 서버로 유입되는 트래픽의 양이 적을 때는 수직적 확장이 좋은 선택이다.

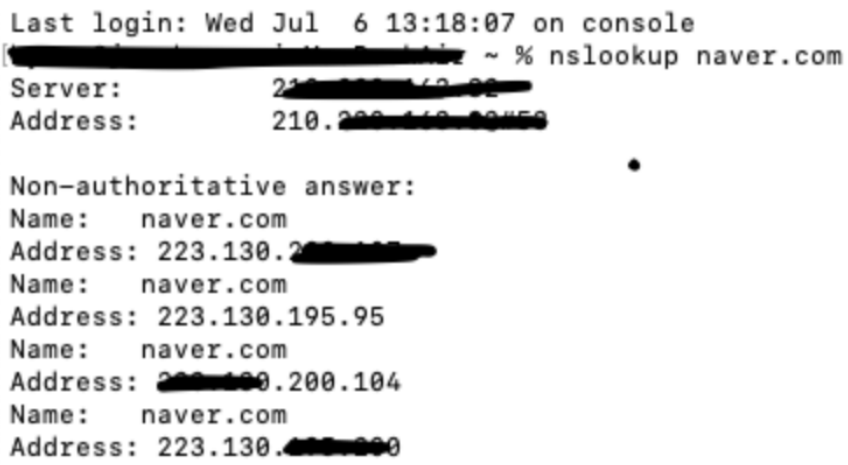
→ 수직적 확장의 단점

1. 무한대로 늘리는 것은 불가능
2. 장애에 대한 자동복구방안이나 다중화 방안을 제시하지 않는다. 서버에 장애가 발생하면 웹사이트/앱이 완전히 중단 된다.

- 즉, 하나의 서버에 많은 사용자가 접속한다면 웹 서버가 한계 상황에 도달하게 되고 응답속도가 느려지거나 서버가 터질 수 있음 → 이 문제를 해결하는 데는 부하 분산기 또는 로드밸런서를 도입하는 것이 최선이다.

로드밸런서

- 로드밸런서는 부하 분산 집합에 속한 웹 서버들에게 트래픽 부하를 고르게 분산하는 역할을 한다.
- 이런식으로 네이버를 검색해보면 여러개의 서버를 가지고 있는 것을 볼 수 있다.



서버 1이 다운되면 서버2로 전송되고 이렇게 웹사이트 전체가 다운되는 일을 방지한다. 저렇게 여러개의 서버들을 두어서 순차적으로 돌아가면서 private서버들을 보낸다.

이렇게 로드밸런스가 자동적으로 트래픽을 분산한다.

데이터베이스 다중화

- 많은 데이터베이스 관리 시스템이 다중화를 지원한다.
- master-slave 구조로 서버들 사이에 관계를 지정해서 원본은 주서버에, 사본은 부서버에 저장하는 방식이다.
- 쓰기 연산은 마스터에서만 지원한다. 부 데이터베이스는 주 데이터베이스로부터 그 사본을 전달 받으며, 읽기 연산만을 지원한다. 데이터베이스를 변경하는 명령어들 가령 insert, delete, update등은 주 데이터베이스로만 전달해야한다. 보통 애플리케이션은 읽기 연산의 비중이 쓰기 연산보다 훨씬 높기 때문에 부 데이터베이스의 수가 주 데이터베이스 수보다 많다.

→ 데이터베이스의 다중화 장점

1. 더나은 성능
2. 안전성
3. 가용성

서버처럼 데이터베이스도 로드밸런서가 시스템 가용성을 높인다.

캐시

- 값비싼 연산 결과 또는 자주 참조되는 데이터를 메모리안에 두고, 뒤 이은 요청이 보다 빨리 처리 될 수 있도록하는 저장소
- 캐시 계층

1. 데이터가 잠시 보관되는 곳, 데이터베이스보다 훨씬 빠르다
2. 캐시부터 들린다.
3. 없다면 데이터베이스를 찾는다. 있다면 캐시에서 요청에대한 응답을 반환한다. → 읽기 주도형 캐시전략!!

- 캐시 사용시 유의점

1. 데이터 갱신은 자주 일어나지않지만 참조는 빈번하게 일어날 때 캐시를 사용한다
2. 영속적으로 보관해야하는 것은 지속적인 저장소에 둬야한다. 캐시는 휘발성이기 때문에 사라진다.
3. 만료에 대한 정책을 마련해 둬야한다.
4. 일관성과 무결성을 지켜야한다.
5. 장애에 대처해야한다.
6. 캐시메모리 크기에 대해서도 생각해야한다
7. 데이터방출 정책에 대해서도 생각해야한다.

CDN

- 정적 콘텐츠를 전송하는 데 쓰이는, 지리적으로 분산된 서버의 네트워크이다. 이미지, 비디오, CSS, JavaScript파일을 캐시할 수 있다.

CDN 동작

1. 사용자가 URL을 이용해 image.png에 접근한다. URL의 도메인은 CDN서비스 사업자가 제공한 것이다.
2. CDN 서버의 캐시에 해당 이미지가 없는 경우, 서버는 원본 서버에 요청하여 파일을 가져온다. 원본 서버는 웹 서버일 수도 있고 아마존 S3와 같은 온라인 저장소일 수도있다.
3. 원본 서버가 파일을 CDN 서버에 반환한다. 응답의 HTTP 헤더에는 해당 파일이 얼마나 오래 캐시될 수 있는지를 설명하는 TTL 값이 들어 있다.
4. CDN서버는 파일을 캐시하고 사용자 A에게 반환한다. 이미지는 TTL 에 명시된 시간이 끝날때까지 캐시된다.

CDN 사용시 고려해야할 사항

- 비용, 적절한 만료 시한 설정, CDN 장애에 대한 대처 방안, 콘텐츠 무효화

무상태 웹계층

로드밸런싱으로 서버를 여러개를 띄우면 A사용자가 서버1에 로그인 요청을하고 서버세션에 사용자A를 저장을 했을 것이다. 하지만 사용자A가 2번째 서버에 요청을 한다면 또 서버는 사용자B로 인식(새로운사용자로 인식)하여 또 세션이라는 값에 저장할 것이다. 이런식으로 서버들은 서로 공유를 안하는 무상태이기때문에 요청들을 항상 새롭게 받아들인다

→ 해결책

1. 고정 세션 : 서버1, 서버2로 사용자 A를 보내지 않고 유저 한명은 한 서버가 전담하도록 하는 것이다. → 로드밸런서에 부담을 준다
2. 공유 저장소 : 서버1, 서버2의 데이터베이스를 공유하는 것이다.

데이터센터

- 가장가까운 데이터 센터로 보내는 것 : 지리적 라우팅
- DNS가 지리적라우팅을 해줌 → 트래픽 우회 : 만약 가까운것이 쓸 수 없으면, 그다음 가까운것으로!!
- 데이터 동기화
- 테스틑와 배포

메시지큐

- 메시지 큐는 메시지의 무손실을 보장하는 비동기 통신 컴포넌트이다.
- 메시지의 버퍼역할을 하며 비동기적으로 전송된다.

로그, 메트릭 그리고 자동화

- 로그 : 에러로그는 단일 서비스로 모아주는 도구를 사용하면 좋다.
- 메트릭 : 메트릭을 잘 수집하면 사업현황에 관한 유용한 정보를 얻을 수 있고 시스템의 현재 상태를 손쉽게 파악할 수도 있다.

1. 호스트 단위 메트릭 : CPU, 메모리, 디스크I/O
2. 종합 메트릭 : DB 계층의 성능, 캐시 계층의 성능

3. 핵심 비즈니스 메트릭 : 일일 능동 사용자, 수익, 재방문

- 자동화 : 지속적 통합을 도와주는 도구들을 활용해서 코드의 검증을 하고 빌드, 테스트, 배포등의 절차를 자동화하여 개발 생산성을 높인다.

데이터베이스의 규모 호가장

- 저장할 데이터가 많아지면 데이터베이스에 대한 부하도 증가 → 수직적 규모 확장법 , 수평적 규모 확장법
- 수직적 확장 (스케일 업) : 서버 용량을 늘린다. 서버 하드웨어에는 한계가 있고 비용이 많이 든다.
- 수평적 확장(샤딩) : 서버 수를 늘린다. 대규모 데이터베이스 샤드로 나누고 데이터를 여러대의 서버에 분산해서 저장한다. 어느 서버에 넣을 지 판단은 사용자 ID에 따라 정한다.→ 문제점 3가지

1. 재샤딩 : 한 샤드가 다른 샤드들에 비해 빨리 소진되어 감당하기 어려울 때 샤딩키를 다시 분배해서 저장해야되는 문제가 있다
2. 셀럽문제
3. 조인과 비정규화문제