[ system design basics ]

I. system design basics (**시스템 설계 기본**)
    1. **고려사항**
        a. **프로젝트에서 사용될 컴포넌트(시스템 구성요소) 조사**
        b. **컴포넌트 작동방식 - 컴포넌트끼리 어떻게 작동되는지**
        c. **컴포넌트 활용방안 - 어떻게 잘 활용 할 수 있는지**
    2. **필요하기전까진 확장하지 않음**
    3. **예측은 귀중한 시간과 리소스를 절약**

II. key characteristics of distributed system (**분산 시스템 주요 특징**)
    1. scalability (**확장성**, 1.jpg)
        a. (**수평확장**)
            i. **서버 수를 늘림**
            ii. cassandra, mongodb
        b. (**수직확장**)
            i. **서버 성능을 늘림** - cpu, ram, hdd upgrade
            ii. mysql
            iii. downtime (**서비스를 종료해야 하는 시간**) **이 발생**
    2. realiability (**안정성**) : **일부 서버가 다운 되어도 서비스는 지속**
    3. availability (**유효성**) : **정상적인 작동상태를 유지하는 시간을 백분율로 측정**
    4. effeiciency (**효율성**) : **투입된 자원 대비 성능**
    5. serviceability or manageability (**관리성**) : **운영이 얼마나 쉬운가?**

III. load balancing (로드밸런싱)

1. 목적 : responsiveness (응답성) 과 availiablity (유효성) 향상 (2.jpg)
2. 기능
    a. 서버의 상태를 추적
    b. 트래픽 분산
        i. 여러 장치에서 작업을 분담
        ii. 문제가 있는 서버에게는 트래픽 전송 중지
3. 위치 (3.jpg)
    a. 사용자와 웹서버 간
    b. 웹서버와 애플리케이션 서버, 내부 플랫폼 계층 (캐시서버) 간
    c. 내부 플랫폼 계층과 db 간
4. 장점
    a. 처리량을 높이고 다운타임을 줄임
    b. 사용자 대기시간 줄임
    c. smart load balancer : bottleneck (트레픽 병목) 이 발생하기 전이 미리 예측 하여 대응
5. algorithm
    a. least connection method (최소 연결 방법)
        i. 활성 연결이 가장 적은 서버로 트래픽 전송
        ii. 분산이 고르지 않은 다수의 영구 클라이언트 연결이 있는 경우 유용
    b. least response time method (최소 응답시간 방법)
        i. 활성 연결이 적고 평균 응답 시간이 가장 적은 서버로 트래픽 전송
    c. least bandwidth method (최소 대역폭 방법)
        i. bps 단위로 측정한 값이 가장 적은 서버로 트래픽 전송
    d. round robin method (라운드 로빈 방법)
        i. 목록 작성 후 작성된 순서로 순환하면서 트래픽 전송
    e. weighted round robin method (가중 라운드 로빈 방법)
        i. 라운드 로빈 방식에 가중치를 적용하는 방식
    f. ip hash
        i. 클라이언트 ip 주소가 hash 계산 되어 선택된 서버에 트래픽 전송

6. redundant load balancers (**중복 로드 밸런서**, 4.jpg)
    a. **다수개의** lb**를** cluster
    b. **각** lb**는 다른** lb**의 상태를 모니터링**
    c. main lb**가 실패하는 경우** second lb**가 대신 수행**

IV. cache

1. lb vs cache
    a. lb : **수평확장**
    b. cache : **수직확장**, optimization (**최적화**)
2. **모든 계층에서 사용가능** : hw, os, brawser, application, etc
3. frontend **근처에서 주로 사용** : **다운 스트림에 부담을 주지 않으면서 데이터를 빠르게 반환가능**
4. short term memory : ram **에 상주**
5. cdn (conents distribution network)
    a. **요청파일이** cache**에 있는 경우** : cache**에 있는 것을 사용**
    b. **요청파일이** cache**에 없는 경우** : storage**에 있는 파일을** cache**에 올린 후 사용**
6. cache invalidation (**캐시 무효화**, cache**와** storage**를** sync)
    a. write through cache
        i. cache**와** storage**를 함께 기록**
        ii. **장점** : **항상** sync
        iii. **단점** : write 2**배 느림**
    b. write around cache
        i. storage **만** write
        ii. cache**에 요청데이터가 없을 경우** cache miss **리턴 후** storage**에 있는 데이터를 서비스**
        iii. **장점** : write **빠름**
        iv. **단점** : cache**에 요청된 데이타가 없을 경우 속도 느림**
    c. write back cache
        i. cache **만** write
        ii. scheduling**을 통해** storage **와** sync
        iii. **장점** : **속도가 빠름**
        iv. **단점** : **장애 발생시 데이터가 손실될 위험이 있음**
7. cache eviction policy (**캐시 제거 정책**)
    a. fifo
    b. lifo
    c. lru

d. mru
e. lfu
f. rr (random replacement)

V. Data Partitioning
    1. db를 나누는 기술
    2. partitioning methods (파티셔닝 방법)
        a. horizontal partitioning (수평분할)
            i. range based partitioning (ex) 우편번호
            ii. 장점 : simple
            iii. 단점 : 분할표를 잘 작성하지 않으면 분포가 불균형해짐
        b. vertical partitioning (수직분할)
            i. categoriy 별 분할
            ii. 고객정보, 사진, 상품정보를 각각의 db에저장
        c. directory based partitioning (디렉토리기반분할)
            i. pc directory 식 분할 방식
            ii. data 위치 탐색 시 directory 서버 사용 (tuple키와 db간에 매핑정보를 보유)
            iii. loosely coupled approach (느슨한 연결방식) : application에 영향을 주지 않고 db pool을 서버에 추가하거나 파티셔닝 구성표를 변경
    3. partitioning criteria (파티셔닝 기준)
        a. key or hash based partitioning (키 및 해시기반 파티셔닝)
            i. hash function : id % number of server
            ii. 문제 : 확장하기 어려움
            iii. 해결 : consistent hashing (일관된 해싱), repartitioning
        b. list partitioning (파티션 분할 목록)
            i. 의미로 나눔
                (ex) 아시아 - 한국, 일본, 중국, 태국 ...
        c. round-robin partitioning (라운드 로빈 파티셔닝)
            i. 서버 순서 대로 하나씩 돌아가면서 배정 (ex) 아이들 과자 나눠주기
        d. composite partitioning (컴포지드 파티셔닝)
            i. 위 3개 분할 기법들을 적절히 활용
                (ex) list partitioning 후 round-robin partitioning 적용
    4. 문제점
        a. 파티션간 쿼리를 수행 할 수 없음

    i. denomalization (비정규화) -> join이 필요한 query를 단일테이블에서 수행하도록 변경
    ii. 데이버 불일치 같은 비정규화로 발생하는 모든 위험을 처리해야 함
b. fk(외래키) 사용 못함 : 참조무결성 (referential integrity) 적용 못함
c. 데이터 분배가 불균일, 특정 파티션에 많은 부하
    i. 더 많은 파티션을 생성
    ii. 기존 파티션을 재조정 (파티션 구성표 변경, 즉 모든 데이터가 새 위치로 이동)
    iii. downtime 발생

VI. index

1. key 와 pointer로 구성 (5.jpg)
2. 검색속도 빨라짐 : key로 검색
3. 추가적인 저장공간 필요 : index (key, pointer) 저장
4. 쓰기/업데이트 속도 느려짐 : 데이터에 key가 추가되어 사이즈가 커짐
5. 쓰기/업데이트 작업이 많고 검색 작업이 적은 시스템에는 부적합

VII. proxy

1. proxy
    a. 클라이언트는 플록시 서버에 접속, 웹페이지/파일/연결과 같은 서비스 요청
    b. 다른 서버의 리소스를 찾는 클라이언트의 요청에 대한 중계 역할을 하는 hw 혹은 sw
    c. 요청을 필터링/로그/변환 (헤더 추가/제거, 암호화/해독, 압축)
    d. 요청 많은 파일 캐쉬 가능
    (ex) 어려 클라이언트가 특정 리소스에 액서스 하는 경우 -> 플록시 서버는 원격캐시로 이동하지 않고도 캐시하여 모든 클라이언트에게 제공 (6.jpg)
2. proxy server type
    a. closed proxy
        i. 네트워크 그룹 내에 사용
        ii. dsn/웹페이지 같은 인터넷 서비스를 저장/전달 하여 대역폭을 줄임/제어
    b. open proxy - 누구나 접속 가능
        i. anonymous proxy : 서버공개, ip비공개
        ii. transparent proxy : 사용자는 proxy server 존재는 알지 못함, proxy를 통해 사용자 ip 확인 가능 (웹서버 보호용)
        iii. reverse proxy
            - client 심부름꾼 : 대신하여 타 서버의 자원 검색 후 결과값을 리턴
            - 검색 당한 서버는 client가 아닌 proxy가 요청한 것으로 인식

VIII. redundancy and replication (**중복 및 복제**)

1. redundancy
    a. **서비스 백업**
    b. **시스템 안정성과 성능을 높이기 위함**
    c. **단일 장애 지점을 제거 - 백업 서비스 제공**
2. replication (7.jpg)
    a. **파일 백업**
    b. DBMS**에서 주로 사용**
    c. **마스터-슬레이브 관계**
        i. **마스터 : 모든** update**를 받은 후 슬레이브로** sync
        ii. **슬레이브 : 업데이트 종료 후 마스터에게 성공 메세지 전송**

IX. sql vs nosql

1. sql (structured query language, 구조화된 질의어)
    a. 관계형 데이터 베이스 - join 사용
    b. row(행)과 column(열)에 저장
    c. mysql, ms-sql, oracle
2. nosql
    a. 비관계형 데이터베이스
    b. type
        i. key-value stores
            - data는 key, value 가 쌍으로 저장
            - redis, danamo, voldemort
        ii. document databases
            - table 대신 document 가 저장
            - 하나의 collection 에는 다수의 document가 있음
            - 각 문서는 전혀 다른 구조를 갖을 수 있음 (동적스키마)
            - mongodb, couchdb
        iii. wide-column databases
            - 컬럼 테이블에서 '테이블' 대신 컬럼 패밀리 (행의 컨테이너)
            - RDBMS처럼 모든 열을 미리 알 필요가 없음
            - 각 행에 동일한 열이 있을 필요도 없음
            - 큰 데이터 세트를 분석하는데 가장 적합
            - cassandra, hbase
        iv. graph databases
            - 그래프로 표현하기 가장 적합한 데이터를 저장
            - 데이터 : node (entity), 속성 (property, entity에 대한 정보) 및 선 (entity 간에 연결) 이 있는 그래프 구조로 저장
            - neo4j, infinitegraph
3. sql 과 nosql의 차이점
    a. storage
        i. sql : 행과 열이 있는 테이블
        ii. nosql : 키-값, 문서, 그래프
    b. schema

i. sql : **고정 스키마**
ii. nosql : **동적 스키마**

c. query
i. sql : sql**을 이용하여 데이터를 정의하고 조작** (select, join, ...)
ii. nosql : **문서 모음에 중점**, UnQL (Unstructed Query Language, db**시스템 마다 다름**)

d. scaling (**확장**)
i. sql : **수직확장, 서버 성능 올림**
ii. nosql : **수평확장, 서버 갯수 늘림**

e. reliability (**신뢰성**) or acid compliancy (atomicity, consistency, isolation, durability)
i. sql : acid **보장, 데이터/트랜젝션 안정성 보장 높음**
ii. nosql : **성능 및 확장을 위해** acid **준수 희생**
[설명] ACID는 db 트랜젝션이 안전하게 수행된다는 것을 보장하기 위한 4가지 성질을 말함

4. sql vs nosql
a. sql **사용**
i. acid **준수를 보장해야 할 경우**
- **데이터/트랜젝션 무결성 보호**
- **전자상거래, 금융 애플리케이션**

ii. **데이터가 구조적이고 변화가 적을 경우**

b. nosql **사용**
i. **데이터의 병목현상 방지**
ii. **구조가 거의 없거나 전혀없는 대량의 데이터 저장**
iii. **분산 환경 적합 - 클라우드 컴퓨팅 및 스토리지 활용 극대**
iv. **빠른 개발 가능 - 사전 준비 없음, 버전간 중단 없이 데이터 구조 변경 용이**

X. cap theorem (cap 이론)

1. cap 중 두개 이상 동시에 제공 할 수 없음을 증명한 이론
2. cap
    a. consistency (일관성)
        i. 모든 노드가 같은 시간에 같은 데이터를 보여줘야 함 - 동일성
        ii. 추가 읽기를 허용하기 전에 여러 노드를 업데이트하여 일관성 유지
    b. availability (가용성)
        i. 모든 요청은 성공, 실패에 대한 응답을 받음
        ii. 몇몇 노드가 다운되어도 다른노드들에 영향을 주어서는 안됨 - 독립성
    c. partition tolerance (분리 내구성)
        i. 노드간에 통신 문제가 생겨 메세지를 주고 받지 못하는 상황이라도 동작해야 함
        ii. 일부 메세지가 손실되어도 시스템은 정상 동작 해야 함 - 생존성
        iii. availibility vs partition tolerance
            - availability : 특정 노드(서버)가 장애가 있어도 없어야 함
            - partition tolerance : 특정 네트워크가 장애가 있어도 문제 없야 함
3. cap 중 두가지를 가진 시스템만 구축가능 (모두를 만족하는 시스템은 존재하지 않음, 8.jpg)
    a. ca (동일성 + 독립성)
        i. 시스템이 죽을지언정 메세지 손실은 있을 수 없음
        ii. 트랜젝션이 필요한 경우
        iii. RDBMS
    b. cp (동일성 + 생존성)
        i. 모든 노드가 함께 성능을 내야 함
        ii. BigTable, MongoDB, HBase
    c. ap (독립성 + 생존성)
        i. 비동기화 된 서비스 스토어에 적합
        ii. Cassandra, couchDB

XI. consistent hashing (일관된 해싱)

1. key based sharding
    a. hash function
        i. key % number of server = server index
        (ex) 22 % 21 = 1
    b. 문제
        i. 수평확장 불가능 - 새 서버 추가시 기존 매핑 손상
        ii. 각 서버에 데이터가 균일하게 배포되지 않음
    c. 해결
        i. consistent hashing, repartition
2. consistent hashing (일관된 해싱)
    a. 서버 추가/제거 시 재구성을 최소화하여 쉽게 확장/축소 가능
    b. 동작방식
        i. 서버목록이 정해지면 해당 범위 내에 index 값으로 해시
        ii. 결과값을 서버에 mapping
            - 정수로 hash 함
            - 첫번째 서버가 발견될 때까지 시계방향으로 이동
            - (9.jpg ~ 13.jpg)

XII. long-polling vs websockets vs server-sent events

1. http (14.jpg)
    a. 클라이언트는 연결을 열고 서버에게 데이터 요청 (request)
    b. 서버는 클라이언트에게 응답 (response)
2. ajax polling (15.jpg)
    a. 클라이언트는 연결을 열고 서버에게 데이터 요청 (http 사용)
    b. 서버는 클라이언트에게 Ajax Component가 있는 web page (html) 를 응답 (response)
    c. 요청된 웹페이지는 일정간격 (ex 0.5초) 으로 서버에게 데이터 요청
    d. 서버는 클라이언트에게 응답
    e. c - d 반복
3. long polling
    a. 클라이언트는 연결을 열고 서버에게 데이터 요청 (http 사용)

b. **서버**
    i. **응답할 데이터가 있을 때 : 클라이언트에게 즉시 응답**
    ii. **응답할 데이터가 없을때**
        - **시간초과가 발생 전 : 응답대기**
        - **시간초과 발생 : a 작업 수행**

4. web socket
    a. **전 이중 방식 지원 : 클라이언트와 서버는 서로 평등하게 통신**
    b. tcp **통신, 클라이언트와 서버는 지속적인 연결 지원**
    c. **동작방식 :** websocket handshake **를 통해 클라이언트와 서버를 연결 성공 후 언제든지 실시간 양방향 통신이 가능**
5. sse (server sent events)
    a. **지속적, 장기적 연결 지원**
    b. **서버는** sse**를 통해 클라이언트에게 데이터를 보냄**
    c. **클라이언트는** sse**를 통해 데이터를 보낼 수 없음 - 다른 기술/프로토콜을 통해 데이터를 보내야 함**
    d. **실시간 전송 가능, 루프에서 데이터 생성 후 여러 이벤트를 보내는 경우 좋음**