4주차 정리

6장. 키-값 저장소 설계

1) 키-값 저장소란?

: 저장되는 값이 고유 식별자를 키로 갖는 저장소

- 키-값 쌍: 키와 값 사이의 연결 관계
- 키는 유일해야 하며 해당 키에 매달린 값은 키를 통해서만 접근 가능
- 키는 일반 텍스트일 수도, 해시 값일 수도 있음
    - 성능 상의 이유로, 키는 짧을 수록 좋음
- 값은 문자열, 리스트, 객체일 수 있음
    - 값으로 무엇이 오든 상관하지 않음

```
put(key, value): 키-값 쌍을 저장소에 저장함
get(key): 인자로 주어진 키에 매달린 값을 꺼냄
```

1. 문제 이해 및 설계 범위 확정

- 읽기, 쓰기 그리고 메모리 사용량 사이에 어떤 균형을 찾고, 데이터의 일관성과 가용성 사이에서 타협적 결정을 내린 설계를 만들었다면 괜찮은 답안이 될 것

우리가 만들 키-값 저장소의 특징

- 키-값 쌍의 크기는 10KB 이하이다.
- 큰 데이터를 저장할 수 있어야 한다.
- 높은 가용성을 제공해야 한다. 따라서 시스템은 설사 장애가 있더라도 빨리 응답해야 한다.

- 높은 규모 확장성을 제공해야 한다. 따라서 트래픽 양에 따라 자동적으로 서버 증설/삭제가 이루어져야 한다.
- 데이터 일관성 수준은 조정이 가능해야 한다.
- 응답 지연시간이 짧아야 한다.

2. 단일 서버 키-값 저장소

키-값 쌍 전부를 메모리에 해시 테이블로 저장하는 방법

1) 특징

- 가장 직관적

2) 장점과 단점

장점

- 빠른 속도를 보장함

단점

- 모든 데이터를 메모리 안에 두는 것이 불가능할 수 있음
    - **개선책**
        - 데이터 압축
        - 자주 쓰이는 데이터만 메모리에 두고 나머지는 디스크에 저장

→ 개선책을 사용해도 한계가 존재하므로 많은 데이터를 저장하기 위해서는 분산 키-값 저장소를 만들어야 함

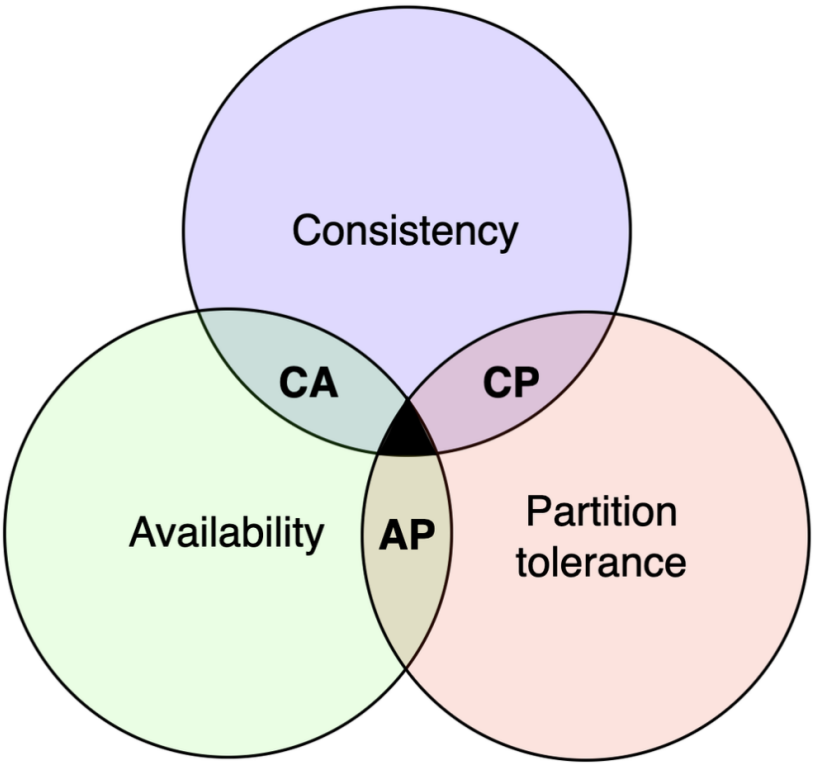
3. 분산 키-값 저장소

- 키-값 쌍을 여러 서버에 분산 시키기에 분산 해시 테이블이라고 불리기도 함

1) CAP 정리

: 데이터 일관성(Consistency), 가용성(availability), 파티션 감내(Partition tolerance)라는 세 가지 요구사항을 동시에 만족하는 분산 시스템을 설계하는 것은 불가능하다는 정리

- 데이터 일관성: 분산 시스템에 접속하는 모든 클라이언트는 어떤 노드에 접속했느냐에 상관 없이 언제나 같은 데이터를 보게 되어야 함
- 가용성: 분산 시스템에 접속하는 클라이언트는 일부 노드에 장애가 발생하더라도 항상 응답 받을 수 있어야 함
- 파티션 감내: 파티션은 두 노드 사이에 통신 장애가 발생하였음을 의미. 파티션 감내는 네트워크에 파티션이 생기더라도 시스템은 계속 동작해야 함을 의미.



2) 키-값 저장소 분류

: 위의 세가지 요구 사항 중 어느 두 가지를 만족하느냐에 따라 아래와 같이 분류 가능

- CP 시스템: 일관성과 파티션 감내를 지원하는 키-값 저장소. 가용성 희생
- AP 시스템: 가용성과 파티션 감내를 지원하는 키-값 저장소. 데이터 일관성 희생
- CA 시스템: 일관성과 가용성을 지원하는 키-값 저장소. 파티션 감내는 지원하지 않음
    - 그러나 통상 네트워크 장애는 피할 수 없음 → 분산 시스템은 반드시 파티션 문제를 감내할 수 있어야 함
        - CA 시스템은 실세계에서 존재하지 않는 시스템이 됨

3-1. 이상적 상태

가정: 세 대의 복제 노드 n1, n2, n3에 데이터를 복제하여 보관하는 상황

- 이상적인 환경이라면 네트워크가 파티션 되는 상황은 절대 발생X
    - n1에 기록된 데이터는 자동적으로 n2와 n3에 복제 → 데이터 일관성과 가용성 만족

3-2. 실세계의 분산 시스템

- 분산 시스템은 파티션 문제를 피할 수 없음
    - 파티션 문제 발생 시, 우리는 일관성과 가용성 중 한 가지를 선택해야 함

예시

- 만약 n3에 장애가 발생하여 n1 및 n2와 통신할 수 없는 상황이 될 경우, 클라이언트가 n1 혹은 n2에 기록한 데이터는 n3에 전달되지 X
- n3에 기록되었으나 아직 n1이나 n2로 전달 되지 않은 데이터가 있다면 n1과 n2는 오래된 사본을 가지고 있을 것
    - 일관성을 선택한 경우: 세 서버 사이에 생길 수 있는 데이터 불일치 문제를 피하기 위해 n1과 n2에 대해 쓰기 연산을 중단
        - 가용성이 깨짐
    - 가용성을 선택한 경우: 과거 데이터를 반환할 위험이 있더라도 계속 읽기 연산을 허용
        - n1과 n2역시 쓰기 연산을 허용, 파티션 문제가 해결된 뒤에 새 데이터를 n3에 전송

3-3. 시스템 컴포넌트

1) 핵심 컴포넌트 및 기술

- 데이터 파티션
- 데이터 다중화

- 일관성
- 일관성 불일치 해소
- 장애 처리
- 시스템 아키텍처 다이어그램
- 쓰기 경로
- 읽기 경로

2) 데이터 파티션

데이터를 파티션 단위로 나눌 때 확인해야 하는 문제

- 데이터를 여러 서버에 고르게 분산할 수 있는가
- 노드가 추가되거나 삭제될 때 데이터의 이동을 최소화할 수 있는가

→ 안정 해시를 사용하여 문제 해결 가능

안정해시 사용의 장점

- 규모 확장 자동화: 시스템 부하에 따라 서버가 자동으로 추가되거나 삭제되도록 만들기 O
- 다양성: 각 서버의 용량에 맞게 가상 노드의 수를 조정할 수 O

3) 데이터 다중화

- 높은 가용성과 안전성 확보를 위해서는 데이터를 N개 서버에 비동기적으로 다중화해야 함
    - N: 튜닝 가능한 값

N개의 서버를 선정하는 방법

- 어떤 키를 해시 링 위에 배치
- 해당 지점으로부터 시계 방향으로 링을 순회, 만나는 첫 N개 서버에 데이터 사본을 보관
- 단, 가상 노드 사용 시에는 선택한 N개의 노드가 대응될 실제 물리 서버의 개수가 N보다 작아질 수 있음
    - 노드를 선택할 때 같은 물리 서버를 중복 선택하지 않도록 하여 문제 회피

4) 데이터 일관성

- 여러 노드에 다중화된 데이터는 적절히 동기화 되어야 함 → 정족수 합의 프로토콜

정족수 합의 프로토콜

N: 사본개수

W: 쓰기 연산에 대한 정족수. 쓰기 연산이 성공한 것으로 간주되려면 적어도 W개의 서버로부터 쓰기 연산이 성공했다는 응답을 받아야 함

R: 읽기 연산에 대한 정족수. 읽기 연산이 성공한 것으로 간주되려면 적어도 R개의 서버로부터 응답 받아야 함

- W, R, N의 값을 정하는 것은 응답 지연과 데이터 일관성 사이의 타협점을 찾는 전형적인 과정
  - W=1 or R=1인 경우: 중재자가 하나의 서버로부터 응답을 받으면 됨 → 응답 속도 ↑
  - W > 1 or R > 1인 경우: 시스템이 보여주는 데이터 일관성 수준 ↑ 응답속도 ↓
  - W+R > N인 경우: 강한 일관성이 보장됨

면접시에 정할 수 있는 W, N, R 구성

- R=1, W = N: 빠른 읽기 연산에 최적화된 시스템
- W =1 , R = N: 빠른 쓰기 연산에 최적화된 시스템
- W + R > N: 강한 일관성이 보장됨(보통 N= 3, W=R=2)
- W + R < N: 강한 일관성이 보장되지 않음

⇒ 요구되는 일관성 수준에 따라 값 조정

5) 일관성 모델

: 키 값 저장소 설계 시 고려해야 하는 중요 요소. 데이터 일관성의 수준을 결정함

일관성 모델 종류

- 강한 일관성: 모든 읽기 연산은 가장 최근에 갱신된 결과를 반환.

- 약한 일관성: 읽기 연산은 가장 최근에 갱신된 결과를 반환하지 못할 수도 있음
- 결과적 일관성: 약한 일관성의 한 형태, 갱신 결과가 결국에는 모든 사본에 반영됨
    - 쓰기 연산이 병렬적으로 발생할 경우 일관성이 깨질 수 있음

일관성이 깨진 데이터를 읽지 않는 방법

- 비 일관성 해소 기법: 데이터 버저닝
    - 버저닝: 데이터 변경 시, 해당 데이터의 새로운 버전을 만드는 것
        - 각 버전의 데이터는 변경 불가능
    - 벡터 시계: [서버, 버전]의 순서쌍을 데이터에 매단 것
        - 충돌을 발견하고 자동으로 해결하는 기술
        - 어떤 버전이 선행 버전인지, 후행 버전인지 혹은 다른 버전과 충돌이 있는지를 판단함
            - 클라이언트 구현이 복잡해짐
            - [서버:버전]의 순서쌍 개수가 빠르게 늘어남

6) 장애 처리

6-1) 장애 감지

- 보통 두 대 이상의 서버가 똑같이 서버 A의 장애를 보고 해야 해당 서버에 실제로 장애가 발생했다고 간주함

장애 감지 방법

- 멀티캐스팅 채널 구축 → 비효율적
- 가십 프로토콜 등 분산형 장애 감지 솔루션 사용

6-2) 일시적 장애 처리

- 가십 프로토콜로 장애를 감지한 시스템은 가용성 보장을 위해 필요한 조치를 취해야 함
    - 엄격한 정족수 접근법: 읽기와 쓰기 연산 금지
    - 느슨한 정족수 접근법: 정족수 요구사항을 강제하는 대신, 쓰기 연산을 수행할 W개의 건강한 서버와 읽기 연산을 수행할 R개의 건강한 서버를 해시 링에서 선택

    - 장애 상태인 서버로 가는 요청은 다른 서버가 임시로 맡아서 처리
    - 임시로 쓰기 연산을 처리한 서버가 이에 대한 단서를 남겨둠 (후에 복구 되면 일괄 반영 → 일관성 보장 위해)

  ⇒ 임시 위탁 방법

6-3) 영구 장애 처리

- 반-엔트로피 프로토콜을 구현하여 문제를 해결
  - 반-엔트로피 프로토콜: 사본들을 비교하여 최신 버전으로 갱신하는 과정을 포함
    - 사본 간 일관성이 망가진 상태 탐지 및 전송 데이터 양을 줄이는 데에 머클 트리 사용
  - 머클 트리: 각 노드에 그 자식 노드들이 보관된 값의 해시, 혹은 자식 노드들의 레이블로부터 계산된 해시 값을 붙여두는 트리
    - 대규모 자료 구조의 내용을 효과적이면서 보안상 안전한 방법으로 검증 가능

6-4) 데이터 센터 장애 처리

- 데이터를 여러 데이터 센터에 다중화 해놓기

7) 시스템 아키텍처 다이어그램

아키텍처 주요 기능

- 클라이언트는 키-값 저장소가 제공하는 두 가지 단순한 API, 즉 get(key) 및 put(key, value)와 통신함
- 중재자는 클라이언트에게 키-값 저장소에 대한 프락시 역할을 하는 노드
- 노드는 안정해시의 해시링 위에 분포
- 노드를 자동으로 추가 혹은 삭제할 수 있도록 시스템은 완전히 분산됨
- 데이터는 여러 노드에 다중화됨
- 모든 노드가 같은 책임을 지므로, SPOF는 존재하지 않음

8) 쓰기 경로

카산드라의 경우

① 쓰기 요청이 커밋 로그 파일에 기록

② 데이터가 메모리 캐시에 기록

③ 메모리 캐시가 가득차거나 사전에 정의된 어떤 임계치에 도달할 경우, 데이터는 디스크에 있는 SStable에 기록됨

SSTable이란?

: Sorted-String Table, <키, 값>의 순서쌍을 정렬된 리스트 형태로 관리하는 테이블

9) 읽기 경로

- 읽기 요청을 받은 노드는 데이터가 메모리 캐시에 있는지 확인한다.
    - 있는 경우: 해당 데이터를 클라이언트에게 반환
    - 없는 경우: 디스크에서 가져옴
        - 블룸 필터를 사용

7장. 분산 시스템을 위한 유일 ID 생성기 설계

1. 1단계-문제 이해 및 설계 범위 확정

1) 유일 ID 설계시 요구사항

- ID는 유일해야 함
- ID는 숫자로만 구성되어야 함
- ID는 64비트로 표현될 수 있는 값이어야 함
- ID는 발급 날짜에 따라 정렬 가능해야 함
- 초당 10,000개의 ID를 만들 수 있어야 함

2. 2단계-개략적 설계안 제시 및 동의 구하기

유일 ID 설계 방법

- 다중 마스터 복제
- UUID

- 티켓 서버
- 트위터 스노플레이크 접근법

2-1. 다중 마스터 복제

1) 다중 마스터 복제 동작 과정

- 데이터베이스의 auto_increment 기능을 활용, 다음 ID의 값을 K만큼 증가하는 방법
  - 이때, K는 현재 사용 중인 데이터베이스 서버의 수가 됨
- 데이터베이스의 수를 늘리면 초당 생산 가능 ID 수도 늘릴 수 있으므로 규모 확장성 문제를 어느정도 해결 가능

2) 장점과 단점

단점

- 여러 데이터 센터에 걸쳐 규모를 늘리기 어려움
- ID의 유일성은 보장되나 그 값이 시간 흐름에 맞추어 커지도록 보장할 수 없음
- 서버 추가 혹은 삭제 시에도 잘 동작하도록 만들기 어려움

2-2. UUID

1) UUID 동작 과정

- UUID: 컴퓨터 시스템에 저장되는 정보를 유일하게 식별하기 위한 128비트 수 → 충돌 가능성이 지극히 낮음
- 웹 서버가 별도의 ID 생성기를 통해 독립적으로 ID를 만들어냄

2) 장점과 단점

장점

- UUID를 만드는 것은 단순 → 서버 사이의 조율이 필요 없으므로 동기화 이슈 없음
- 각 서버가 자신이 사용할 ID를 알아서 만드는 구조, 규모 확장이 쉬움

단점

- ID가 128비트로 김

- ID를 시간순으로 정렬할 수 없음
- ID에 숫자가 아닌 값이 포함될 수 있음

2-3. 티켓 서버

1) 티켓 서버 동작 과정

- auto_increment 기능을 갖춘 데이터베이스 서버, 즉 티켓 서버를 중앙 집중형으로 하나만 사용

2) 장점과 단점

장점

- 유일성이 보장되는 오직 숫자로만 구성된 ID를 쉽게 만들 수 있음
- 구현하기 쉽고, 중소 규모 애플리케이션에 적합

단점

- 티켓 서버가 SPOF가 됨
  - 해당 이슈를 피하려면 티켓 서버를 여러 대 준비해야 함 → 데이터 동기화 등의 새로운 문제 발생

2-4. 트위터 스노플레이크 접근법

1) 트위터 스노플레이크 접근법 특징

- 트위터가 사용하는 ID 생성 기법
- 생성해야 하는 ID 구조를 여러 절로 분할함

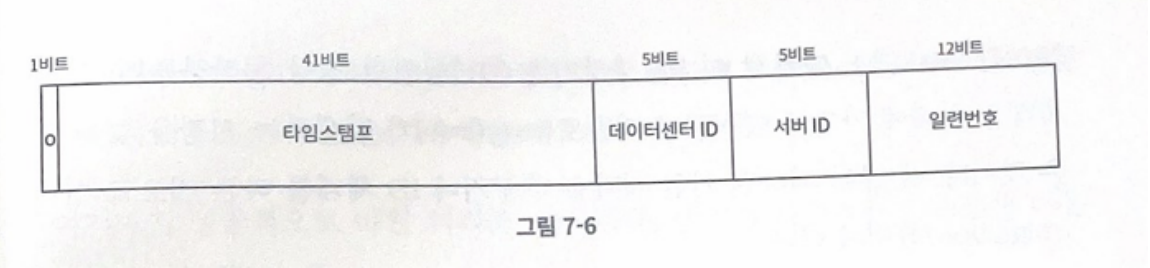


그림 7-6

  - 사인 비트: 1비트를 할당, 음수 양수 구별에 사용
  - 타임스탬프: 41비트 할당. 기원 시각 이후 몇 밀리초가 경과했는지 나타내는 값

- 데이터센터 ID: 5비트 할당. 데이터 센터 당 32개의 서버 사용 가능
- 일련번호: 12비트를 할당. 각 서버에서 ID를 생성할 때마다 일련 번호를 1만큼 증가, 1밀리초가 경과할 때 마다 0으로 초기화

3. 3단계-상세 설계

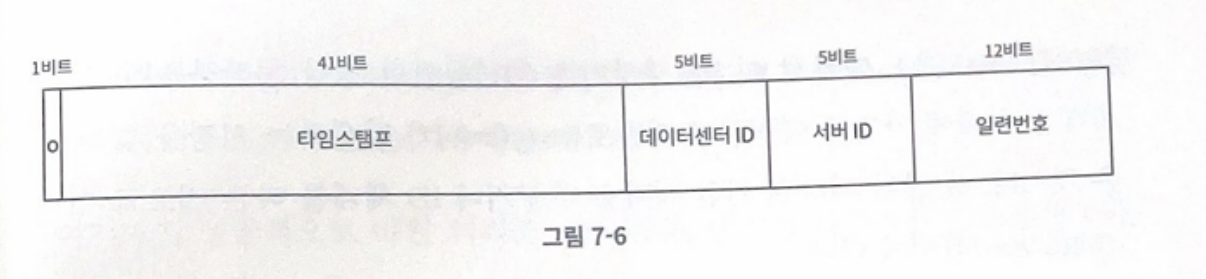


그림 7-6

- 데이터 센터 ID와 서버 ID는 시스템이 시작할 때 결정, 일반적으로 시스템 운영 중에는 변경되지 않음
  - 데이터 센터 ID와 서버 ID를 잘못 변경하면 ID 충돌이 발생할 수 있으므로 해당 작업 시 신중하게 해야 함
- 타임스탬프, 일련번호는 ID 생성기가 돌고 있는 중에 만들어지는 값

3-1. 타임스탬프

- 타임스탬프는 시간이 흐름에 따라 점점 큰 값을 가지게 되므로 ID를 시간 순으로 정렬하는 것이 가능
- **이진 표현 형태에서 UTC 시각 추출 방법**
  - 역으로 사용 시, UTC 시각을 타임스탬프 값으로 변경 가능

  ① 이진 표현 형태를 십진수로 변환

  ② 변경된 십진수에 트위터 기원 시각 더하기

  ③ 결과로 얻어진 밀리초 값을 UTC시각으로 변환
- 41비트로 표현할 수 있는 타임스탬프의 최대값 = 2^41-1 = 약 69년
  - 69년이 지나면 기원 시각을 바꾸거나 ID 체계를 다른 것으로 이전해야 함

3-2. 일련 번호

- 2^12=4096개의 값을 가질 수 있음
- 어떤 서버가 같은 밀리초 동안 하나 이상의 ID를 만들어 낸 경우에만 0보다 큰 값을 가지게 됨

4. 4단계-마무리

1) 추가 논의 사항

- 시계 동기화
- 각 절의 길이 최적화
- 고가용성