6️⃣

6장. 키-값 저장소 설계

키-값 저장소

> **키-값 저장소 == 키-값 데이터베이스**라고도 불리는 비 관계형 데이터베이스이다.

이 저장소에 저장되는 값은 고유 식별자를 키로 가져야 한다.

키는 일반 텍스트, 혹은 해시 값이 수도 있다. 성능상의 이유로 짧을수록 좋다.

ex)

- 일반 텍스트 키 : “last_logged_in_at”
- 해시 키 : 253DDEC4

값은 문자열, 리스트, 객체 일 수도 있다.

다음 연산을 지원하는 키-값 저장소를 설계해 볼 것이다.

- put(key, value) ; 키-값 쌍을 저장소에 저장
- get(key): 인자로 주어진 키에 대한 값을 조회

문제 이해 및 설계 범위 확정

다음 특성을 갖는 키-값 저장소 설계 예정

- 키-값 쌍의 크기는 10KB 이하
- 큰 데이터 저장 가능
- 높은 가용성 제공해야 함. 설사 장애가 있더라도 빨리 응답해야 한다.
- 높은 규모 확장성 제공. 트래픽 양에 따라 자동적으로 서버 증설/삭제가 이뤄져야 한다.

- 데이터 일관성 수준은 조정이 가능해야 한다.
- 응답 지연시간이 짧아야 한다.

단일 서버 키-값 저장소

가장 직관적인 방법 → 키-값 쌍 전부를 메모리에 해시 테이블로 저장

빠른 속도를 보장하지만 모든 데이터를 메모리 안에 두는 것은 불가능

다음 개선책 사용 가능

- 데이터 압축
- 자주 쓰이는 데이터만 메모리에 두고 나머지는 디스크에 저장

그러나, 이렇게 개선하더라도 한 대 서버로는 부족한 때가 찾아온다. 많은 데이터 저장을 위해서는 분산 키-값 저장소를 만들 필요가 있다.

분산 키-값 저장소

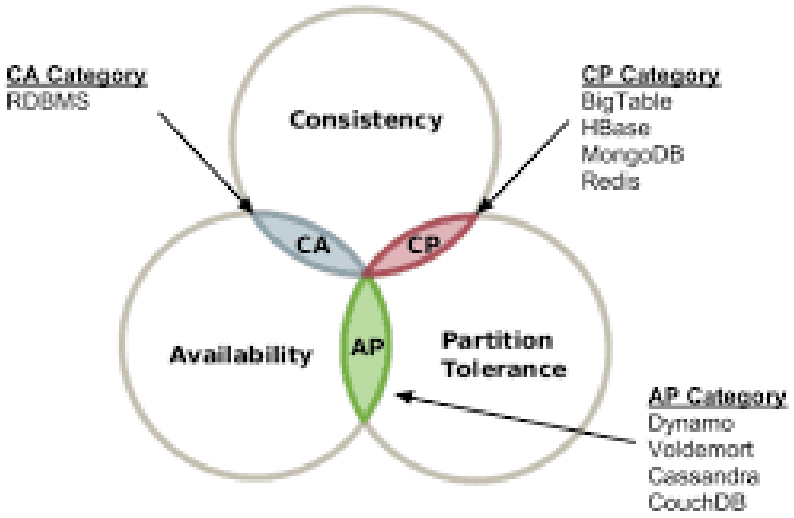
> 분산 키-값 저장소 == 분산 해시 테이블

키-값 쌍을 여러 서버에 분산시키기 때문

분산 시스템을 설계할 때는 **CAP 정리(Consistency, Availability, Partition Tolerance theorem)**를 이해하고 있어야 한다.

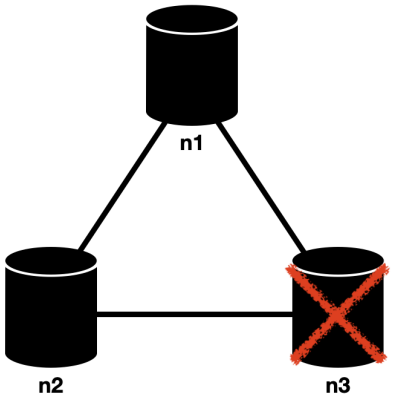
CAP 정리

> 어떤 두 가지를 충족하려면 나머지 하나는 반드시 희생되어야 한다.



- Consistency : 데이터 일관성
  - 모든 클라이언트는 어떤 노드에 접속했느냐에 관계없이 **언제나 같은 데이터**를 봐야 한다.
- Availability: 가용성
  - 일부 노드에 **장애가 발생하더라도 항상 응답**을 받을 수 있어야 한다.
- Partition Tolerance: 파티션 감내
  - 파티션 == 두 노드 사이에 통신 장애가 발생
  - 파티션 감내는 네트워크에 **파티션이 생기더라도 시스템은 계속 동작**해야 한다.

실세계의 분산 시스템



- 이상적 환경이라면 네트워크가 파티션되는 상황은 절대 일어나지 않는다.
  - n1에 기록된 데이터는 자동적으로 n2와 n3에 복제
- 분산 시스템은 파티션 문제를 피할 수 없다.
  - 일관성, 가용성 사이에 하나를 선택해야한다.
- n3에 기록되었으나 아직 n1 및 n2로 전달되지 않은 데이터가 있다면 n1과 n2는 오래된 사본을 갖고 있을 것이다.

- if 일관성 선택? CP 시스템
  - 세 서버 사이에 생길 수 있는 데이터 불일치 문제를 피하기 위해 n1과 n2에 대해 쓰기 연산을 중단해야한다.
  - 그렇게 하면 가용성이 깨진다.

- if 가용성 선택? AP 시스템
  - 설사 낡은 데이터를 반환할 위험이 있더라도 계속 읽기 연산을 허용해야한다.
  - n1, n2는 계쏙 쓰기 연산을 허용 → 파티션 문제가 해결된 뒤에 새 데이터를 n3에 전송

요구사항에 맞도록 CAP 정리를 적용해야한다. 면접관과 상의하고, 그 결론에 따라 시스템을 설계하도록 하자.

시스템 컴포넌트

키-값 저장소 구현에 사용될 핵심 컴포넌트 및 기술들

- 데이터 파티션
- 데이터 다중화
- 일관성
- 일관성 불일치 해소
- 장애 처리
- 시스템 아키텍쳐 다이어그램
- 쓰기 경로
- 읽기 경로

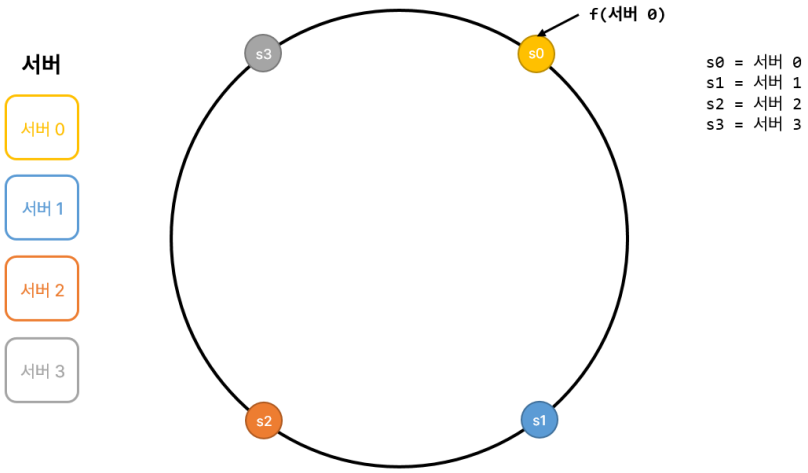
데이터 파티션, 다중화

전체 데이터를 한 대 서버에 욱여넣는 것은 불가능 → 데이터들을 작은 파티션들로 분할한 다음 여러 대 서버에 저장

다음 두 가지 문제를 중요하게 따져봐야 한다.

- 데이터를 여러 서버에 고르게 분산할 수 있는가 → 가상 노드
- 노드가 추가되거나 삭제될 때 데이터의 이동을 최소화 할 수 있는가 → 규모 확장 자동화

안정 해시 기술을 사용하여 위의 문제들 해결 가능



또한, 높은 가용성과 안정성을 위해 데이터를 N개 서버에 비동기적으로 다중화할 필요 있다.

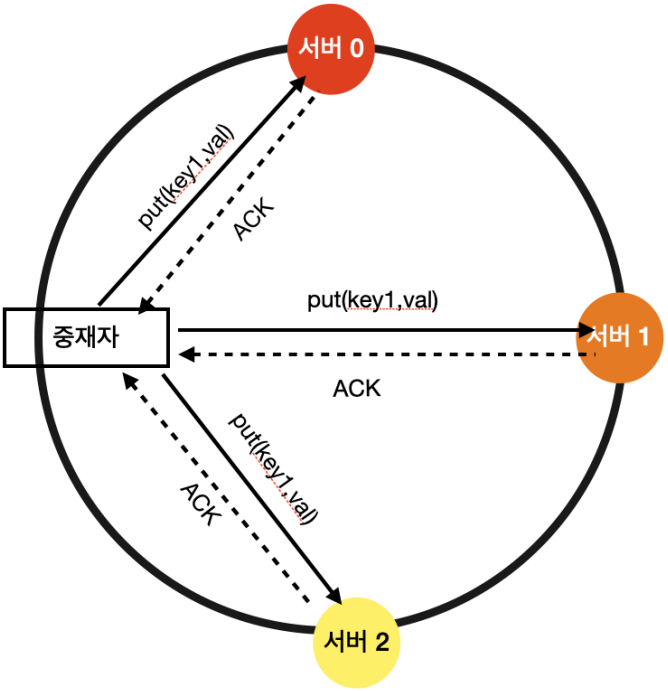
시계 방향으로 링을 순회하면서 만나는 첫 N개 서버에 데이터 사본 보관

데이터 일관성

> **정족수 합의 프로토콜**을 사용하여 읽기/쓰기 연산 모두에 일관성 보장

- N = 사본 개수
- W = 쓰기 연산에 대한 정족수. W개 서버로부터 쓰기 연산이 성공해야함
- R = 읽기 연산에 대한 정족수. R개 서버로부터 읽기 연산이 성공해야함

N = 3 인 경우, 다음과 같다.



중재자 : 클라이언트와 노드 사이에서 프락시 역할을 수행

N, W, R 값 정하기 기준

- R = 1, W = N : 빠른 읽기 연산에 최적화
- W = 1, R = N : 빠른 쓰기 연산에 최적화
- W + R > N : 강한 일관성 보장 (보통 N = 3, W = R = 2)
- W + R ≤ N : 강한 일관성 보장 X

요구되는 일관성 수준에 따라 W, R, N 값 조정하면 된다.

일관성 모델

- 강한 일관성을 위한 일반적인 방법 → 모든 사본에 현재 쓰기 연산의 결과가 반영될 때까지 해당 데이터에 대한 읽기/쓰기를 금지
- 해당 방법은 고가용성 시스템에는 부적합 → 새로운 요청 처리가 중단되기 때문
    - 다이나모, 카산드라 : 100% 낡은 데이터 보지 못하는 것은 아니지만, 결국에는 갱신 결과가 모든 사본에 반영(동기화)되는 형태 지님 → 최종 일관성 모델

- 최종 일관성 모델: 쓰기 연산 병렬 수행 → 일관성 깨지는 것을 방지하기 위해 클라이언트 쪽에서 **데이터의 버전 정보**를 활용

비 일관성 해소 기법 : 데이터 버저닝

> **벡터 시계** : [서버, 버전] 의 순서쌍을 데이터에 매달기

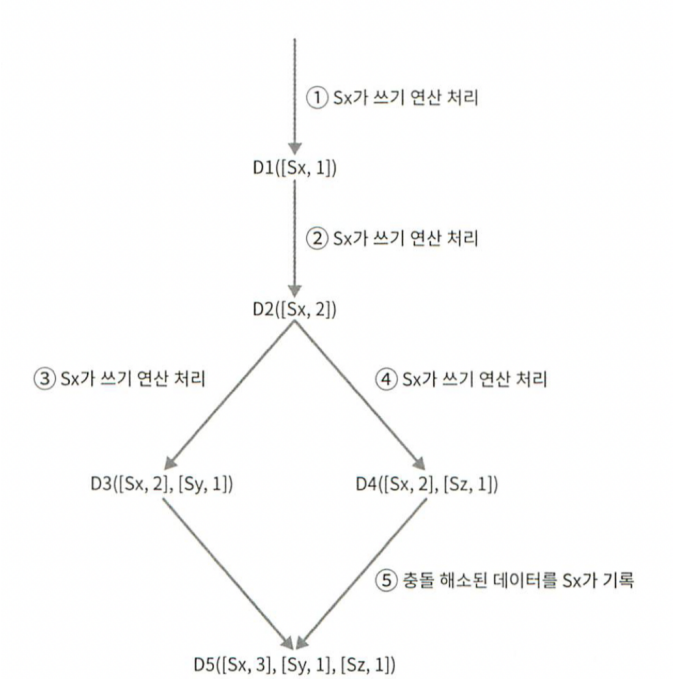
버전 순서 및 다른 버전과 충돌이 있는지 판별하는데 사용된다.

표현 형식 : D([S1, v1], [S2, v2], ... , [Sn, vn])

- D : 데이터
- vi : 버전 카운터
- Si : 서버 번호

데이터 D를 서버 Si에 기록하면, 서버는 아래 작업 가운데 하나를 수행해야 한다.

- [Si, vi] 가 있으면 vi 증가
- 없으면 새 항목 [Si, 1] 생성



어떤 클라이언트가 D3 와 D4 읽으면, 데이터 간 충돌이 있다는 것을 알게 된다.

D2를 Sy와 Sz가 각기 다른 값으로 바꿨기 때문

이 충돌은 클라이언트가 해소한 후에 서버에 기록한다.

충돌 감지 방법

버전 Y에 포함된 모든 구성요소의 값이 X에 포함된 모든 구성요소 값보다 같거나 큰지만 보면 된다.

ex)

- D([s0, 1], [s1, 1])은 D([s0, 1], [s1, 2])의 이전 버전 → 따라서 두 요소간 충돌 존재 X
- D([s0, 1], [s1, 2])와 D([s0, 2], [s1, 1])은 서로 충돌

단점

- 충돌 감지 및 해소 로직이 클라이언트에 들어가야 하므로, 클라이언트 구현이 복잡해진다.
- [서버: 버전]의 순서쌍 개수가 굉장히 빨리 늘어난다.

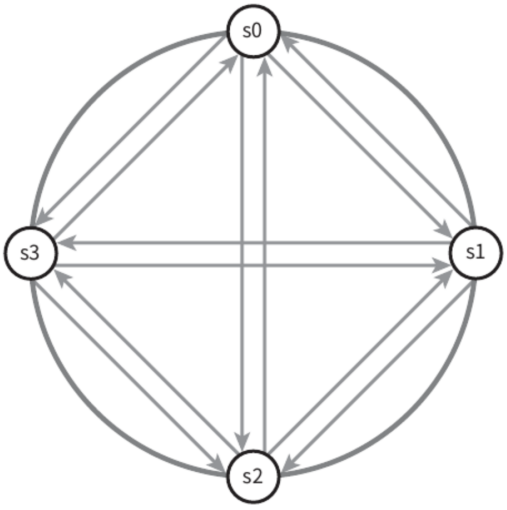
- 다이나모 db에 관계된 문헌에 따르면 아마존은 실제 서비스에서 임계치를 넘어서는 문제 발생한 적 없다고 한다.

장애 처리

> 장애 감지, 장애 해소

장애 감지

보통 두 대 이상의 서버가 똑같이 하나의 서버의 장애를 보고해야 해당 서버가 실제로 장애가 발생했다고 간주

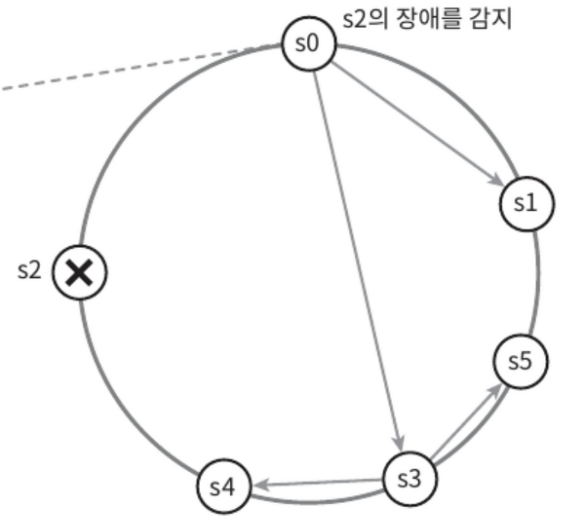


멀티캐스팅 채널 구축

하지만 해당 방법은 서버가 많아지면 비효율적 → **가십 프로토콜** 같은 분산형 장애 감지 솔루션이 효율적

s0's membership list

Member ID	Heartbeat counter	Time
0	10232	12:00:01
1	10224	12:00:10
2	9908	11:58:02
3	10237	12:00:20
4	10234	12:00:34



- 각 노드는 멤버십 목록을 유지. 멤버십 목록은 멤버ID와 그 박동 카운터 쌍의 목록
- 각 노드는 주기적으로 자신의 박동 카운터를 증가
- 각 노드는 무작위로 선정된 노드들에게 주기적으로 자기 박동 카운터 목록을 보냄
- 박동 카운터 목록을 받은 노드는 멤버십 목록을 최신 값으로 갱신
- 어떤 멤버의 박동 카운터 값이 일정 시간 동안 갱신되지 않으면 해당 멤버는 장애 상태인것으로 간주

일시적 장애 처리

- 엄격한 정족수 접근법 : 데이터 일관성을 위해 읽기와 쓰기 연산 금지해야함
- 느슨한 정족수 접근법 : 윗 조건을 완화하여 가용성 높인다.

→ 정족수 요구사항을 강제하는 대신, 쓰기 연산을 수행할 W개의 건강한 서버와 읽기 연상을 수행할 R개의 건강한 서버를 해시 링에서 고른다.

→ 이때 장애 상태인 서버는 무시

→ 장애 상태인 서버로 가는 요청은 다른 서버가 잠시 맡아 처리

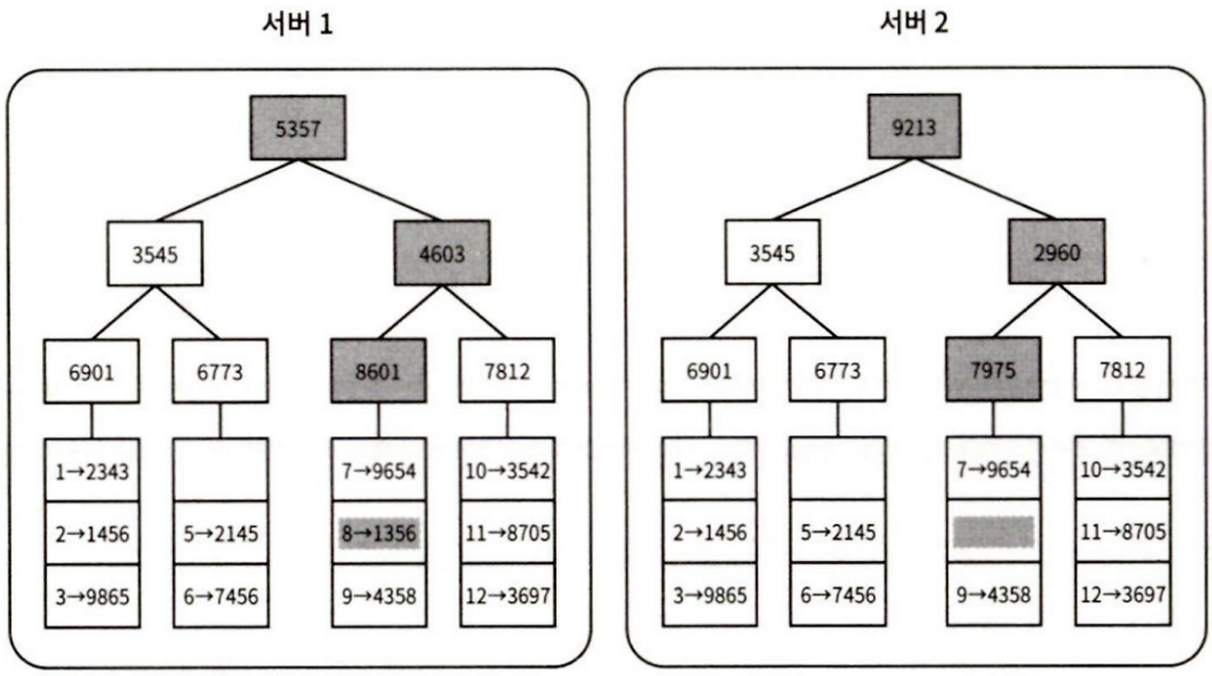
→ 임시로 쓰기 연산을 처리한 서버는 이에 관한 힌트를 남겨, 나중에 서버 복구 완료 후 일괄 반영하여 데이터 일관성 보존

→ 해당 장애 처리 방안을 **단서 후 임시 위탁 기법**이라 부른다.

영구 장애 처리

> **반-엔트로피(anti-entropy) 프로토콜**을 구현하여 사본들을 동기화한다.

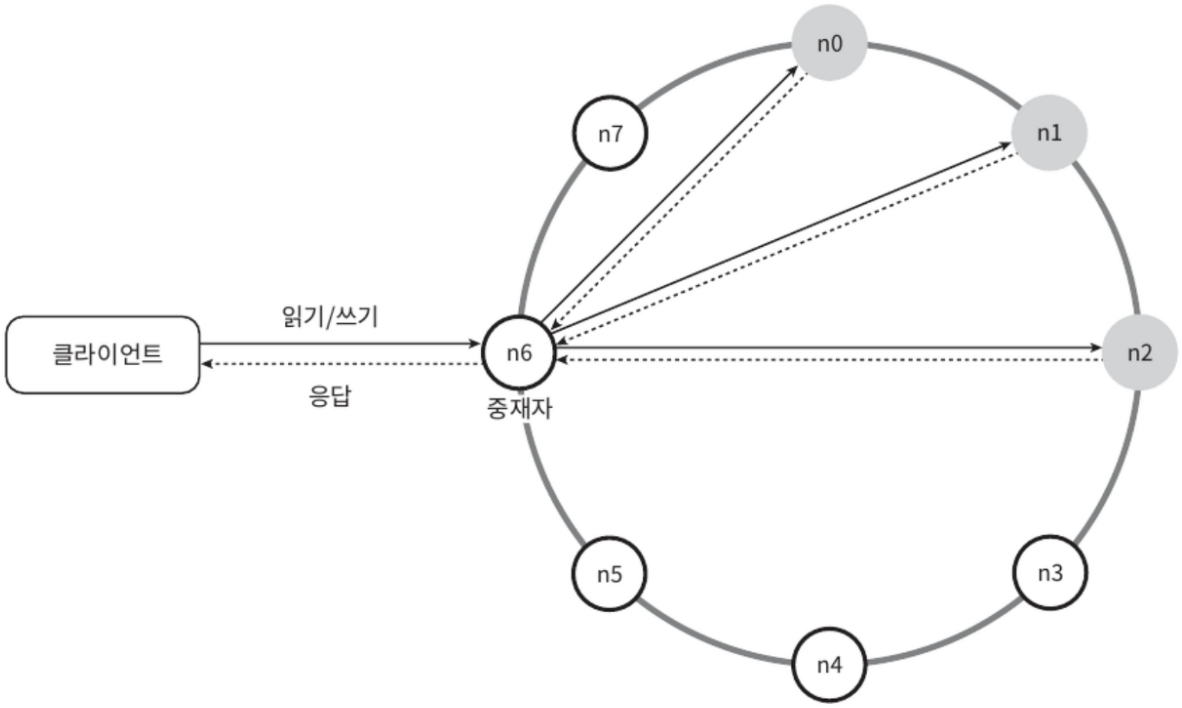
- 사본들을 비교하여 최신 버전으로 갱신하는 과정을 포함
- 사본 간 일관성이 망가지는 상태를 탐지하고 전송 데이터의 양을 줄이기 위해 머클(Merkle) 트리를 사용
- 머클 트리는 각 노드에 그 자식 노드들에 보관된 값의 해시를 레이블로 붙여두는 트리. 해시 트리를 사용하면 대규모 자료 구조의 내용을 효과적이면서 보안상 안전한 방법으로 검증 가능.



색칠된 부분 : 데이터가 망가져서 일관성이 보장되지 않는 해시값

루트 노드의 해시값부터 시작해서 자식 노드 해시값을 비교해서 다른 해시값을 가진 버킷을 찾아서 동기화한다.

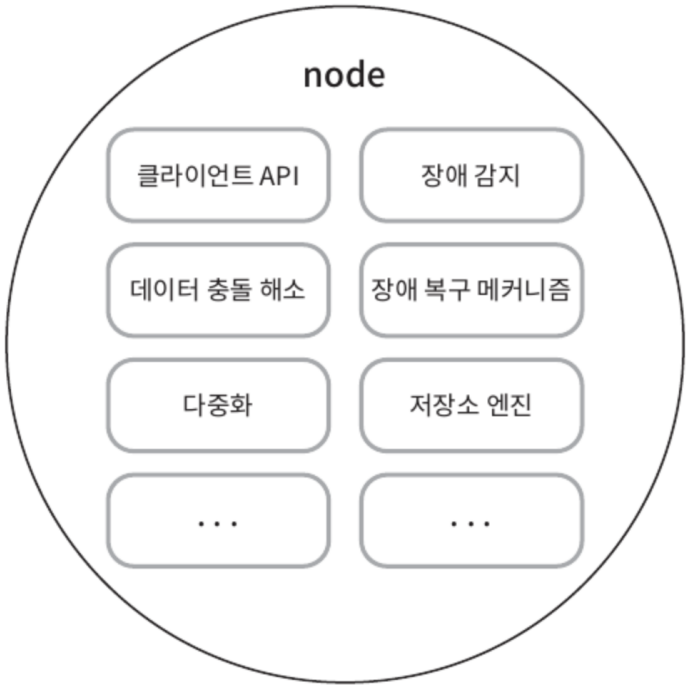
시스템 아키텍쳐 다이어그램



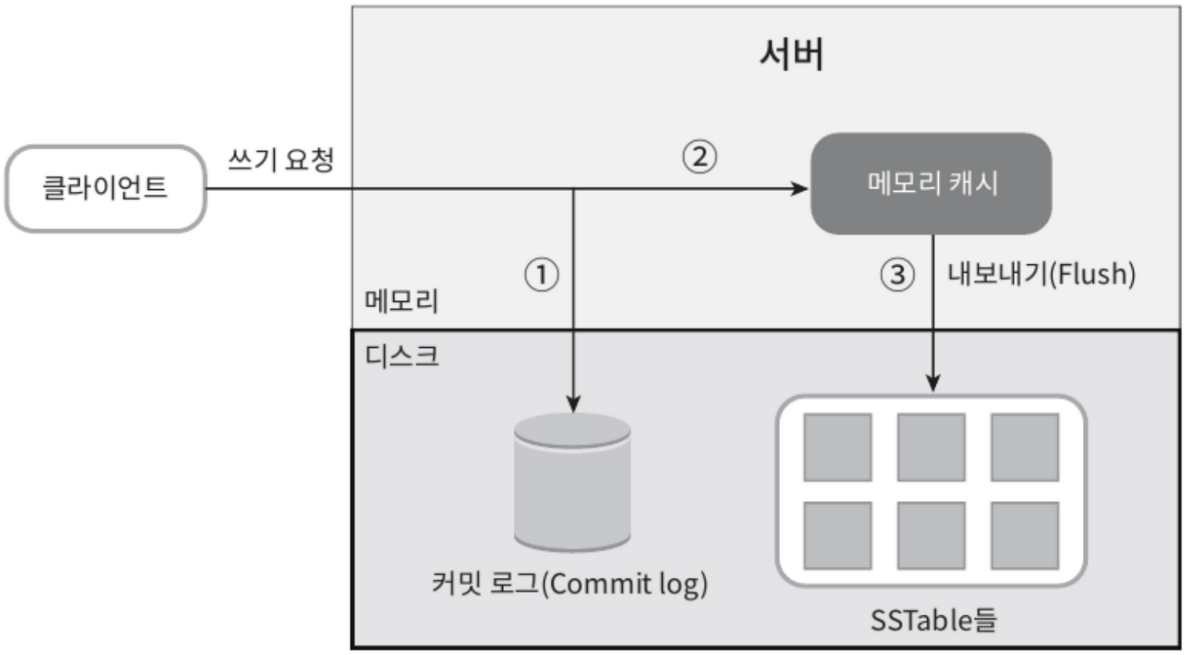
아키텍쳐의 주된 기능

- 클라이언트는 키-값 저장소가 제공하는 2가지 단순한 API, get(k), put(k,v)와 통신
- 중재자는 클라이언트에게 키-값 저장소에 대한 프락시 역할을 하는 노드
- 노드는 안정 해시의 해시 링 위에 분포
- 노드를 자동으로 추가 또는 삭제할 수 있도록, 시스템은 완전히 분산
- 데이터는 여러 노드에 다중화된다
- 모든 노드가 같은 책임을 진다 → SPOF(Singe Poing Of Failure)는 존재X

완전히 분산된 설계를 채택하였으므로, 모든 노드는 아래의 기능을 전부 지원해야한다.



쓰기 경로

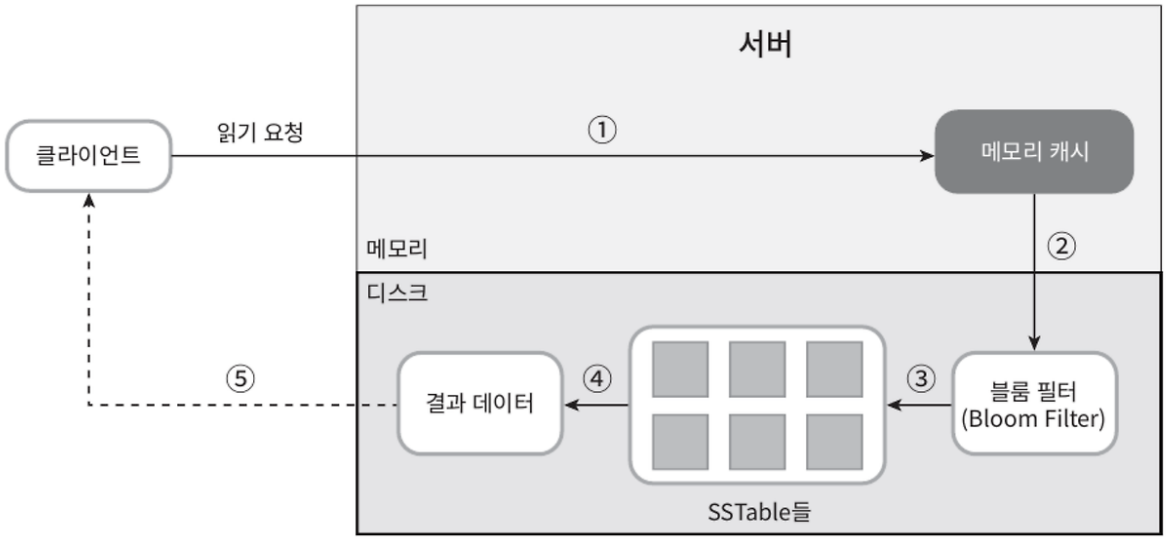


1. 쓰기 요청이 커밋 로그 파일에 기록된다.

2. 데이터가 메모리 캐시에 기록된다.

3. 메모리 캐시가 가득찼거나 사전에 정의된 임계치에 도달하면 데이터는 디스크에 있는 SSTable에 기록된다. SSTable(Sorted-String Table)은 <키,값>의 순서쌍을 정렬된 리스트 형태로 관리하는 테이블

읽기 경로



1. 데이터가 메모리에 있는지 검사. 없으면 2로 간다.
2. 블룸 필터(어느 SSTable에 찾는 키가 있는지 알아낼 효율적인 방법)를 검사
3. 키가 보관된 SSTable 찾아냄
4. SSTable에서 데이터 가져온다.
5. 해당 데이터를 클라이언트에게 반환

요약

목표/문제	기술
대규모 데이터 저장	안정 해시를 사용해 서버들에 부하 분산
읽기 연산에 대한 높은 가용성 보장	데이터를 여러 데이터센터에 다중화
쓰기 연산에 대한 높은 가용성 보장	버저닝 및 벡터 시계를 사용한 충돌 해소
데이터 파티션	안정 해시
점진적 규모 확장성	안정 해시
다양성	안정 해시
조절 가능한 데이터 일관성	정족수 합의
일시적 장애 처리	느슨한 정족수 프로토콜과 단서 후 임시 위탁
영구적 장애 처리	머클 트리
데이터 센터 장애 대응	여러 데이터 센터에 걸친 데이터 다중화

참고링크

https://dongwooklee96.github.io/post/2021/03/26/cap-이론이란/

https://velog.io/@mmy789/System-Design-6

https://imnotabear.tistory.com/644?category=1124689