# 6장

- 키 값 저장소는 non-relational 데이터베이스이고 identifier를 키로 가져가 value를 가지는 pair를 저장하는 데이터베이스이다.

키 값 저장소의 요구사항

- 10KB미만의 key pair size
- 큰 데이터 저장 가능
- high availability
- high extendable
- 조정 가능한 일관성 레벨
- 짧은 지연 시간

단일 서버 키-값 저장소

- 모든 값을 메모리 안에 두는 방향성이 존재함
- 하지만 크기에 따라 불가능할 수 있으니, 데이터를 압축하거나 or 자주 쓰이는 데이터만 메모리에 두고 디스크에 저장하는 방식을 택할 수 있음

분산 키 값 저장소

- CAP 정리
  - Consistency, Availability, Partition Tolerance 라는 3가지 요구사항을 동시에 만족하는 분산 시스템을 설계하는 것은 불가능
    - Consistency : 어떤 노드에 접근해도 언제나 같은 값을 볼 수 있어야함
    - Availability : 일부 노드에 장애가 발생해도 응답을 받을 수 있어야함
    - Partition tolerance : 네트워크 파티션이 생겨도 시스템은 동작해야 함
  - 어느 두가지를 만족하려면 하나는 희생해야함
    - CP : 일관성과 파티션 감내 - 가용성 희생
    - AP : 가용성과 파티션 감내 지원 - 일관성 희생

        - CA : 일관성 가용성 지원 - 파티션 감내 지원안함. 네트워크 장애는 흔하므로 데이터베이스는 반드시 P를 지원해야함. 일반적으로 존재하지 않는 케이스

- 시스템 컴포넌트
    - 데이터 파티션
        - 대규모 애플리케이션의 경우 한 노드에 모든 데이터를 보관하는 것은 불가능
            - 데이터를 여러 서버에 고르게 분산
            - 노드가 추가 삭제될 때 데이터의 이동을 최소화 → 안정해시
        - 안정해시의 장점
            - 규모 확장 자동화 (automatic scaling) → 시스템 부하에 따라 서버가 자동으로 추가되거나 삭제되도록 만들 수 있음
            - 다양성(heterogeneity) : 각 서버의 용량에 맞게 가상 노드의 수를 조정할 수 있음 → 고성능 서버는 더 많은 가상 노드를 갖도록 설정 가능
    - 데이터 다중화
        - 높은 가용성과 안정성을 확보하기 위해 데이터를 N개의 서버로 비동기적으로 다중화(replication)할 수 있음.
        - 가상 서버를 사용해 배치하는 경우 물리적으로 같은 서버를 참조할 수도 있으니 노드를 선택할 때 같은 물리 서버를 선택하지 않도록 해야함.
    - 일관성
        - 데이터는 적절히 동기화되어야 함.
        - 정족수 합의 (Qourum Consensus) 프로토콜을 사용하면 읽기/쓰기 연산 모두에 일관성을 보장할 수 있음
            - N = 사본의 개수
            - W = 쓰기 연산에 대한 정족수 - 성공한 것으로 간주되려면 적어도 W개의 서버로부터 연산 성공 응답을 받아야함
            - R = 읽기 연산에 대한 정족수 - 읽기 연산이 성공한 것으로 간주되려면 적어도 R개의 서버로부터 응답을 받아야
        - 일반적인 구성들
            - R = 1, W = N : 빠른 읽기 연산에 최적화
            - W = 1, R = N : 빠른 쓰기 연산에 최적화
            - W + R > N ：강한 일관성이 보장됨 (보통 N = 3, W = R = 2)

        - W + R ≤ N : 강한 일관성 보장 안됨
    - 일관성 모델
        - 강한 일관성 : 모든 읽기 연산은 가장 최근에 갱신된 결과 반환 보장
            - 사본에 모든 쓰기 연산의 결과가 반영될 때까지 해당 데이터에 대한 읽기/쓰기를 금지하는 것
            - 고 가용성 모델에는 적합하지 않음
        - 약한 일관성 : 읽기 연산은 가장 최근 갱신 결과를 반환하지 못할 수 있음
        - 최종 일관성 : 약한 일관성의 한 형태 - 갱신 결과가 결국에는 모든 사본에 반영되는 모델
            - 다이나모나 카산드라가 선택하고 있는 모델
- 일관성 불일치 해소
    - 데이터 버저닝
        - 어떤 쓰기 연산에 대한 결과로 데이터의 새로운 버전이 생긴다. 버전은 변경 불가능하다.
        - 하지만 동시에 각각의 노드에 쓰기 연산을 한다면 동일한 버전이 양 노드에 생길 수 있다. → conflict
    - 벡터 시계
        - [서버, 버전] 의 순서쌍을 데이터에 매단 것
        - 어떤 버전이 선행인지 후행인지, 다른 버전과 충돌이 있는지 판별하는데 쓰임

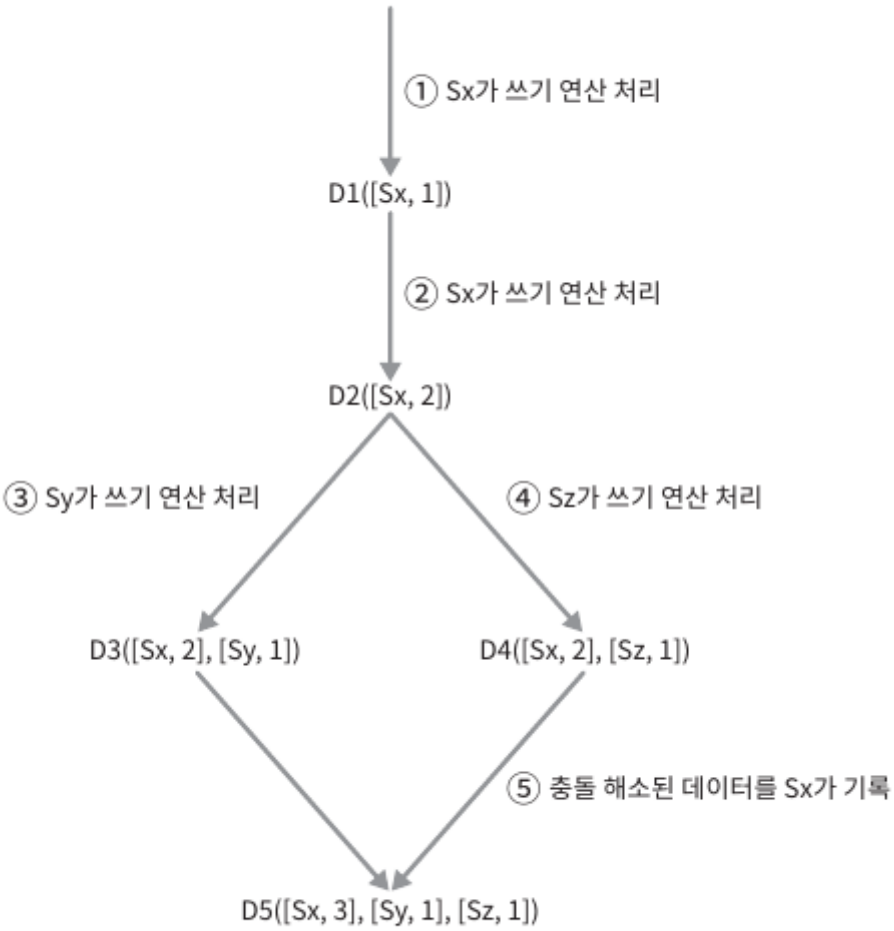



그림 6-9

- 버전 X와 버전 Y가 있을 때
  - 버전 X가 Y보다 이전 버전인지 (X < Y) 판별하는 방법 : X의 구성요소가 Y의 구성요소보다 작은지 판별하면 됨
    - D([S0, 1], [S1, 1]) < D([S0, 2], [S1, 2])
  - X와 Y사이에 충돌이 있는지 보려면(=같은 이전 버전에서 파생된 다른 버전인지)
    - Y의 구성요소 중 X의 구성요소보다 부분적으로 작은 값이 있는지 확인하면 됨
    - note: Y의 모든 구성 요소가 X보다 작다면 Y는 X의 이전버전임
    - D([S0, 1], [S1, 2]) <> D([S0, 2], [S1, 1])
- 단점 :

        - 충돌 감지 및 해소 로직이 클라이언트에 있어야함 → 클라 구현이 복잡해짐
        - [서버: 버전] 순서쌍의 개수가 굉장히 빨리 늘어남 → 물리적 한계 때문에 임계치를 설정하게 되고 임계치를 넘게되면 선후 관계를 정확하게 결정시킬 수 없는 문제가 발생함. 실제 세계에서는 그다지 많이 발생하지는 않음.
- 장애 처리
    - 장애 감지 (failure detection)
        - 멀티 캐스팅 프로토콜을 구성하는게  가장 쉬운 방법, 비효율적

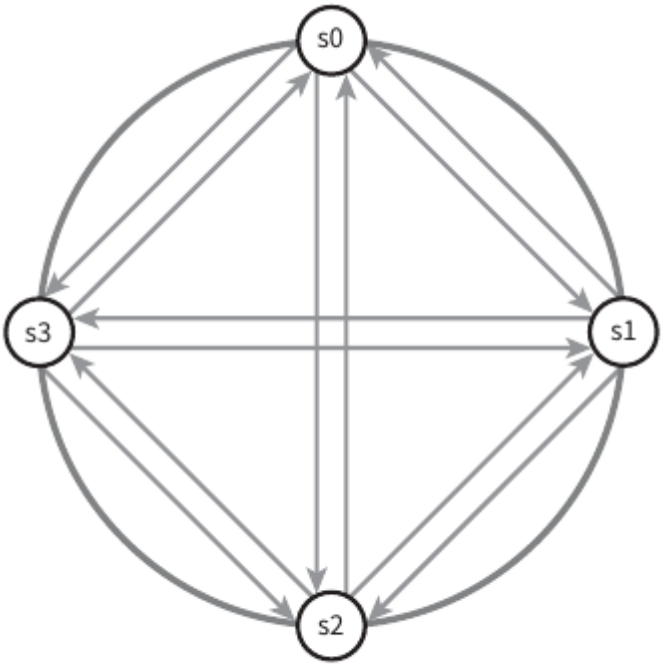



그림 6-10

        - 가십(gossip protocol)와 같은 분산형 장애 감지(decentralized failure detection) 솔루션을 채택하는 것이 효율적
            - 각 노드는 membership list를 유지한다. 보통 멤버 ID와 hearbeat counter의 쌍으로 구성됨
            - 각 노드는 주기적으로 자신의 박동 카운터를 증가시킴
            - 각 노드는 무작위로 선정된 노드들에게 주기적으로 자기 박동 카운터 목록을 보냄

        - 어떤 멤버의 박동 카운터 값이 지정된 시간 동안 갱신되지 않으면 멤버는 장애(offline) 상태인 것으로 간주함.

- 장애 해소 (failure resolution)
    - 일시적 장애 처리
        - 엄격한 정족수 접근법은 전체 장애 상황으로 간주될 수 있음
        - 느슨한 정족수 접근법을 통해 가용성을 높일 수 있음! W개의 건강한 서버와 읽기 연산을 수행할 R개의 건강한 서버를 골라서 잠시 해당하는 서버가 처리하는 것
        - 임시로 쓰기 연산한 서버는 단서(hint)를 남겨두고 (hinted handoff) 기법에 따라서 적재한 후 복구가 되었을 때 인계함.
    - 영구 장애 처리
        - 반 엔트로피(anti-entropy) 프로토콜을 구현하여 사본들을 동기화할 수 있음.
        - 머클(merkle) 트리를 사용하여 일관성이 망가진 상태를 가밎하고 데이터 전송량을 줄일 수 있음.
            - 각 노드의 그 자식노들에 보관된 값의 해시 또는 자식 노드들의 레이블로부터 계산된 해시 값을 레이블로 붙혀두는 트리
            - 보안상 안전한 방법으로 verification을 할 수 있다는 장점이 있다.

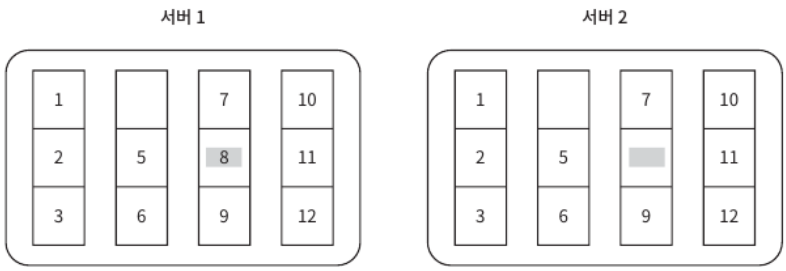



그림 6-13

서버 1과 2의 차이점인 3번째 버킷의 8값을 기준으로 보면된다.

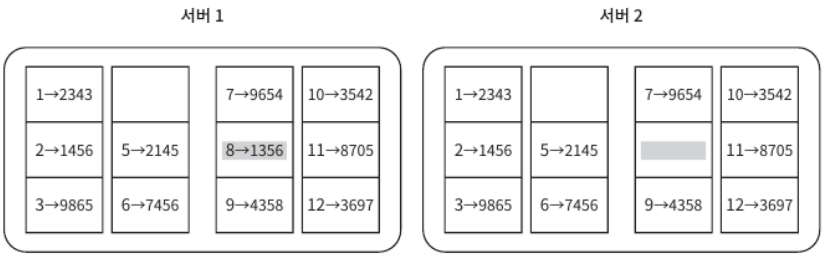



그림 6-14

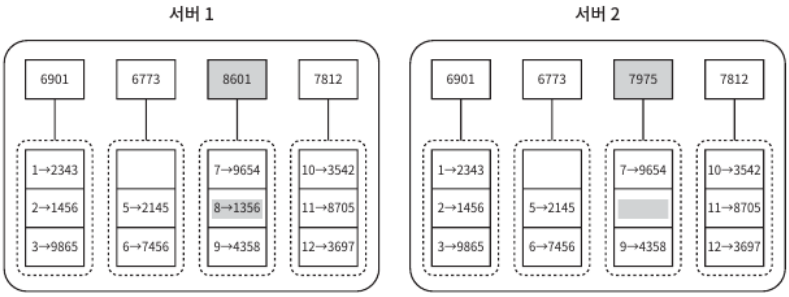



그림 6-15

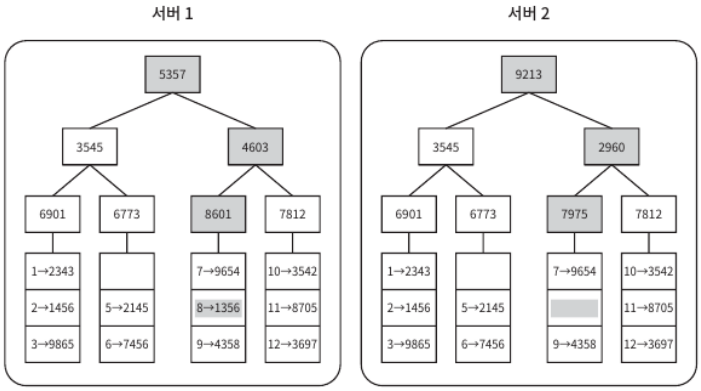



그림 6-16

- 결과적으로 루트의 값이 같다면 서버간의 데이터가 완전히 동일한 것이고 아니라면 개별 노드별로 log2만큼 통신을 하여 차이가 발생하는 버킷을 찾아 동기화하면 된다.

- 시스템 아키텍처 다이어그램

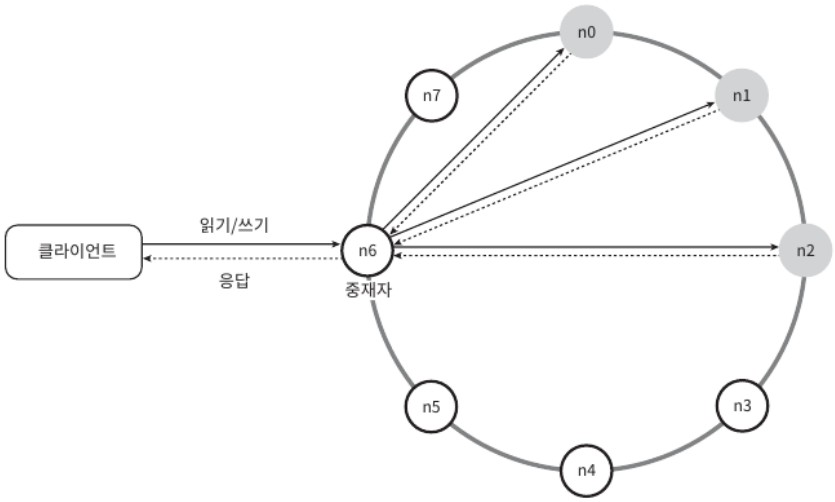



그림 6-17

- 단순한 API로 제공하고
- 중재자(coordinator)는 클라이언트에게 키 값 저장소에 대한 프락시를 제공한다.
- 노드는 안정 해시(consistent hash)의 해시 링 위에 분포된다.
- 노드는 자동으로 추가 또는 삭제할 수 있도록 시스템은 완전히 분산된다.
- 데이터 노드는 여러 노드에 다중화된다.
- 모든 노드가 같은 책임을 지므로 SPOF는 발생하지 않는다.

◦ 쓰기 경로

- 쓰여지는 동안 어떠한일이 발생하는가?

1. 커밋로그를 기록하고
2. 데이터를 메모리 캐시에 기록하고
3. 메모리 캐시가 가득차면 SSTable(Sorted String Table)에 기록한다.

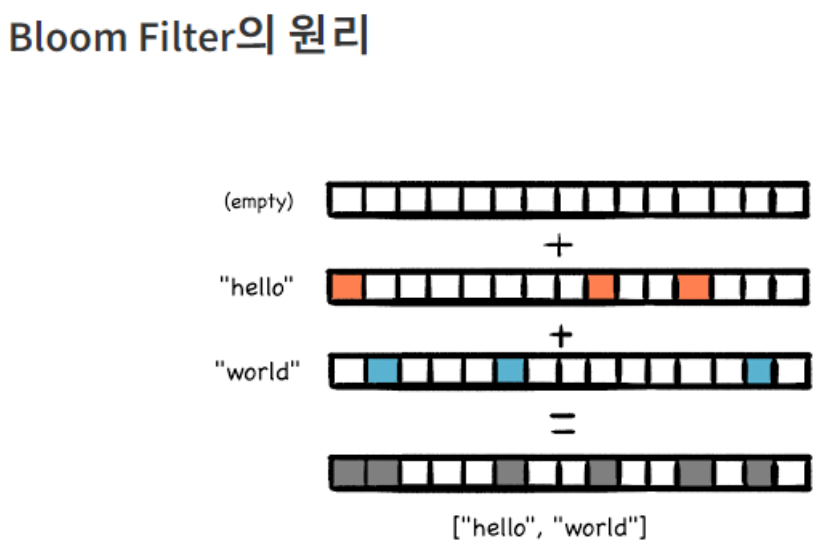
◦ 읽기 경로

1. 메모리 캐시에 있는지 살핀다.
2. 없다면 SSTable에 이를 찾아서 가져온다. 이때 효율적으로 키가 어떤 SSTable에 있는지 알아내야하는데 Bloom filter를 사용한다.

---

# 읽어보기

## Bloom Filter




해시함수의 결과들을 합산하고 검사 대상의 값의 해시를 합산한 결과와 일치하는지 검사하는 방법

1. False Positive : 거짓 긍정 결과를 낼 수 있음
2. None False Negative : 그래도 거짓 부정 결과는 없음.

## SSTable

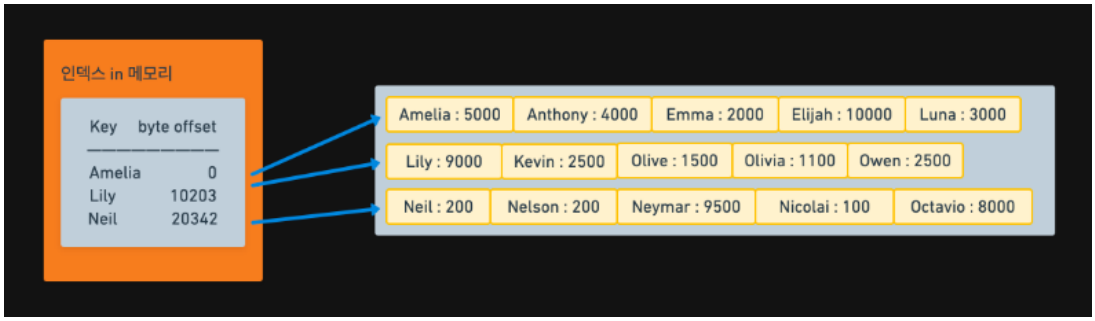
- SSTable를 이용한 데이터 저장 (velog.io)
- sjo200 - CS 대학원 수업 정리 블로그 :: NoSQL SSTable, LSM tree의 구조 (tistory.com)

SSTable은 기본적으로 Segment files로 구성되는데 각 segment에는 sorted key-value가 존재함.

특징

- 병합이 쉬움 정렬되어 있는 두 segment를 하나의 segment로 합칠 때 O(n)의 비용이 소모됨
    - append only 자료구조 이므로 같은 key의 값이 변경된 경우 여러 segment에 같은 key가 존재할 수 있는데, 후순위 segment가 가장 우선시 됨

- sparse index로 데이터 검색을 개선시킬 수 있음.
    - sparse index vs dense index : 일부분의 값들을 인덱스로 가지고 있는 형태




Cassandra에서는..?

- LSM을 인메모리 데이터 저장 자료구조로 활용함
- LSM이 어느정도 차면 이를 SSTable로 flush
- flush를 하는 동안은 새로운 LSM 트리를 만들어서 관리