6장 키-값 저장소 설계

키-값 저장소 (key-value store) 란?

- 키-값 데이터베이스라고도 불리는 비 관계형 (non-relational) 데이터베이스
- 저장되는 값은 고유 식별자 (identifier)를 키로 가져야 한다.
- 키와 값 사이의 연결 관계를 '키-값' 쌍 (pair)라고 한다.
- 키-값 쌍에서 키는 유일해야 하며, 일반 텍스트이거나 해시 값일 수도 있다.
- 키는 기능적으로 짧을수록 좋다.
- 키-값 쌍에서 값은 문자열, 리스트, 객체 등 여러 형태일 수 있다.
- 저장소로 널리 알려진 것은 아마존 다이나모, memcached, 레디스 같은 것들이 있다.

문제 이해 및 설계 범위 확장

`put (key, value)` , `get (key)` 연산을 지원하는 키-값 저장소를 설계해보자.

저장소는 다음과 같은 특성을 가진다.

- 키-값 쌍의 크기는 10KB 이하.
- 큰 데이터를 저장할 수 있어야 한다.
- 높은 가용성 제공해야 한다. 장애가 있더라도 빨리 응답해야 한다.
- 높은 규모 확장성을 제공해야 한다. 트래픽 양에 따라 자동으로 서버 증설/삭제가 이뤄져야 한다.
- 데이터 일관성 수준은 조정이 가능해야 한다.
- 응답 지연시간이 짧아야 한다.

단일 서버 키-값 저장소

한 대 서버만 이용하여 저장소 구현하는 것은 쉽다. 하지만 모든 키-값 쌍을 모두 메모리에 둘 수 없으므로 다음과 같은 개선책을 제시할 수 있다.

- 데이터 압축 (compression)
- 자주 쓰이는 데이터만 메모리에 두고 나머지는 디스크에 저장

하지만 그래도 한 대 서버가 부족한 순간이 찾아오므로 분산 키-값 저장소 (distributed key-value store) 를 만들어야 한다.

분산 키-값 저장소

CAP 정리

아래 세 가지 요구사항을 동시에 만족하는 분산 시스템 설계는 불가능하다는 정리다.

- **데이터 일관성 (consistency)**
  - 분산 시스템에 접속하는 모든 클라이언트는 어떤 노드에 접속했느냐에 관계없이 언제나 같은 데이터를 보게 되어야 한다.
- **가용성 (availability)**
  - 분산 시스템에 접속하는 클라이언트는 일부 노드에 장애가 발생하더라도 항상 응답을 받을 수 있어야 한다.
- **파티션 감내 (partition tolerance)**
  - 파티션은 두 노드 사이에 장애가 발생하였음을 의미.
  - 파티션 감내는 네트워크에 파티션이 생기더라도 시스템은 계속 동작하여야 한다는 것이다.

세 가지 요구사항 가운데 어떤 두 가지를 만족하느냐에 따라 다음과 같이 분류할 수 있다.

- `CP` : 일관성과 파티션 감내 지원. 가용성을 희생.
- `AP` : 가용성과 파티션 감내 지원. 데이터 일관성 희생.
- `CA` : 일관성과 가용성 지원. 파티션 감내는 지원하지 않지만 네트워크 장애는 피할 수 없으므로 실제로 이는 존재하지 않는다.

시스템 컴포넌트

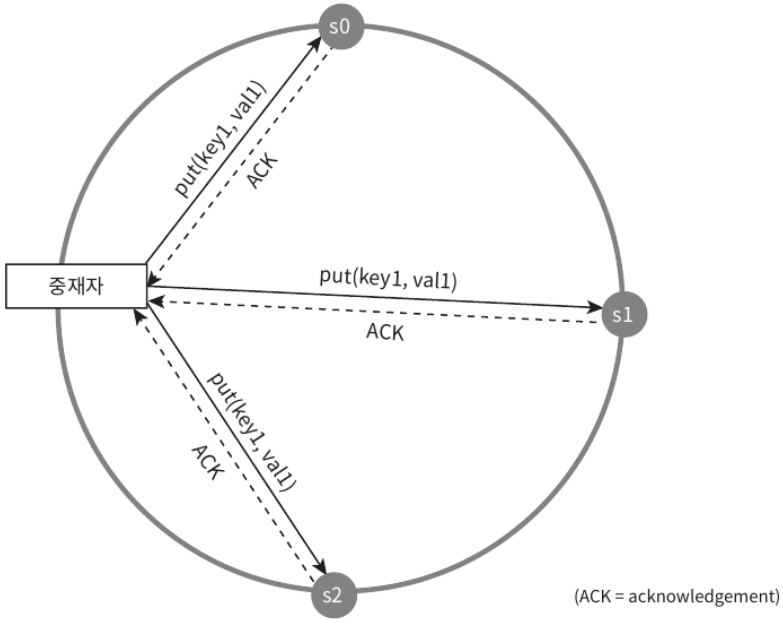
데이터 파티션

- 데이터를 한 서버에 넣는 것을 불가능하므로 작은 파티션들로 분할하여 저장해야 하며 다음을 고려해야 한다.
  - 데이터를 여러 서버에 고르게 분산할 수 있는가
  - 노드가 추가되거나 삭제될 때 데이터의 이동을 최소화할 수 있는가
- 안정 해시를 이용하여 해당 문제를 해결할 수 있다. 다음과 같은 이점이 있다.
  - 규모 확장 자동화 (automatic scaling) : 시스템 부하에 따라 서버가 자동으로 추가되거나 삭제되도록 만들 수 있다.
  - 다양성 (heterogeneity) : 각 서버의 용량에 맞게 가상 노드의 수를 조장할 수 있다. 즉 고성능 서버는 더 많은 가상 노드를 갖도록 할 수 있다.

데이터 다중화

- 높은 가용성과 안정성을 위해 데이터를 N개의 서버에 비동기적으로 다중화 (replication) 한다.
- 해시 링을 시계 방향으로 순회하며 가장 먼저 만나는 N개의 서버에 사본을 보관.
- 가상 노드를 사용한다면 같은 물리 서버를 중복 선택하지 않도록 해야 한다.

데이터 일관성

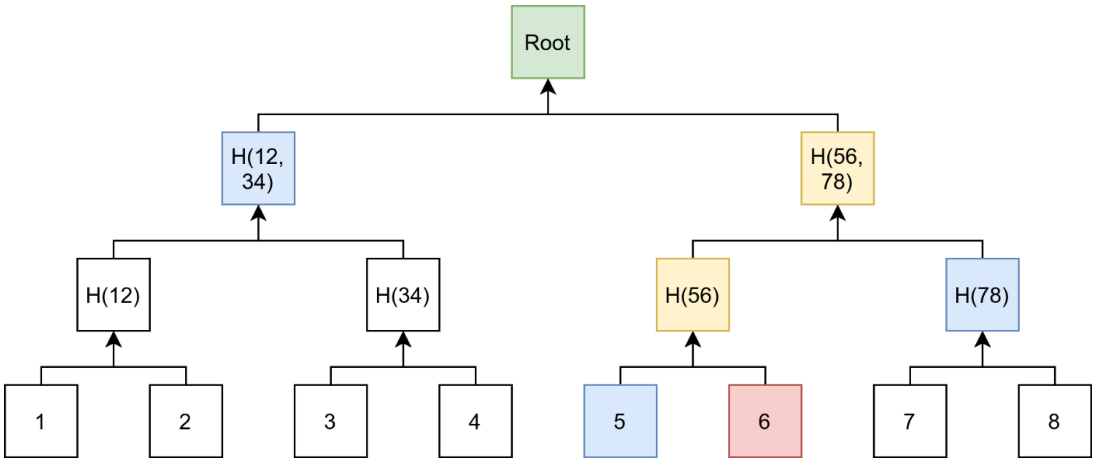


- 여러 노드에 다중화된 데이터는 적절히 동기화 되어야 한다.
- 정족수 합의 (Quorum Consensus) 프로토콜을 통해 읽기/쓰기 모두 일관성을 보장.
  - N = 사본 개수

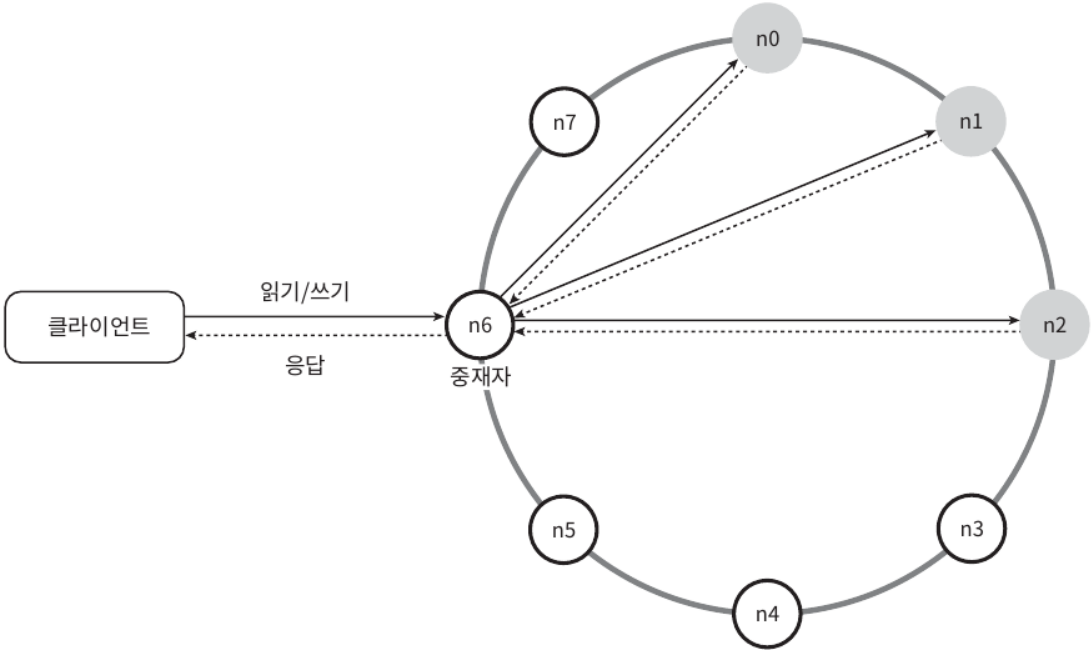
    - W = 쓰기 연산에 대한 정족수. 적어도 W개의 서버로부터 성공 응답 받아야.
    - R = 읽기 연산에 대한 정족수. 적어도 R개의 서버로부터 성공 응답 받아야.
    - 중재자 (coordinator)는 클라이언트와 노드 사이에서 프록시 역할을 한다.
    - N, W, R 값을 정하여 응답 지연과 데이터 일관성의 타협점을 찾아야 한다.
        - R = 1, W = N : 빠른 읽기 연산 최적화.
        - W = 1, R = N : 빠른 쓰기 연산 최적화.
        - W + R > N : 강한 일관성 보장 (보통 N=3, W=R=2)
        - W + R ≤ N : 강한 일관성 보장 안됨.
- 일관성 모델 (consistency model)
    - 강한 일관성 (strong consistency)
        - 모든 읽기 연산은 가장 최근 갱신된 결과 반환. 클라이언트는 절대 out-of-date 데이터를 보지 못한다.
        - 일반적으로 모든 사본이 현재 쓰기가 반영될 때까지 r/w를 금지하면 되지만, 이는 고가용성 시스템에는 적합하지 않다.
    - 약한 일관성 (weak consistency) : 읽기 연산은 가장 최근 갱신 결과를 반환하지 않을 수 있다.
    - 최종 일관성 (eventual consistency) : 약한 일관성의 하나로, 갱신 결과가 결국은 동기화되어 모든 사본에 반영되는 모델이다.
- 비 일관성 해소 기법 : 데이터 버저닝
    - 데이터 버저닝
        - 데이터를 변경할 때마다 해당 데이터의 새로운 버전을 만든다. 이 때 버전은 변경 불가.
        - 하지만 여러 연산이 동시에 일어날 경우 버전 사이의 충돌 발생. 클라이언트에 백터 시계 필요.
    - 백터 시계
        - (서버, 버전) 의 순서쌍을 데이터에 매단 것. 선행 버전 후행 버전 충돌을 판별할 때 쓰임.
        - 백터 시계를 D([s1, v1], [s2, v2] … [sn,vn]) 이라고 하자.
            - D : 데이터, vi : 버전 카운터, si : 서버 번호

        - [si, vi] 있으면 vi 증가
        - 아니라면 [si, 1] 을 만든다.
    - 단점
        - 충돌 감지 및 해소 로직이 클라이언트에 들어가야 하므로 클라이언트 구현 복잡.
        - (서버:버전) 쌍이 매우 빠르게 증가. 임계치를 설정하고 이를 넘어가면 오래된 순서쌍을 제거해야 하는데, 이러면 버전 간 선후 관계가 정확하지 않음.
        - 하지만 이를 활용하는 다이나모 데이터베이스 문헌에 따르면 아마존 실제 서비스에서 그런 문제는 발생하지 않았다고 한다.
- 장애 감지
    - 가십 프로토콜
        - 각 노드는 멤버십 목록을 유지. 맴버십 목록은 각 멤버 ID와 그 박동 카운터 쌍의 목록이다.
        - 각 노드는 주기적으로 자신의 박동 카운터 증가시킴.
        - 각 노드는 무작위로 선정된 노드들에게 주기적으로 자기 박동 카운터 목록 전송.
        - 박동 카운터 받는 노드는 멤버십 목록 갱신.
        - 어떤 멤버의 박동 카운터 값이 지정된 시간 동안 갱신되지 않으면 해당 멤버는 장애 (offline) 상태인 것으로 간주.
- 일시적 장애 처리
    - 가십 프로토콜로 장애 감지한 뒤 정족수 접근법으로 문제 해결.
        - 엄격한 정족수를 사용한다면 읽기와 쓰기 연산 금지해야 한다.
        - 느슨한 정족수를 사용한다면 가용성을 높임.
            - 정족수를 강제하는 대신 쓰기 연산을 수행할 W개의 서버와 읽기 연산 수행할 R개의 서버를 해시링에서 고른다.
    - 네트워크나 서버 문제로 발생한 경우 해당 서버 요청은 다른 서버가 잠시 처리.
        - 임시 위탁 기법 (hinted handoff) : 서버가 복구되었을 때 그 동안의 변경 사항을 일괄 반영. 임시로 쓰기 연산을 수행한 서버에 단서 (hint)를 남겨둔다.
- 영구 장애처리

- `반-엔트로피 (anti entropy) 프로토콜`
  - 사본들을 비교하여 최신 버전으로 갱신하는 과정 포함. 머클 트리를 이용하여 일관성 상태 감지.
- `머클 (Merkle) 트리`
  - 해시 트리라고도 하며, 각 노드에 자식 노드들에 보관된 값의 해시, 또는 자식 노드들의 레이블로부터 계산된 해시값을 레이블로 붙여둔다.



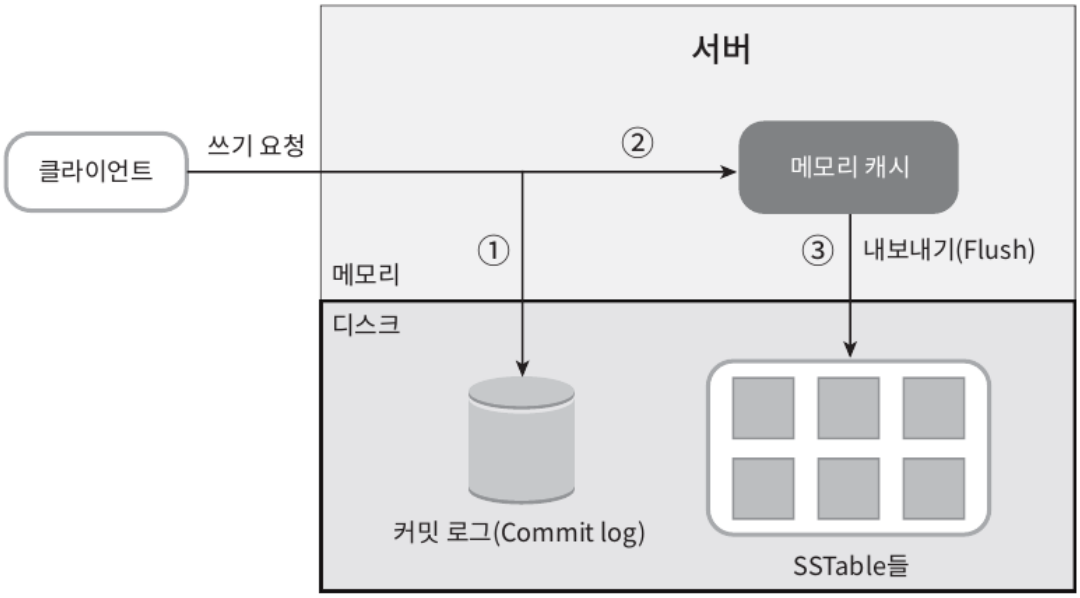
- 시스템 아키텍처 다이어그램



- 다음과 같은 기능이 추가되었다.

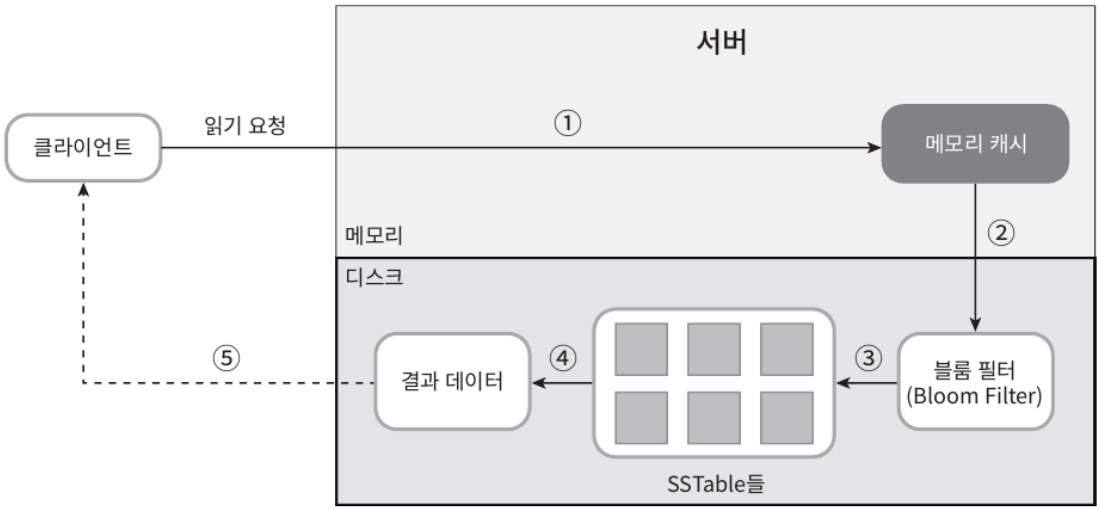
- 클라이어트는 get, put API 와 소통
- 중재자는 클라이언트에게 프락시 역할
- 노드는 해시링 위에 분포
- 노드 자동 추가 삭제 가능하도록 시스템과 분산
- 데이터는 여러 노드에 다중화
- 모든 노드가 같은 책임이므로 SPOF 방지

- 쓰기 경로



- 쓰기 요청이 커밋 로그에 기록
- 데이터가 메모리 캐시에 기록
- 메모리 케시 가득차거나 임계치 도달시 데이터는 디스크의 SSTable에 저장.

- 읽기 경로



- 데이터가 메모리에 있는지 검사.
- 없다면 블룸 필터 검사
- 블룸 필터 통해 SSTable에 보관된 키를 확인
- SSTable 에서 데이터 가져와서 반환