6장. 키-값 저장소 설계

제이(jay-so)

목차

목차

1. 문제 이해 및 설계 범위 확정

기본 전제

- 완벽한 설계란 존재하지 않음
- 읽기, 쓰기, 메모리 사용량 사이에 균형을 찾고, 데이터의 일관성과 가용성 사이에서 타협적인 결정을 해야 함

설계 예시

- 키-값 쌍의 크기는 10KB 이하
- 큰 데이터를 저장할 수 있어야 함
- 높은 규모 확장성, 가용성을 제공해야 함
- 데이터 일관성 수준은 조정이 가능해야 함
- 응답 지연시간이 짧아야 함

2. 단일 서버 키-값 저장소

- 하나의서버만 키- 값 저장소를 설계하는 것은 쉬움
- 키-값 쌍 전부를 메모리에 해시 테이블로 저장함

단점

- 모든 데이터를 메모리 안에 두는 것은 불가능할 수 있음

개선책

- 데이터 압축
- 자주 사용되는 데이터만, 메모리에 두고 나머지는 디스크에 저장

-> 보다 많은 데이터를 저장하려면 분산 키-값 저장소를 만들어 사용해야 함

3. 분산 키-값 저장소

- 분산 키-값 저장소 = 분산 해시 테이블이라고 불리기도 함
- 분산 시스템을 설계에서는 CAP정리를 이해하고 있어야 함

CAP 정리

- 데이터 일관성, 가용성, 파티션 감내라는 3가지 요구사항을 동시에 만족하는 분산 시스템을 설계하는 것은 불가능하다는 정리

- 파티션 = 두 노드 사이에 통신장애가 발생했음을 의미함
- 파티션 감내 = 네트워크에 파티션이 생기더라도 시스템은 계속 동작해야 한다는 것을 의미함

3. 분산 키-값 저장소

CAP 정리

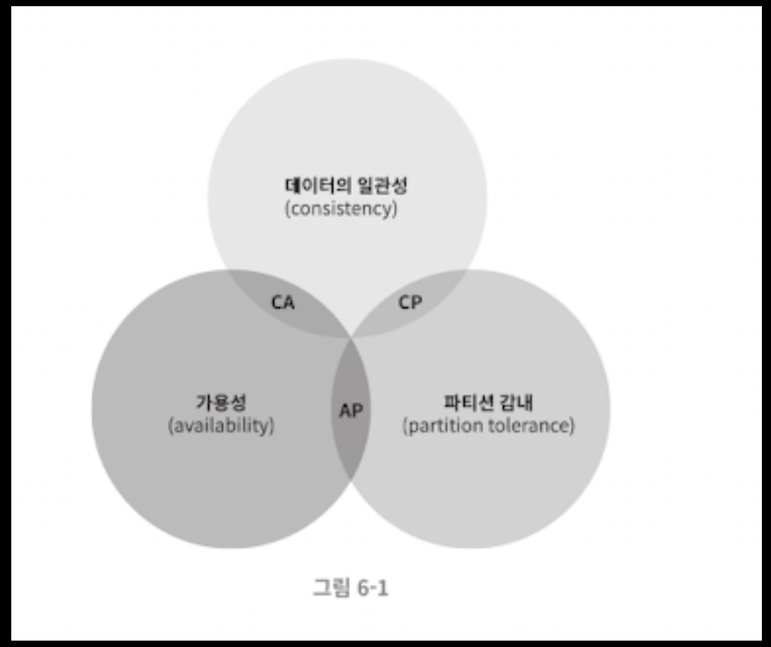


그림 6-1

CP 시스템: 일관성과 파티션 감내를 지원하는 키-값 저장소

- 가용성을 희생함

AP 시스템: 가용성과 파티션 감내를 지원하는 키-값 저장소

- 일관성을 희생함

CA 시스템: 일관성과 가용성을 지원하는 키-값 저장소

- 파티션 감내을 희생함
- 실세계에는 파티션 문제는 무조건 감내할 수 있도록 설계되어야 함

3. 분산 키-값 저장소

- 이상적 상태

네트워크가 파티션이 되는 현상은 일어나지 않음

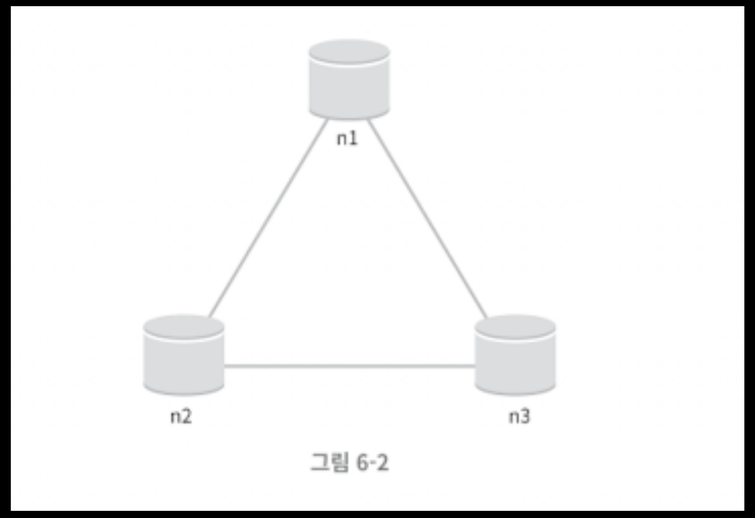


그림 6-2

- 실세계의 분산 시스템

분산 시스템은 파티션 문제를 피할 수 없음
파티션 문제가 발생하면 일관성과 가용성 사이에서 하나를 선택해야 함

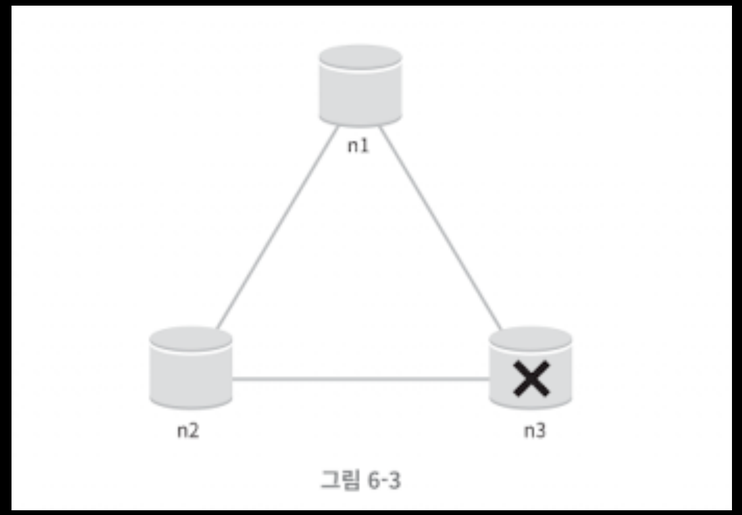


그림 6-3

3. 분산 키-값 저장소

- 가용성 대신 일관성을 선택한 경우(CP 시스템)

- 서버에 생길 수 있는 데이터 불일치를 위해 n1,n2에 대한 쓰기 연산을 중단함

- 가용성이 깨지게 됨

- 일관성 대신 가용성을 선택한 경우(AP 시스템)

- 일관성 대신 가용성을 선택한 시스템은, 낡은 데이터를 반환할 위험이 있더라도 계속 읽기 연산을 허용해야 함
- n1과 n2에서는 계속 읽기를 허용함

-> 파티션 문제가 해결된 뒤로 새로운 데이터를 n3에 전송함

3. 분산 키-값 저장소

데이터 파티션

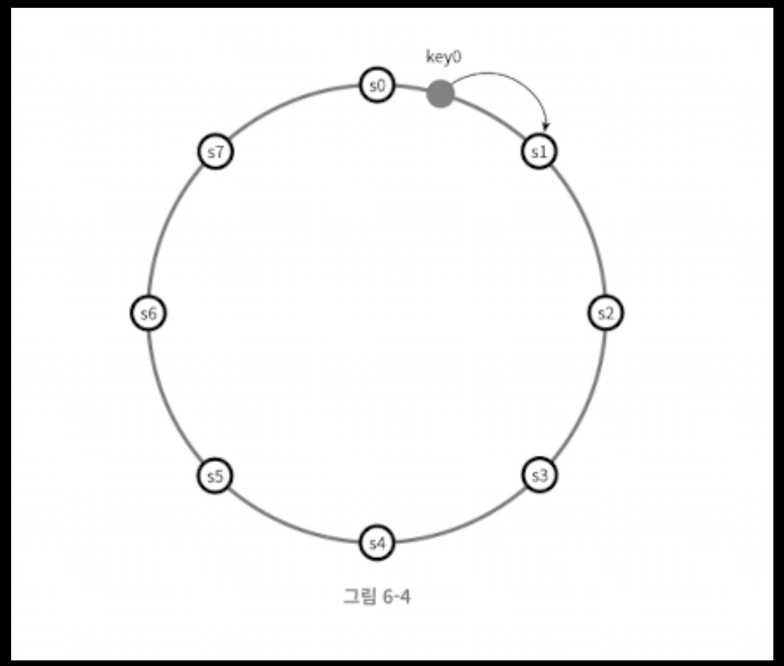


그림 6-4

대규모 어플리케이션의 경우, 전체 데이터를 한 대 서버에 욱여 넣는 것은 불가능함

해결책

- 데이터를 작은 파티션들로 분할한 다음, 여러 대의 서버에 저장함

주의 사항

1. 데이터를 여러 서버에 고르게 분산할 수 있는가?
2. 노드가 추가되거나, 삭제될 때 이동을 최소화 할 수 있는가?

- 안정해시는 해당 문제를 푸는데 적합한 기술임

3. 분산 키-값 저장소

데이터 다중성

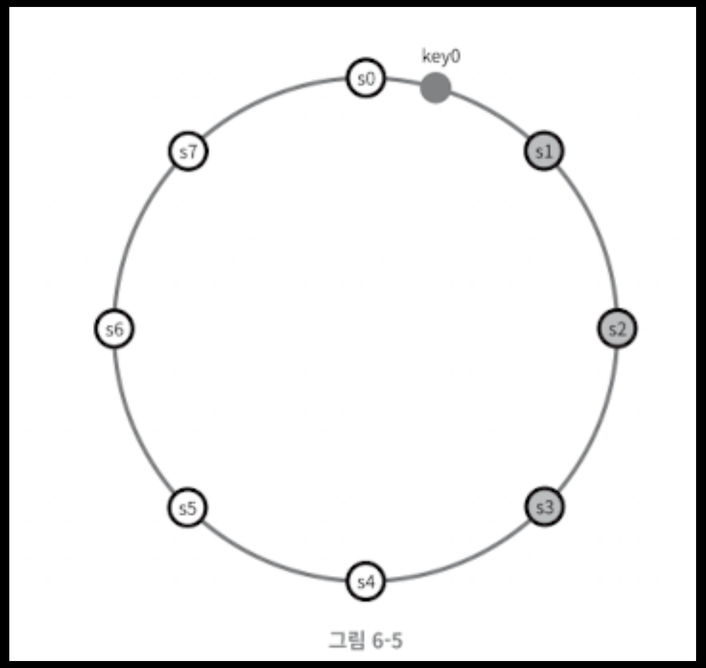


그림 6-5

높은 가용성과 안정성을 확보하기 위해 데이터를 N개의 서버에 비동기적으로 다중화 할 필요가 있음

- 데이터를 작은 파티션들로 분할한 다음, 여러 대의 서버에 저장함

어떤 키를 해시 링 위에 배치한 후, 해당 지점으로부터 시계 방향으로 링을 순회하면서 만나는 첫 N개의 서버의 데이터 사본을 보관함

3. 분산 키-값 저장소

데이터 일관성

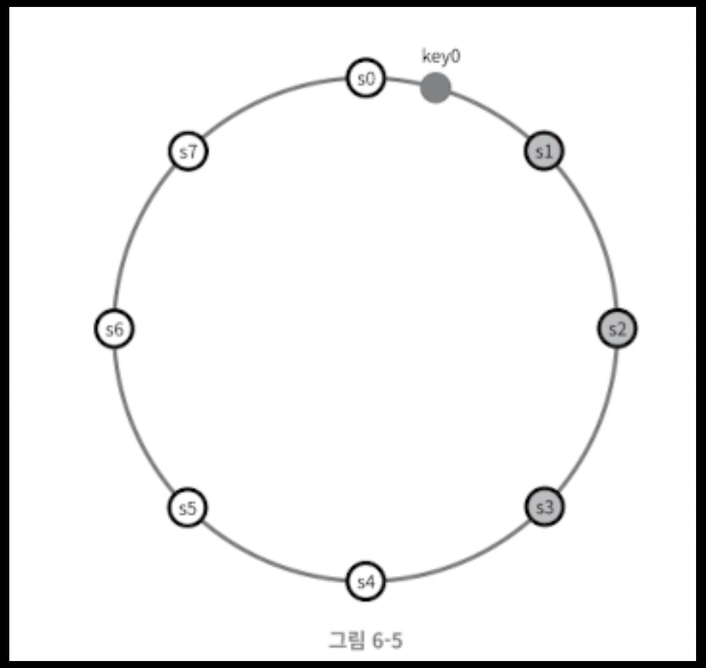


그림 6-5

여러 노드에 다중화된 데이터는 절적히 동기화되어야 함

- 정족수 합의 프로토콜을 사용하면, 읽기/쓰기 연산에 모두 일관성을 보장할 수 있음

정족수 합의 프로토콜

- 프로토콜의 핵심 아이디어는 모든 읽기 및 쓰기 연산이 정해진 수의 노드들 간에 합의를 이루어져야 한다는 것임

정족수= 시스템 내에서 특정 연산(읽기 또는 쓰기)를 수행하기 위해 동의해야 하는 최소한의 노드 수를 의미함

3. 분산 키-값 저장소

장애 처리

대다수의 대규모 시스템에서 장애는 아주 흔하게 벌어짐

- 따라서 장애를 어떻게 처리할 것인가 하는건 매우 중요한 문제임

장애 감지

분산 시스템에서 한 대의 서버가 죽었다고 바로 해당 서버를 장애처리를 하지 않음

- 보통 2대 이상의 서버가 똑같이 장애를 보고해야 해당 서버가 실제로 장애가 발생되었다고 간주함

3. 분산 키-값 저장소

장애 감지 - 멀티 캐스팅

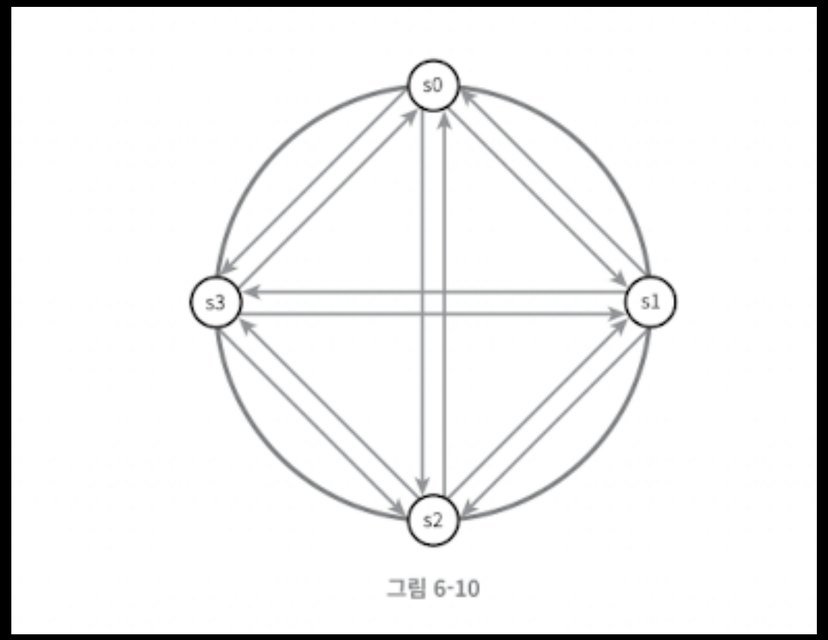


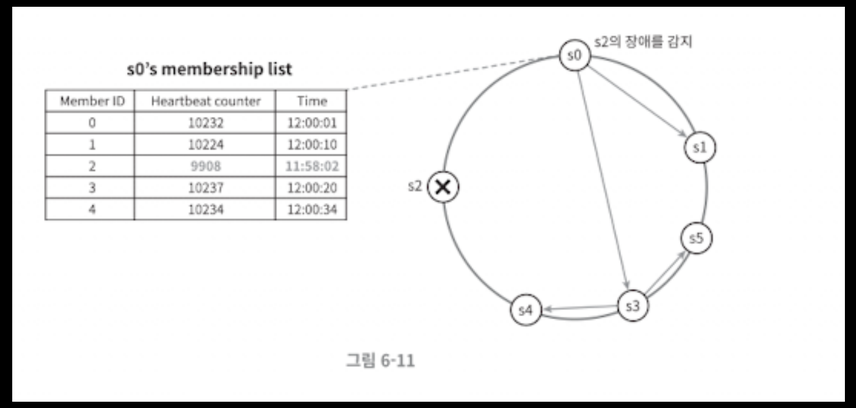
그림 6-10

모든 노드 사이에 멀티 캐스팅 채널을 구축함

- 서버 장애를 감지하는데 가장 쉬운 방법이지만, 해당 방식은 서버가 많을 때 비효율적임

3. 분산 키-값 저장소

장애 감지 - 가십 프로토콜 같은 분산형 장애 감지 솔루션을 선택함



동작원리

1. 각 노드는 멤버십 목록을 유지함
- 멤버십 목록 = 각 멤버 ID와 해당 박동 카운터 쌍의 목록

2. 각 노드는 주기적으로 자신의 박동 카운터를 증가시킴

3. 각 노드는 무작위로 선정된 노드들에게 주기적으로 자기 박동 카운터 목록을 보냄

4. 박동 카운터 목록을 받은 노드는 멤버쉽 목록을 최신값으로 갱신함

5. 어떤 멤버의 박동 카운터 값이 지정된 시간동안 갱신되지 않으면, 해당 멤버는 장애 상태인 것으로 간주함

3. 분산 키-값 저장소

장애 처리 - 일시적 장애처리

가십 프로토콜로 장애를 감지한 시스템은 가용성을 보장하기 위해, 필요한 조치를 취해야 함

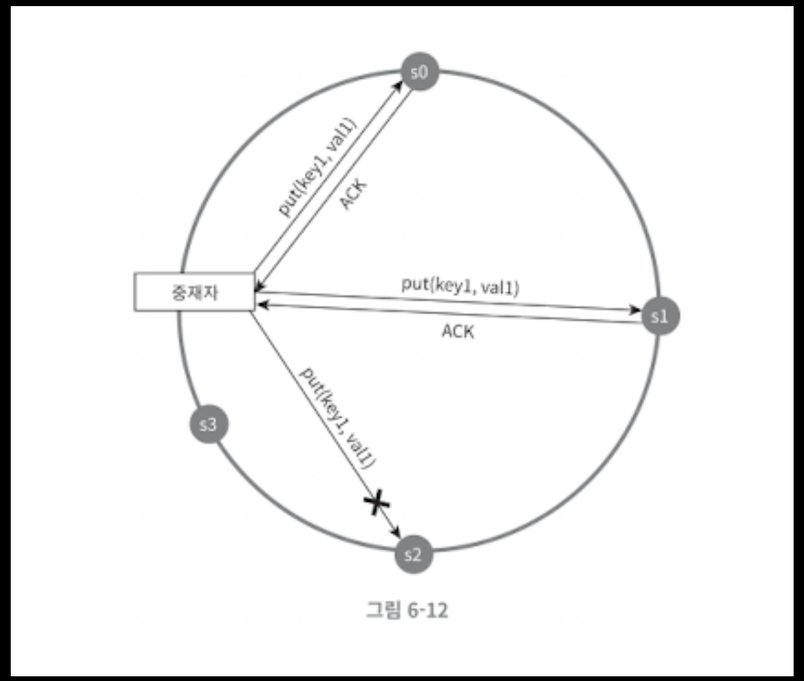


그림 6-12

느슨한 정족수 접근법을 사용함

조건을 완화하여 가용성을 높임

정족수 요구사항을 강제하는 대신, 쓰기 연산을 수행할 W개의 건강한 서버와 읽기 연산을 수행할 R개의 서버를 해시링에서 고르며, 장애인 서버는 무시함

- 장애 상태인 서버로 가는 요청은 다른 서버가 잠시 맡아서 처리함
- 해당 장애 서버가 복구되었을 때, 일괄 반영하여 데이터 일관성을 보존함

3. 분산 키-값 저장소

장애 처리 - 영구적 장애처리

반 - 엔트로피 프로토콜을 구현하여 사본들을 동기화함

반 - 엔트로피 프로토콜은 사본들을 비교하여 최신 버전으로 갱신하는 과정을 의미함

사본 간 일관성이 망가진 상태를 탐지하고, 전송 데이터의 양을 줄이기 위해 머클 트리를 사용함

머클 트리 정의

각 노드에서 자식 노드들에 보관된 값의 해시 또는 자식 노드들의 레이블로부터 계산된 값을 레이블로 붙여두는 트리

해시 트리를 사용하면서 대규모 자료구조의 내용을 효과적으로 두며, 보안상 안정한 방법으로 검증할 수 있음

3. 분산 키-값 저장소

키 공간 1~12까지 일때, 머클 트리를 만드는 예시

1단계(키 공간을 버킷으로 놔누어둠)

- 해당 예시는 4개의 버킷으로 나눠두었음

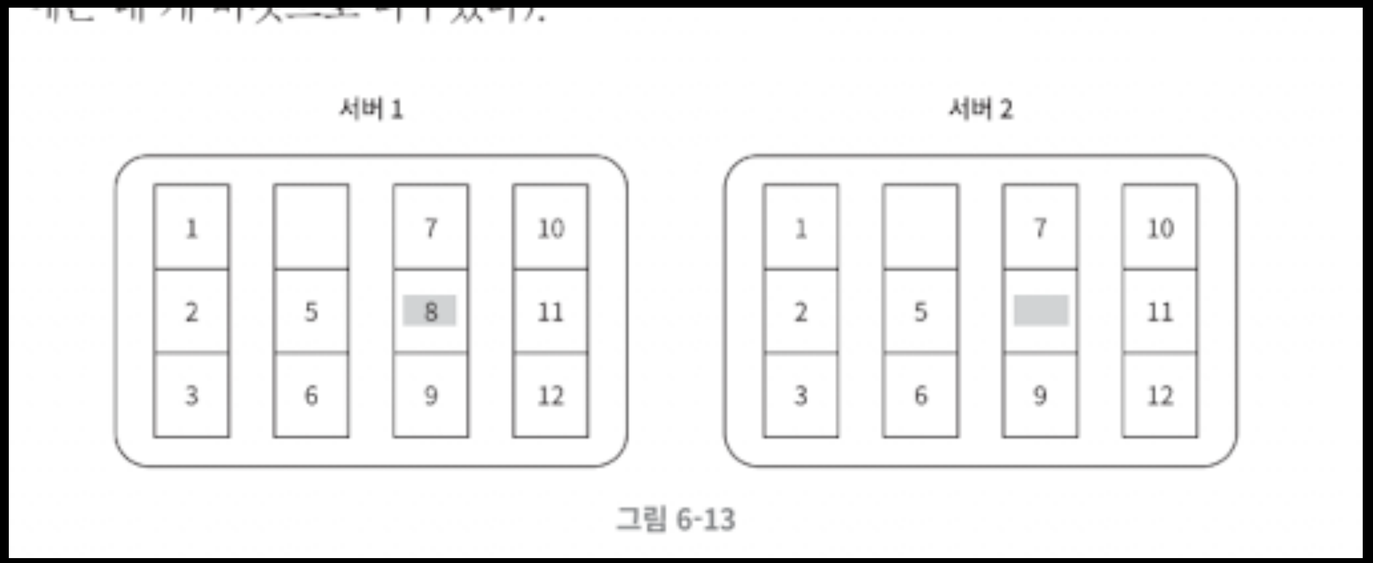


그림 6-13

3. 분산 키-값 저장소

키 공간 1~12까지 일때, 머클 트리를 만드는 예시

2단계

- 버킷에 포함된 각각의 키에 균등 분포 해시 함수를 적용하여 해시 값을 계산함

그림 6-14

3. 분산 키-값 저장소

키 공간 1~12까지 일때, 머클 트리를 만드는 예시

3단계

- 버킷별로 해시 값을 계산한 후, 해당 해시 값을 레이블로 갖는 노드를 만듦

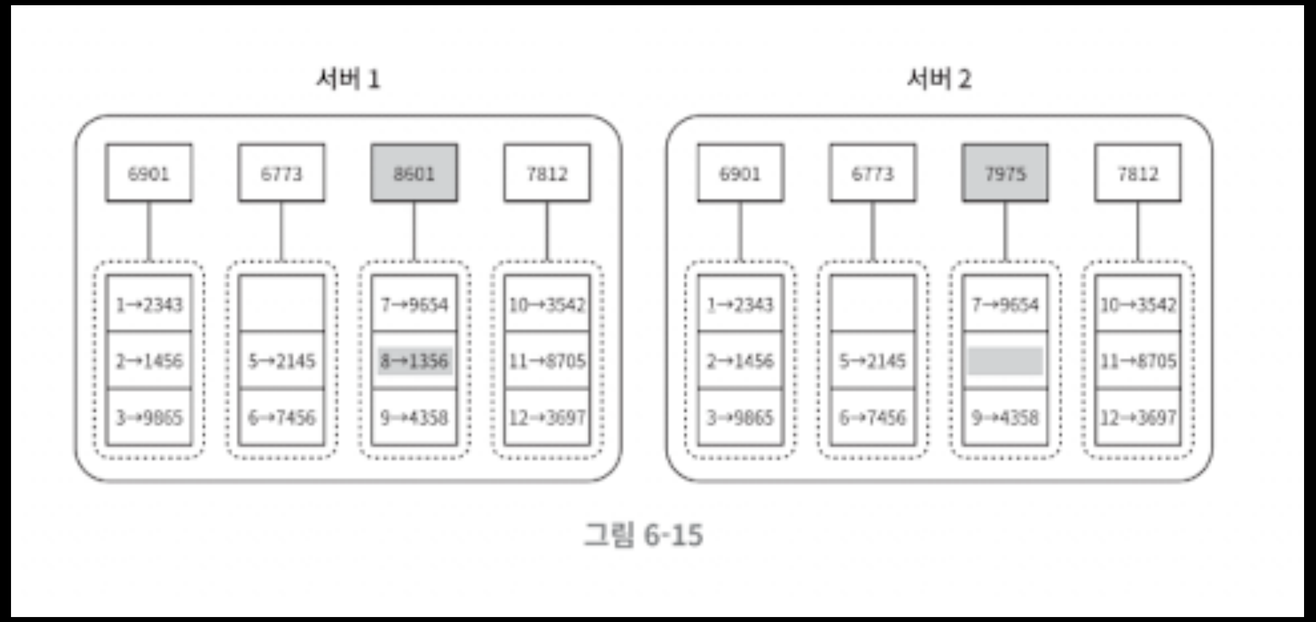


그림 6-15

3. 분산 키-값 저장소

키 공간 1~12까지 일때, 머클 트리를 만드는 예시

4단계

- 자식 노드의 레이블로부터, 새로운 해시 값을 계산하여 이진 트리를 상향식으로 구성해 나감
- 루트 노드의 해시 값이 일치하다면, 두 서버는 같은 데이터를 가짐
- 다를 경우 - 왼쪽 자식 노드의 해시값을 비교하고, 오른쪽 자식 노드의 해시 값을 비교함

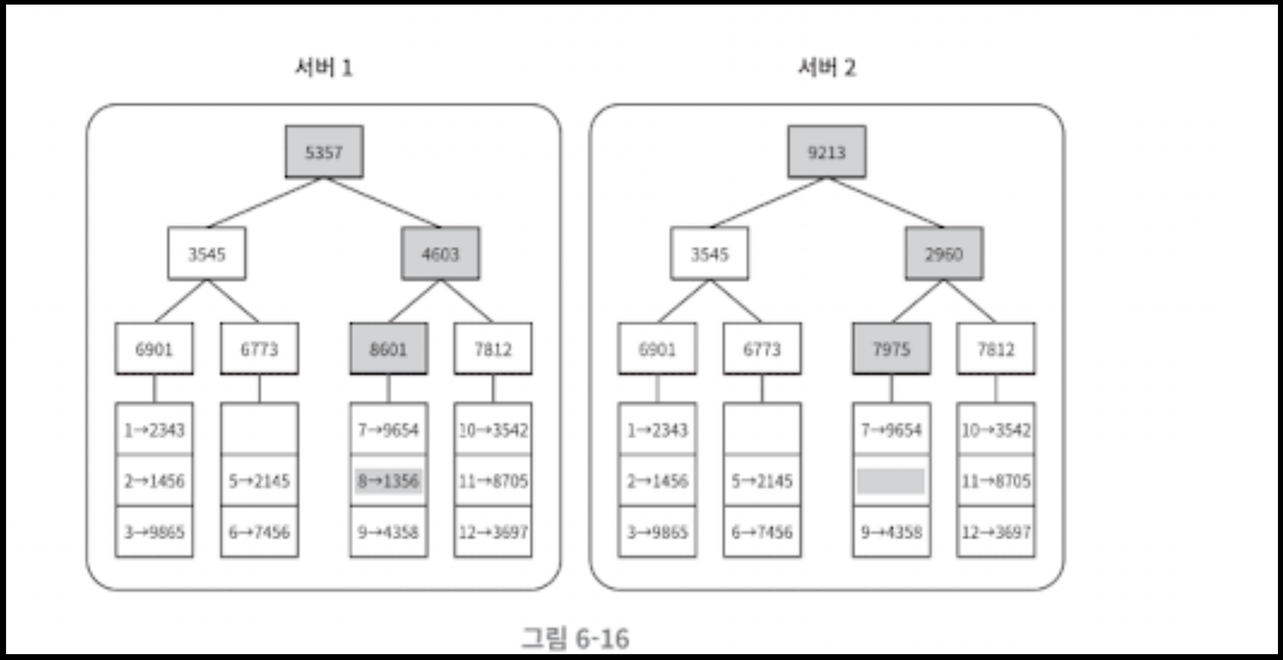


그림 6-16

3. 분산 키-값 저장소

단점

머클 트리를 사용하면, 동기화해야 하는 데이터의 양은 실제로 존재하는 차이의 크기에 비례할 뿐,
두 서버에 보관된 데이터의 총량과는 무관해짐

- 실제로 사용하는 시스템의 경우, 버킷 하나의 크기가 꽤 크게 됨

3. 분산 키-값 저장소

시스템 아키텍처 다이어그램

1. 클라이언트는 키-값 저장소가 제공하는 2가지 단순한 API, 즉 get(key)와 put(key,Value)와 통신함
2. 중재자는 클라이언트에게 키-값 저장소에 대한 프록시 역할을 하는 노드임
3. 노드는 안정 해시의 해시 링 위에 분포됨

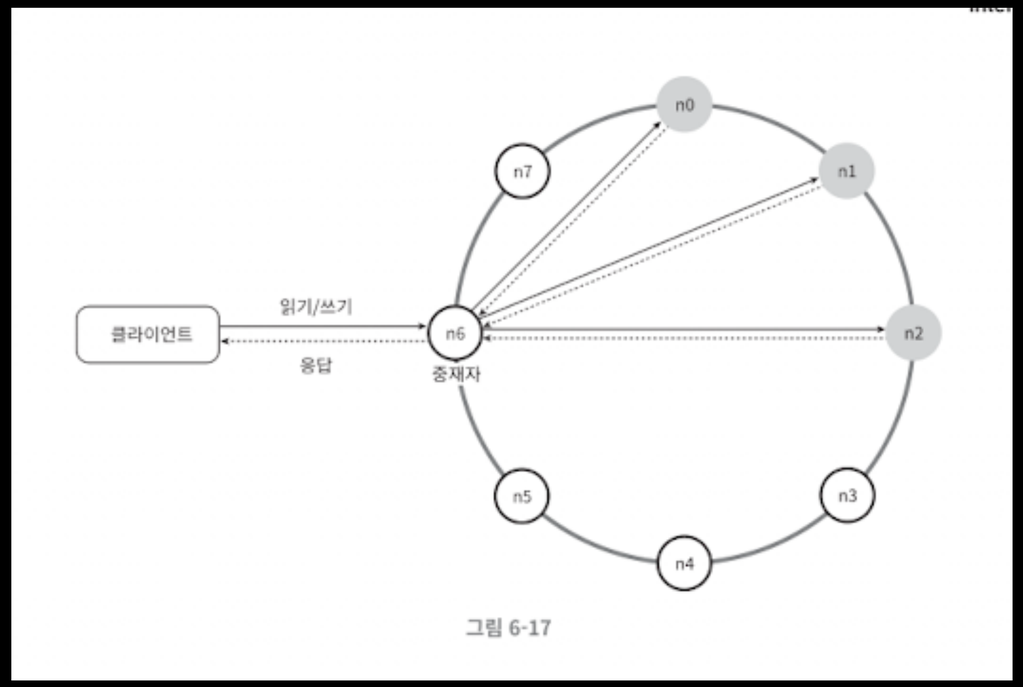


그림 6-17

3. 분산 키-값 저장소

시스템 아키텍처 다이어그램

4. 노드를 자동으로 추가 또는 삭제할 수 있도록, 시스템은 완전히 분산됨
5. 데이터는 여러 노드에 다중화됨
6. 모든 노드가 같은 책임을 지므로, SPOF가 없음

- 완전히 분산된 설계를 채택하였으므로. 모든 노드는 다음과 같은 제시된 기능 전부를 전부 지원해야 함

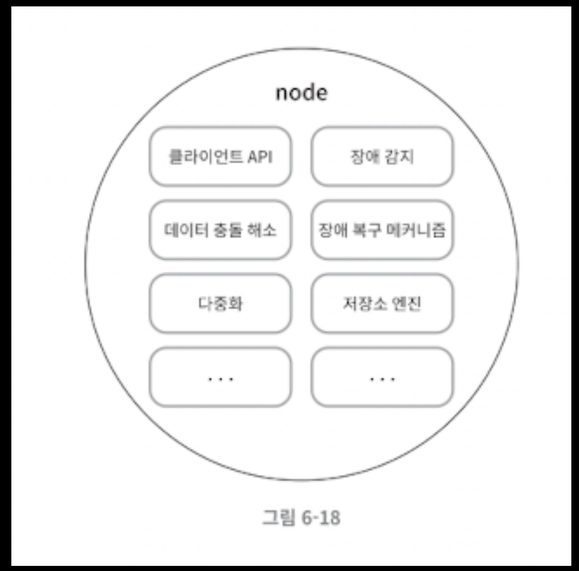


그림 6-18

3. 분산 키-값 저장소

쓰기 경로

- 카산드라의 사례를 참고한 쓰기 경로

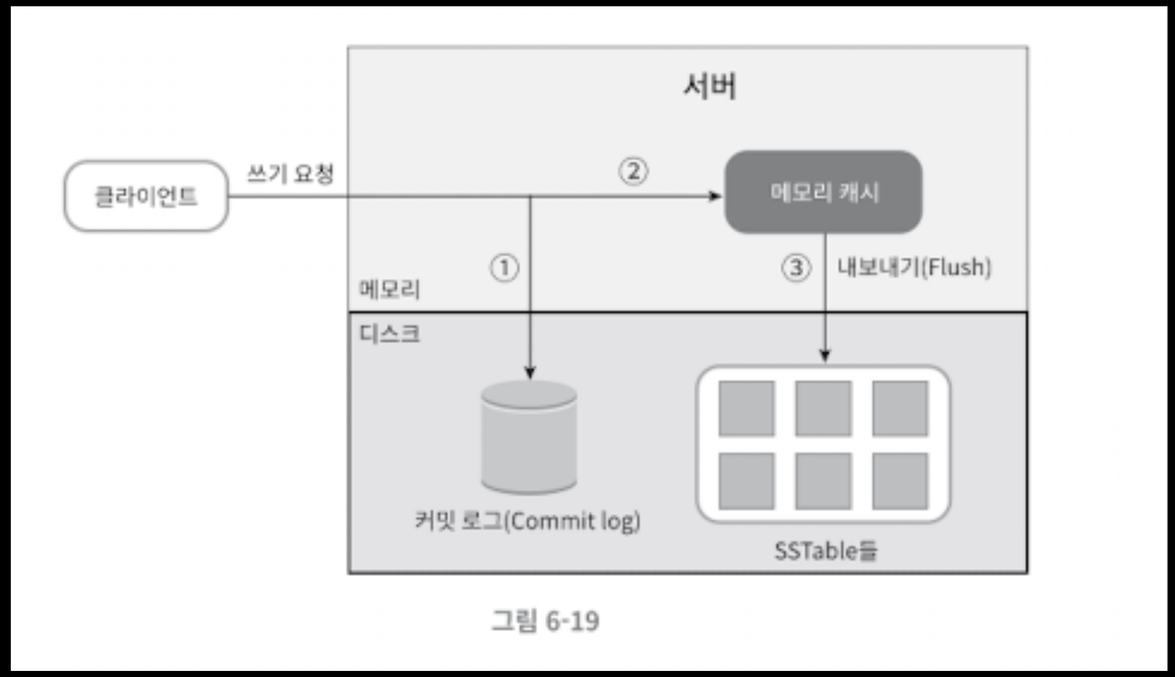


그림 6-19

1. 쓰기 요청이 커밋 로그(commit log) 파일에 기록됨
2. 데이터가 메모리 캐시에 기록됨
3. 메모리 캐시가 가득 차거나, 사전에 정의된 임계치에 도달하면 데이터는 디스크에 있는 SSTable에 기록됨

- SSTable은 Sorted-String Table의 약어로 〈키,값〉의 순서쌍을
- 정렬된 리스트 형태로 관리하는 테이블

3. 분산 키-값 저장소

읽기 경로

- 읽기 요청을 받은 노드는 데이터가 메모리 캐시에 있는지부터 살핌

데이터가 메모리에 있는 경우(읽기 경로가 있는 경우)

- 읽기 경로가 있는 경우, 해당 그림처럼 데이터를 클라이언트에게 반환함

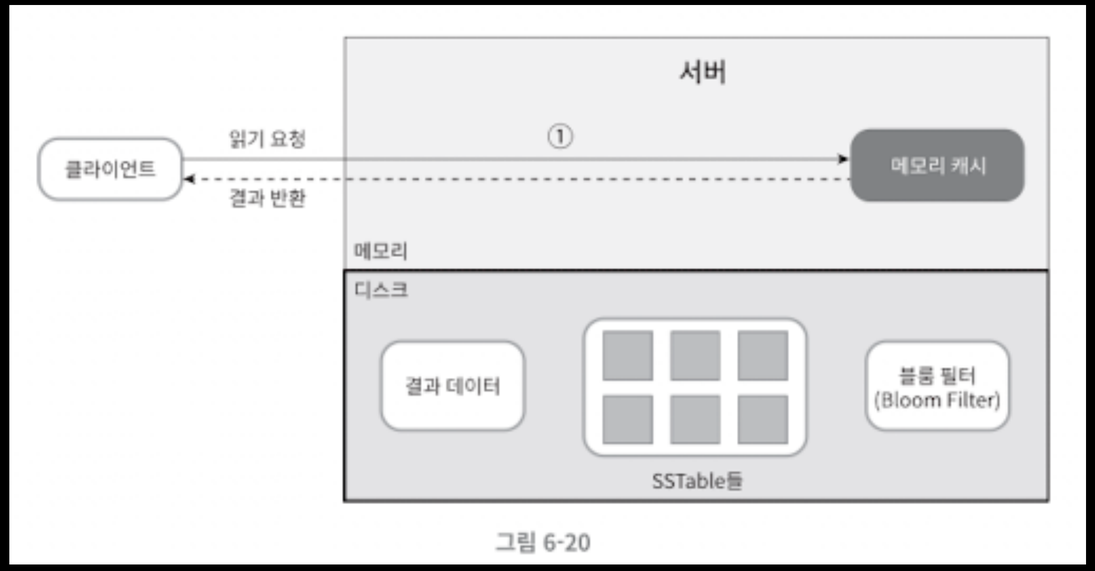


그림 6-20

3. 분산 키-값 저장소

읽기 경로

데이터가 메모리에 없는 경우(읽기 경로가 없는 경우)

- 디스크에서 가져와야 함
- 어느 SSTable에 있는지 알아낼 효율적인 방법이 필요함

-> 해당 문제를 푸는데 블룸 필터가 흔히 사용됨

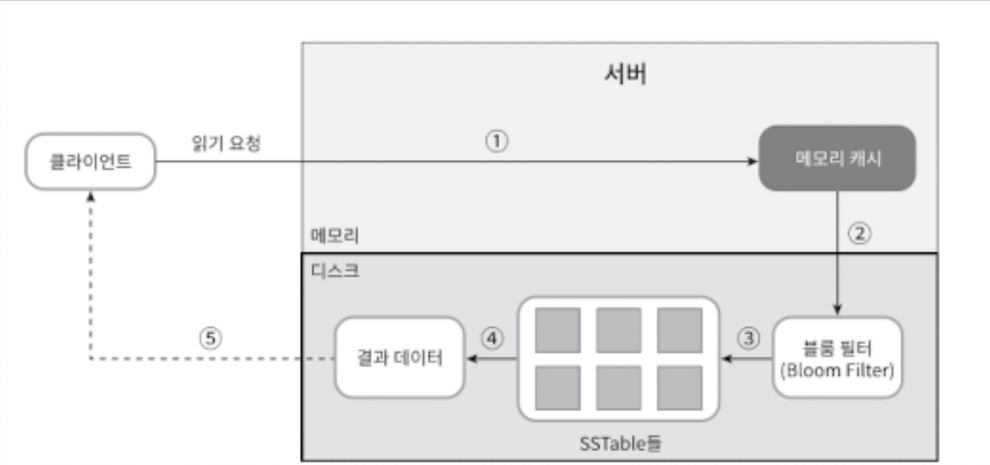


그림 6-21

- +) 블룸 필터
- M인 비트 배열과 k개의 독립적인 해시 함수로 구성된 확률적 데이터 구조
- "어떤 값이 집합에 속해 있는가?"를 검사하는 필터 및 이를 구성하는 자료형"

4. 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(heterogeneity)	안정 해시
조절 가능한 데이터 일관성	정족수 합의(quorum consensus)
일시적 장애 처리	느슨한 정족수 프로토콜(sloppy quorum)과 단서 후 임시 위탁(hinted handoff)
영구적 장애 처리	머클 트리(Merkle tree)
데이터 센터 장애 대응	여러 데이터 센터에 걸친 데이터 다중화

표 6-2

5. 참고자료