키 값 저장소 설계

¿ε´◠`⨝ვა 너무 어려운 거시애오,,

키 값 저장소 설계

1. 키 값 저장소란?

2. 키 값 저장소 설계하기

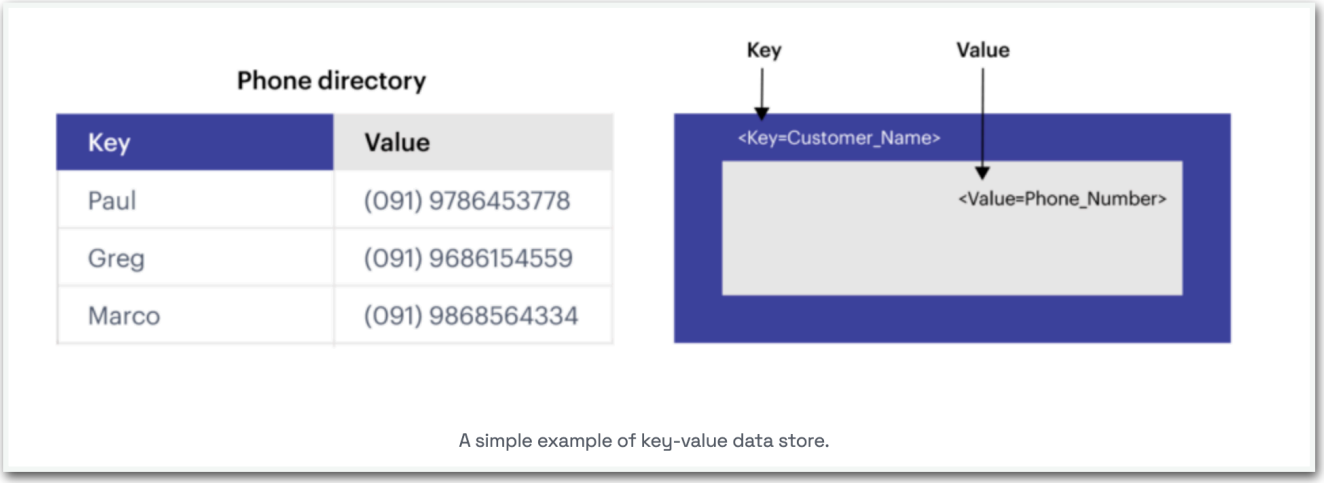
1. 문제 이해 및 설계 범위 확정
2. 단일 키 값 저장소
3. 분산 키 값 저장소
   - (부록) CAP 정리
   - 시스템 컴포넌트
   - 시스템 아키텍처 다이어그램

키 값 저장소란?

✔️키-값 저장소(key value stores)는 키-값 데이터베이스(Key value databases) 라고도 불리는 비 관계형 데이터베이스
✔️값은 언제나 고유 식별자(identifier)를 key로 가져야 함
✔️키와 값 사이의 이런 연결 관계를 키-값 쌍(pair) 라고 함
✔️ NoSQL 데이터베이스의 한 종류

키
- 키는 유일해야 함
- 해당 키의 값은 언제나 키를 통해서만 접근 가능
- 일반 텍스트 / 해시 값
- 키는 짧을수록 좋음

값
- 문자열 / 리스트 / 객체 등 자유로움



A simple example of key-value data store.

https://redis.io/nosql/key-value-databases/
https://www.influxdata.com/key-value-database/
https://www.mongodb.com/resources/basics/databases/key-value-database

키 값 저장소 설계하기

문제 이해 및 설계 범위 확정

📌 put(key, value), get(key) 을 지원
📌 키 값 쌍의 크기는 10kb 이하
📌 큰 데이터를 저장할 수 있어야 함
📌 높은 가용성 제공 -> 장애가 있더라도 얼른 응답
📌 높은 규모 확장성 -> 트래픽 양에 따라 서버 증설/삭제 필요
📌 데이터 일관성 수준은 조정 가능
📌 응답 지연시간(latency)이 짧아야 함

키 값 저장소 설계하기

단일 서버 키 값 저장소

키-쌍 전부를 메모리에 해시 테이블로 저장
- 빠른 속도를 보장
- 단일 서버의 메모리는 물리적으로 제한되어 있으므로, 저장할 수 있는 데이터의 양에 한계가 존재

데이터 압축
- snappy, lz4 같은 압축 알고리즘을 사용하여 저장되는 값을 압축
- 장점
  - 압축된 데이터를 저장하면 메모리를 절약할 수 있으며, 디스크 I/O 성능도 향상시킬 수 있습니다.
- 단점
  - 데이터 압축은 메모리 사용을 최적화할 수 있지만, 압축과 해제 과정에서 CPU 자원이 소모됩니다.
  - 특히, 압축률이 높은 알고리즘은 CPU 사용량이 높아져 전체 시스템 성능에 영향을 줄 수 있습니다.

자주 쓰이는 데이터만 메모리에 두고 나머지는 디스크에 저장
- 단일 서버 환경에서는 데이터의 영속성을 보장하기 위해 주기적으로 데이터를 디스크에 저장하는 스냅샷(snapshot) 기능을 구현
- 데이터 변경사항을 로그로 기록하여 장애 발생 시 로그를 재생하여 데이터를 복구할 수 있습니다.

단일 서버로만 키-값 저장소를 설계하는 경우, 확장성이나 재해복구 측면에서 부족한 점이 많음

키 값 저장소 설계하기

분산 서버 키 값 저장소

* 데이터 파티션
* 데이터 다중화
* 일관성과 일관성 불일치 해소
* 장애처리
* 시스템 아키텍처 다이어그램

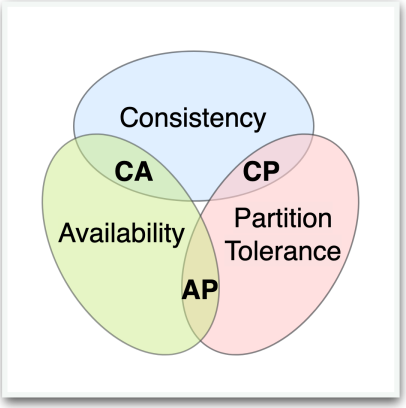
잠깐!

~~키 값 저장소 설계하기~~ 분산 시스템 설계를 위한 CAP 정리

~~분산 서버 키 값 저장소~~

CAP theorem 또는 Brewer's theorem
분산 시스템은 **일관성*(Consistency)***, **가용성*(Availability*** 및 **파티션 감내*(Partition tolerance)***(CAP의 '**C**', '**A**' 및 '**P**')의 세 가지 원하는 특성 중 두 가지만 제공할 수 있습니다.

Consistency: 모든 노드가 같은 순간에 같은 데이터를 볼 수 있다.
일관성이란 어떤 노드에 연결되었는지와 무관하게 모든 클라이언트가 동시에 동일 데이터를 볼 수 있음을 의미합니다. 이러한 상황이 발생하려면, 데이터가 하나의 노드에 기록될 때마다 이 데이터는 쓰기가 '성공'으로 간주되기 전에 시스템의 다른 모든 노드로 즉시 전달되거나 복제되어야 합니다.



Availability: 모든 요청이 성공 또는 실패 결과를 반환할 수 있다.
가용성이란 하나 이상의 노드가 작동 중지된 경우에도 데이터를 요청하는 클라이언트가 응답을 받음을 의미합니다. 이를 다른 방법으로 설명해 보면, 분산 시스템의 모든 작업 중인 노드는 예외 없이 모든 요청에 대해 유효한 응답을 리턴합니다.

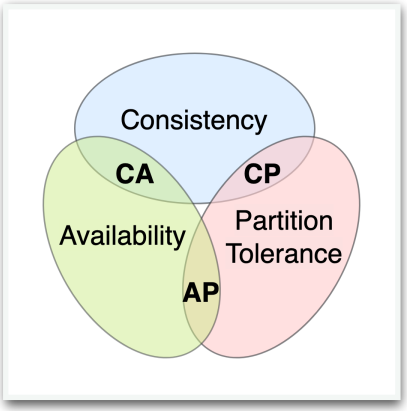
Partition tolerance: 메시지 전달이 실패하거나 시스템 일부가 망가져도 시스템이 계속 동작할 수 있다.
분할(Partition)이란 분산 시스템 내의 통신 단절, 즉 두 노드 간의 연결이 유실되거나 일시적으로 지연된 상태입니다. 분할 내성(Partition Tolerance)이란 시스템의 노드 간에 다수의 통신 단절에도 불구하고 클러스터가 계속해서 작동함을 의미합니다.

https://www.ibm.com/kr-ko/topics/cap-theorem

http://eincs.com/2013/07/misleading-and-truth-of-cap-theorem/. —> 요거는 생각해볼만한 여지가 있으니 꼭 읽어보기

~~키 값 저장소 설계하기~~ 분산 시스템 설계를 위한 CAP 정리

~~분산 서버 키 값 저장소~~



CP: 일관성-파티션감내, 가용성을 희생
AP: 가용성-파티션감내, 일관성을 희생
CA: 일관성과 가용성을 지원하는 키-값 저장소
파티션 감내는 지원하지 않는다. 그러나 네트워크 장애는 피할 수 없는 일로 여겨지므로 분산 시스템은 반드시 파티션 문제를 감내할 수 있도록 설계되어야 한다.
—> CA 시스템은 존재하지 않는다.

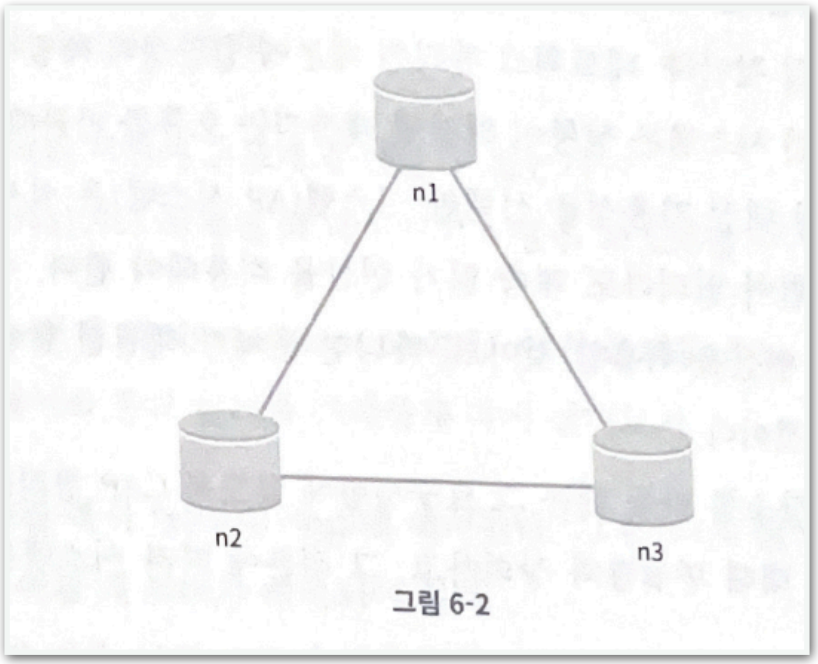
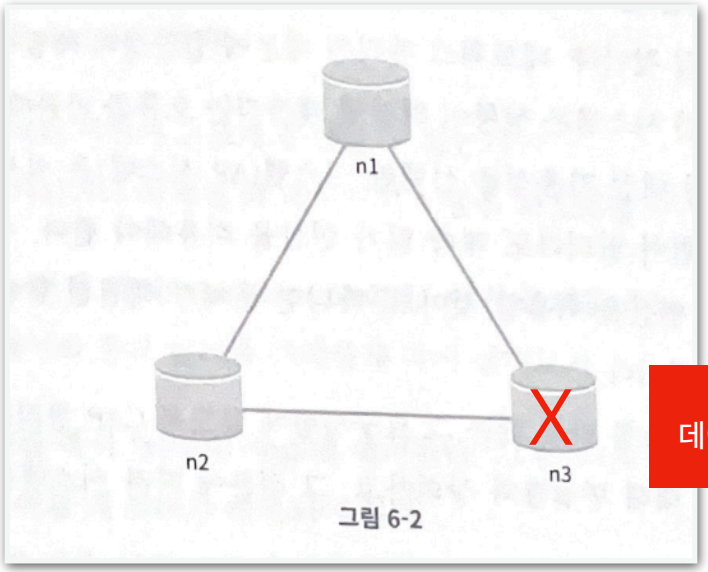


그림 6-2

~~키 값 저장소 설계하기~~ 분산 시스템 설계를 위한 CAP 정리

~~분산 서버 키 값 저장소~~



n3에 장애가 발생하여
데이터 불일치가 발생한 경우,

CP 시스템

- n1과 n2에 대해 쓰기 연산 중단 (가용성 희생)
- 읽기 연산 불가능, 오류 반환

Ex) 은행권 시스템 등 일관성이 중요한 경우

AP 시스템

- 낡은 데이터라도 읽기 연산 허용 (일관성 희생)
- n1, n2 의 쓰기 연산 허용

키 값 저장소 설계하기

분산 서버 키 값 저장소: 시스템 컴포넌트

데이터 파티션

대규모 애플리케이션의 경우 전체 데이터를 한대 서버에 넣을 수 없음

데이터 파티션이란,
데이터를 작은 파티션ㅇ들로 분할한 다음 여러대 서버에 저장하는 것

고려사항
- 데이터를 여러 서버에 고르게 분산할 수 있는가
- 노드가 추가되거나 삭데될 떄 데이터의 이동을 최소화할 수 있는가

대안
- 안정해시
  - 규모 확장 자동화: 시스템 부하에 따라 서버가 자동으로 추가되거나 삭제되도록 만들 수 있다.
  - 다양성: 각 서버의 용량에 맞게 다상 노드의 수를 조정할 수 있다. 고성능일수록 더 많은 가상 노드 사용

키 값 저장소 설계하기

분산 서버 키 값 저장소: 시스템 컴포넌트

데이터 다중화

높은 가용성과 안정성을 확보하기 위해
데이터를 N개 서버에 비동기적으로 다중화 (replication)
(N은 튜닝 가능한 값)

어떤 키를 해시 링 위에 배치한 후,
그 지점 으로부터 시계 방향으로 링을 순회하면서 만나는 첫 N개 서버에 데이터 사본 을 보관

데이터의 사본은 다른 데이터 센터에 보관하고, 각 센터들은 고속 네트워크로 연결

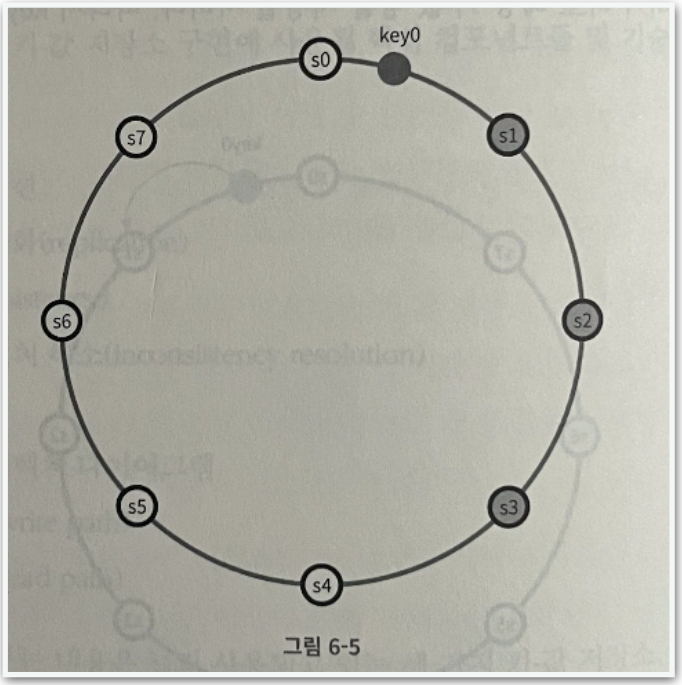


그림 6-5

키 값 저장소 설계하기

분산 서버 키 값 저장소: 시스템 컴포넌트

데이터 일관성

여러 노드에 다중화된 데이터는 적절히 동기화가 되어있어야 함 -> **정족수 합의 프로토콜**

정족수 ? 정족수는 여러 사람의 합의로 운영되는 의사기관에서 의결을 하는데 필요한 최소한의 참석자 수

N = 사본 개수
W = 쓰기 연산에 대한 정족수
R = 읽기 연산에 대한 정족수

요구되는 일관성 수준에 따라 W,R, N의 값을 조정
- 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

카산드라는 코디네이터라고 칭함
https://nosqldb.tistory.com/15?category=957763

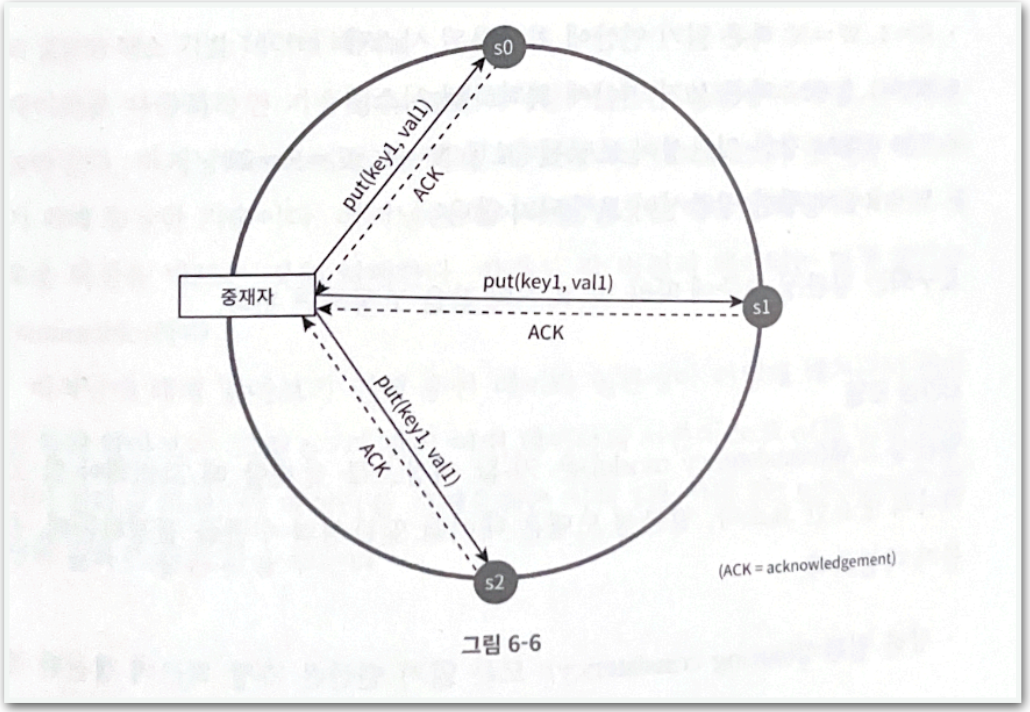


그림 6-6

키 값 저장소 설계하기

분산 서버 키 값 저장소: 시스템 컴포넌트

일관성 모델

강한 일관성

- 모든 읽기 연산은 최근에 갱신된 결과를 반환한다.
- 클라이언트는 절대로 낡은 데이터를 보지 못한다.
- W + R > N, 일관성을 보증할 최신 데이터를 가진 노드가 최소 하나는 겹치기 때문
- 모든 사본에 현재 쓰기 연산의 결과가 반영될 때 까지 해당 데이터에 대한 R/W를 금지. —> 고가용성 시스템 X

약한 일관성

- 읽기 연산은 가장 최근에 갱신된 결과를 반환하지 못할 수도 있다.

결과적 일관성

- 약한 일관성의 한 형태로, 갱신 결과가 결국에는 모든 사본에 반영(=동기화) 되는 모델
- 다이나모 또는 카산드라 등 사용하는 방식

키 값 저장소 설계하기

분산 서버 키 값 저장소: 시스템 컴포넌트

비 일관성 해소 기법: 데이터 버저닝

버저닝과 벡터시계
버저닝은 데이터를 변경할 때 마다 해당 데이터의 새로운 버전을 만드는 것을 의미
따라서 각 저번의 데이터는 변경 불가능

백터 시계는 데이터 불일치 문제를 푸는데 보편적으로 사용되는 기술

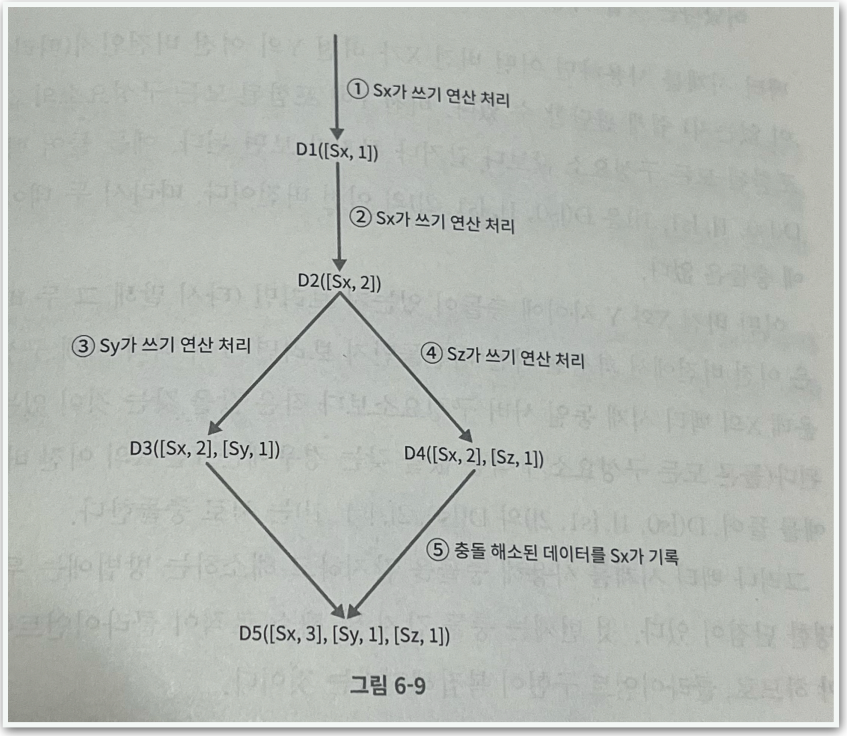


그림 6-9

키 값 저장소 설계하기

분산 서버 키 값 저장소: 시스템 컴포넌트

장애 처리

보통 2대 이상의 서버가 특정 서버의 장애를 보고할 때 장애 발생으로 간주

장애 감지 (failure detection)

- 가장 확실하고 쉬운 방법은 멀티캐스팅, but 비효율
- 가십 프로토콜 (gossip protocol)
  - 각 노드는 멤버십 목록(membershup list)를 유지한다. 멤버십 목록은 각 ID와 그 박동 카운터(heartbeat counter) 쌍의 목록
  - 각 노드는 주기적으로 자신의 박동 카운터를 증가시킨다.
  - 각 노드는 무작위로 선정된 노드들에게 주기적으로 자기 박동 카운터 목록을 보낸다.
  - 박동 카운터 목록을 받은 노드는 멤버십 목록을 최신 값으로 갱신한다.
  - 어떤 멤버의 박동 카운터 값이 지정된 시간 동안 갱신되지 않으면 해당 멤버는 장애(offline) 상태인 것으로 간주한다

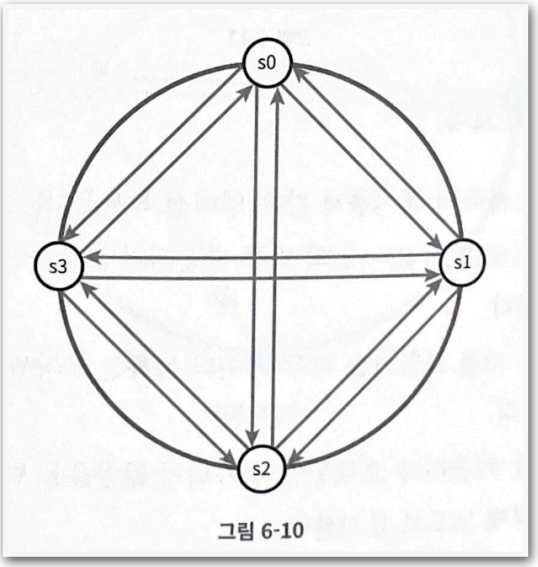


그림 6-10

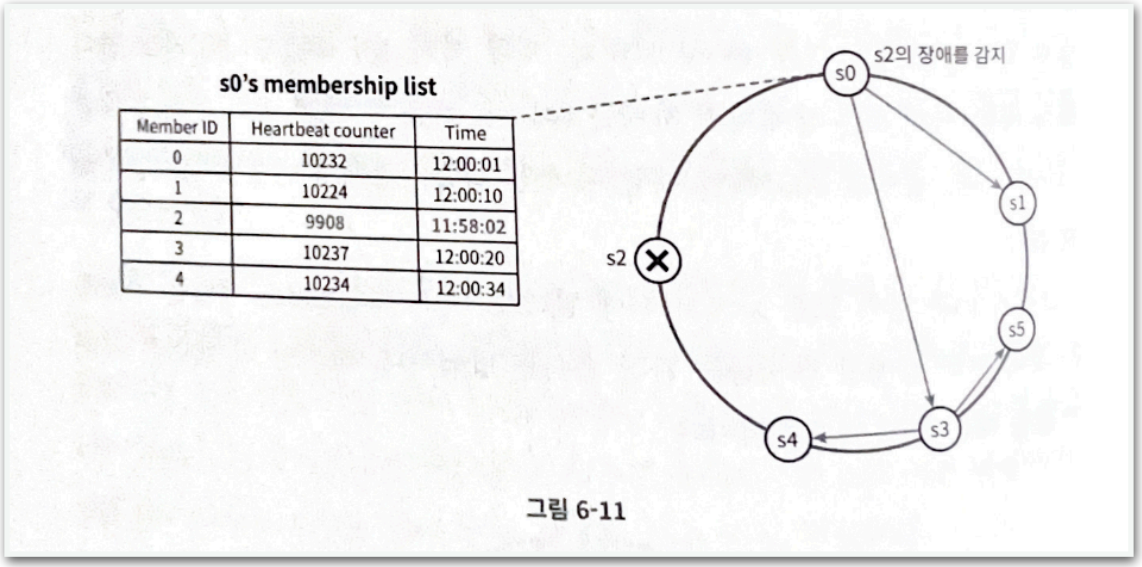


그림 6-11

키 값 저장소 설계하기

분산 서버 키 값 저장소: 시스템 컴포넌트

장애 처리

장애 해소 (failure resolution) - 일시적 장애처리
가십 프로토콜로 장애를 감지한 시스템

엄격한 정족수 접근법

- 강한 일관성에 따라 읽기와 쓰기 연산을 금지

느슨한 정족수 접근법

- 쓰기 연산산을 수행할 W 개의 서버와 읽기 연산을 수행할 R개의 서버를 고름
- 서버가 복구되었을 때 일괄 반영할 수 있게끔 선택한 쓰기 서버에 단서(hint) 를 남겨둠 —> 단서 후 임시 위탁 (hinted handoff)

키 값 저장소 설계하기

분산 서버 키 값 저장소: 시스템 컴포넌트

장애 처리

장애 해소 (failure resolution) - 영구 장애 처리

반-앤트로피 프로토콜을 이용 (anti-entropy) 하여 사본을 동기화
사본간 일관성이 망가진 상태를 탐지 / 전송데이터의 양을 줄이기 위해 —-> 머클 트리 (=해시트리)
사본들을 비교하여 최신 버전으로 갱신

1. 2개의 서버에서 각각 버킷단위로 키를 나눈다
2. 버킷에 포함된 각각에 대해 해시값을 계산한다
3. 해시값을 레이블로 갖는 노드를 만든다
4. 자식 노드의 레이블로부터 이진트리를 상향식으로 구성
5. 루트 노드의 해시값부터 비교…

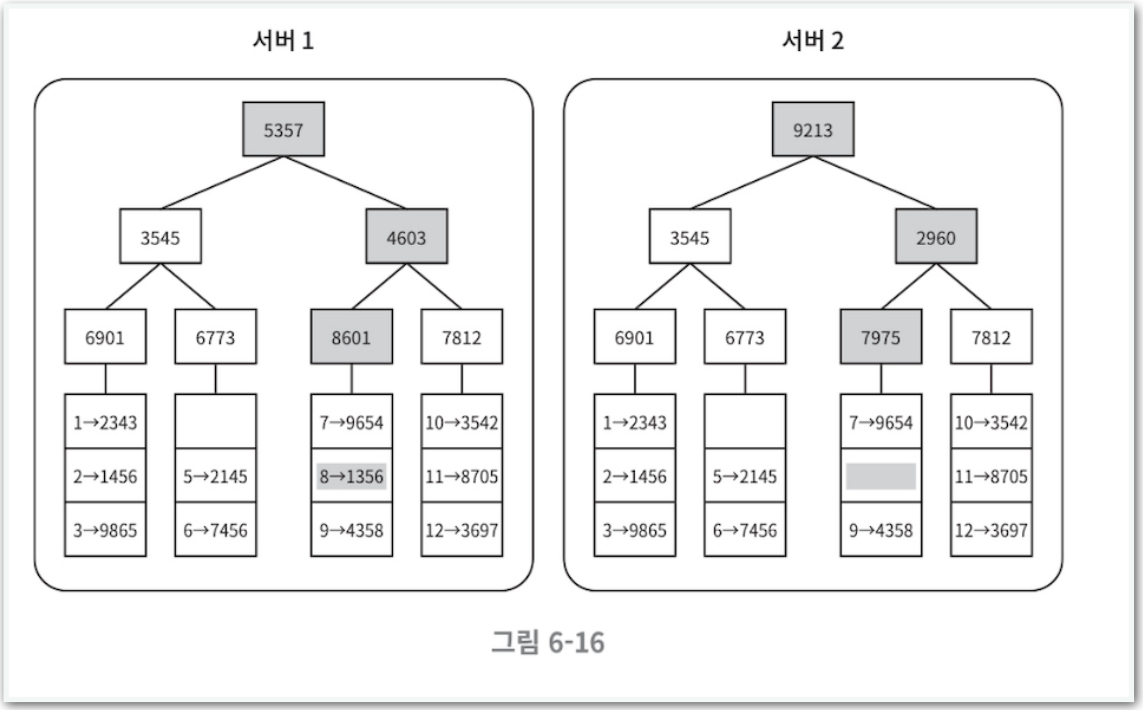


그림 6-16

데이터 센터 장애 처리
- 데이터 센터 다중화 필요

키 값 저장소 설계하기

분산 서버 키 값 저장소: 시스템 컴포넌트

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

키 값 저장소 설계하기

분산 서버 키 값 저장소: 시스템 아키텍처 다이어그램

- ✔ 클라이언트는 키-값 저장소가 제공하는 두 가지 단순한 API (get/put) 과 통신한다.
- ✔ 중재자는 클라이언트에게 키=값 저장소에 대한 proxy 역할을 하는 노드이다
- ✔ 노드는 안정해시의 해시링 위에 분포한다
- ✔ 노드를 자동으로 추가/삭제할 수 있도록 시스템은 완전히 분산된다.
- ✔ 데이터는 여러 노드에 다중화된다.
- ✔ 모든 노드가 같은 책임을 지기 때문에 SPOF는 존재하지 않는다.

SPOF (single point of failure)
시스템 구성 요소 중에서, 동작하지 않으면
전체 시스템이 중단되는 요소

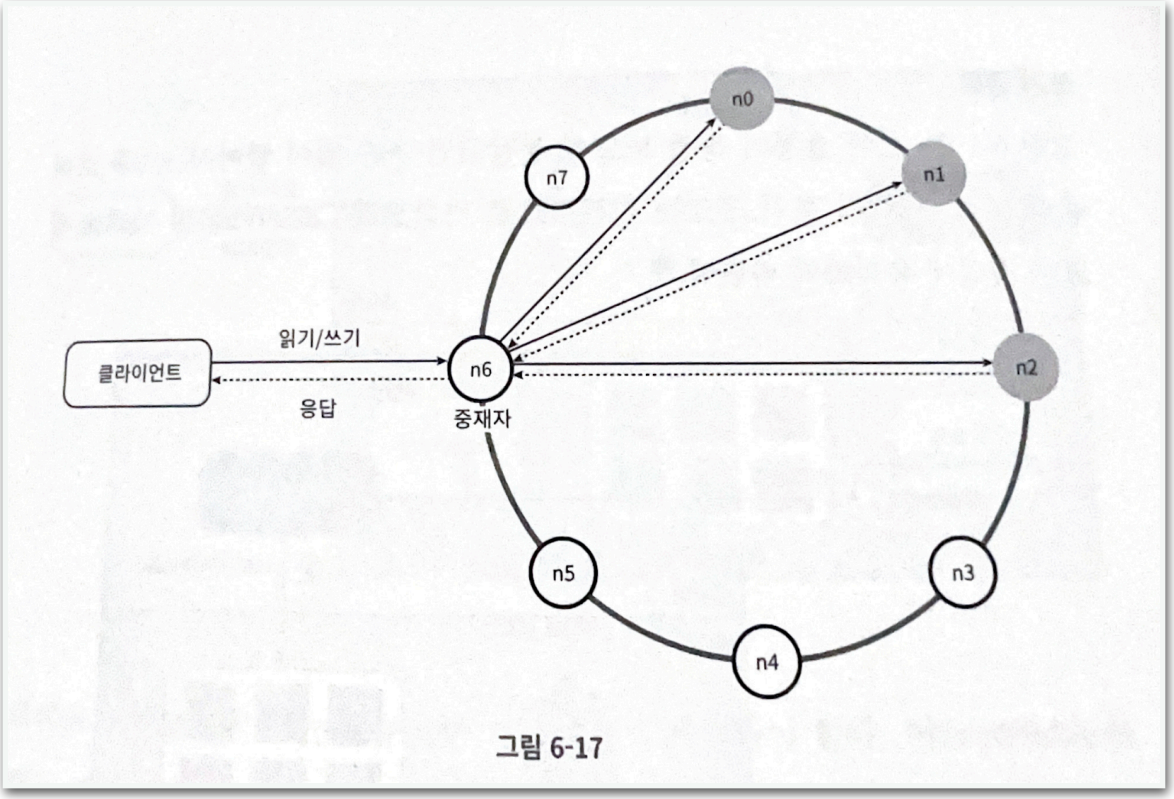


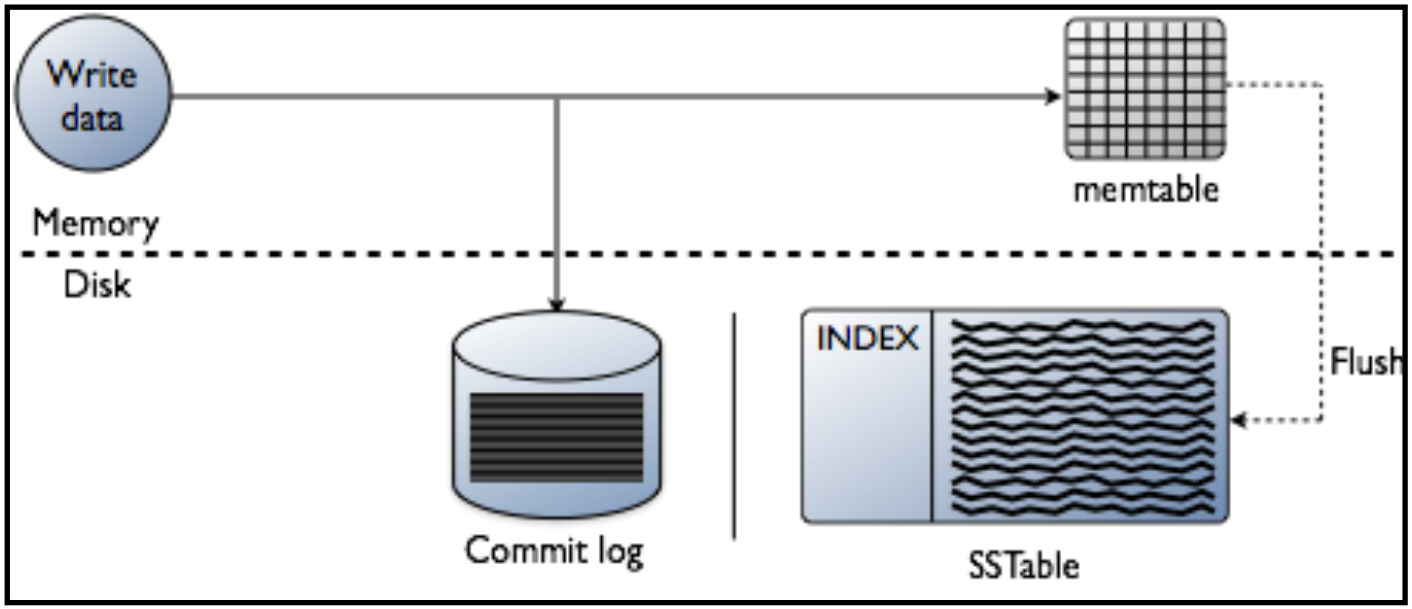
그림 6-17

키 값 저장소 설계하기

분산 서버 키 값 저장소: 시스템 아키텍처 다이어그램

쓰기경로

1. 커밋 로그에 기록
2. 데이터가 메모리 캐시에 기록
3. 메모리에 할 수 없는 경우, SSTable에 기록 (SSTable: Sorted-String Table)



https://docs.datastax.com/en/cassandra-oss/3.x/cassandra/dml/dmlHowDataWritten.html

키 값 저장소 설계하기

분산 서버 키 값 저장소: 시스템 아키텍처 다이어그램

읽기 경로

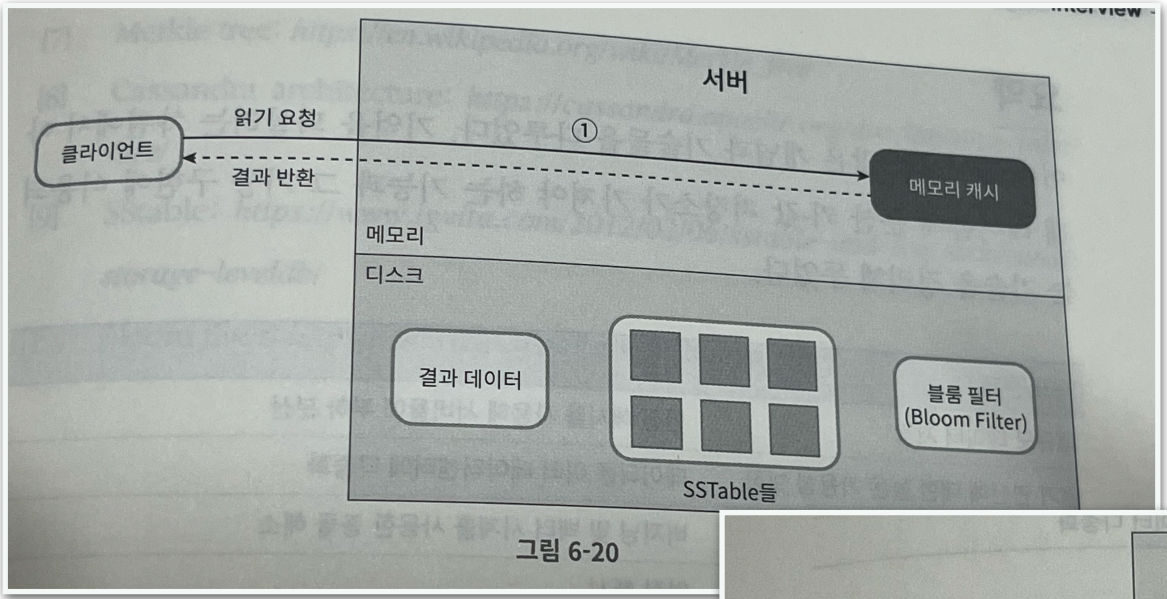


그림 6-20

데이터가 메모리에 있는 경우 바로 반환

데이터가 메모리에 없는 경우, 디스크에서 가져옴
SSTable에 있는 Key를 찾기 위해 Bloom filter 사용

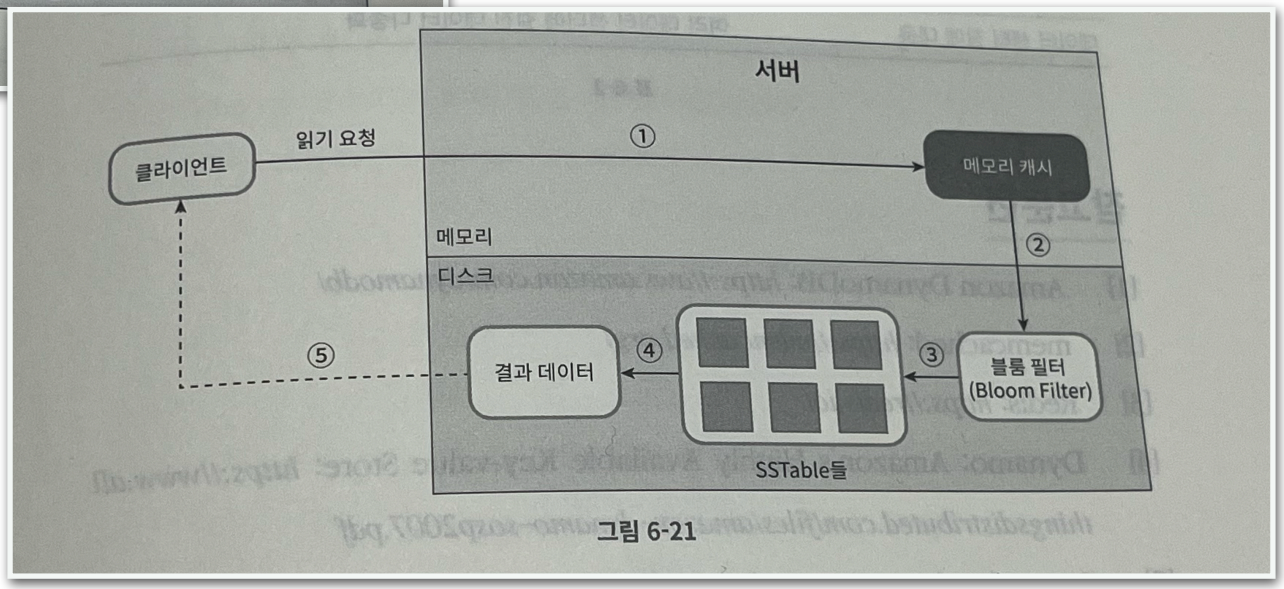


그림 6-21