[6장] 키-값 저장소 설계

가상 면접 사례로 배우는 대규모 시스템 설계 기초

이민석 / unchaptered

https://inblog.ai/monthly-cs
https://github.com/monthly-cs/2024-03-system-design-interview-1

키-값 저장소

- **비관계형 데이터베이스**
- 키는 주로 고유 식별자(Unique Identifier)
- 값은 문자열, 리스트, 객체, …

문제 이해 및 설계 범위 확정

- 키-값 쌍의 크기는 10KB
- 큰 데이터를 저장 가능
- **높은 가용성(High Availability)**
- **높은 확장성(High Scalability)**
- 데이터 일관성 수준은 조정이 가능해야함
- 응답 지연시간이 짧아야함

단일 서버 키-값 저장소

키-값 쌍 전체를 **메모리**에 해시 테이블로 저장

메모리 상한선 만큼의 용량의 제약이 있으며, 아래의 방법으로 해결 가능

1. 데이터 압축
2. 저장장소 이중화(메모리 + 디스크)

안정 해쉬 + 가상 노드

5장에서 언급한 **안정 해쉬 + 가상 노드 기법**을 이용해서 **데이터를 분산 저장**

결론

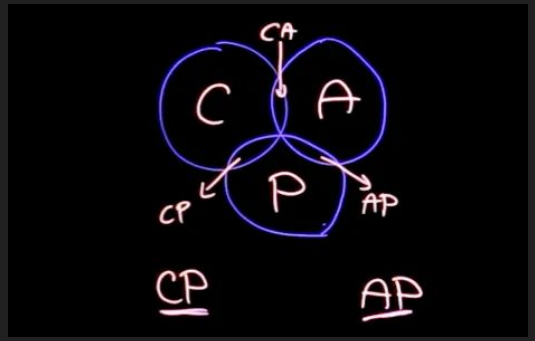
- 임의의 요청을 균등하게 분배하는데 해쉬(Hash)를 사용할 수 있다.
- 해쉬는 여러 가지 문제를 가지고 있어 안정해쉬를 써야한다.
- 안정해쉬를 구현할 때에는 가상 노드를 사용하여 분포를 고르게 유지해야 한다.

분산 서버 키-값 저장소

분산 시스템과 CAP 정리

아래 3가지 중에 2가지를 얻으려면, **1가지는 반드시 희생**되어야 함

- 데이터 일관성(Consistency) : 모든 클라이언트는 어떤 노드에서든 **같은 정보**를 봐야함
- 가용성(Availability) : 일부 노드에 장애가 발생해도 **항상 응답**을 받을 수 있어야함
- 파티션 감내(Partition Tolerance) : **네트워크에 파티션**이 생기더라도 시스템은 계속 동작해야함

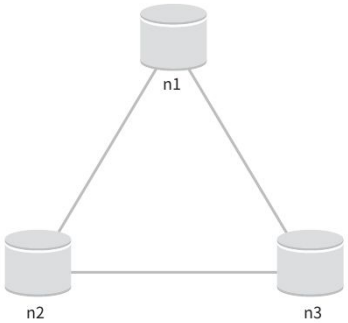


https://inblog.ai/unchaptered/whats-cap-theorem-in-distributed-system-17681

이상적 상태

이상적으로는 **네트워크 파티션**되는 상황이 **절대로 일어나지 않을 것**

- **C**onsistency | n1에 기록된 데이터는 자동으로 n2, n3에 저장됨
- **A**vailability
- **P**artition Tolerance

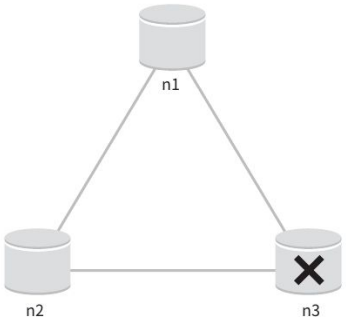


https://velog.io/@bbkyoo/%EA%B0%80%EC%83%81-%EB%A9%B4%EC%A0%91-%EC%82%AC%EB%A1%80%EB%A1%9C-%EB%B0%B0%EC%9A%B0%EB%8A%94-%EB%8C%80%EA%B7%9C%EB%AA%A8-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EC%84%A4%EA%B3%84-%EA%B8%B0%EC%B4%88-6%EC%9E%A5-%ED%82%A4-%EA%B0%92-%EC%A0%80%EC%9E%A5%EC%86%8C-%EC%84%A4%EA%B3%84

실세계의 분산 시스템

현실적으로는 **네트워크 파티션**을 피할 수 없음
그리고 네트워크 파티션은 Availability, Consistency에 영향을 미침

- **C**onsistency, **A**vailability | 둘 중 하나 포기해야함
- **P**artition Tolerance | 포기 불가



https://velog.io/@bbkyoo/%EA%B0%80%EC%83%81-%EB%A9%B4%EC%A0%91-%EC%82%AC%EB%A1%80%EB%A1%9C-%EB%B0%B0%EC%9A%B0%EB%8A%94-%EB%8C%80%EA%B7%9C%EB%AA%A8-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EC%84%A4%EA%B3%84-%EA%B8%B0%EC%B4%88-6%EC%9E%A5-%ED%82%A4-%EA%B0%92-%EC%A0%80%EC%9E%A5%EC%86%8C-%EC%84%A4%EA%B3%84

데이터 파티션

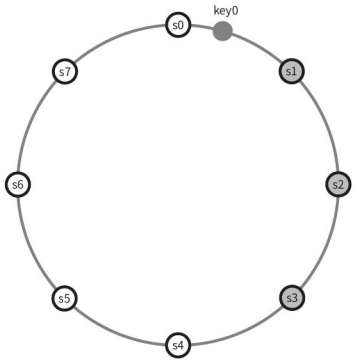
데이터를 어떻게 다양한 파티션에 분배할 것인가?

→ **안정 해쉬(Consistent Hash)**를 사용해서 가능

데이터 다중화

임의의 키(K0)를 다수의 서버(S1, S2, S3)에 분할해야함

- 가상 노드(S1_a, S2_b, S3_c)일 경우에는 a,b,c가 모두 한 서버일 수 있음
- 이 부분은 주의해야함

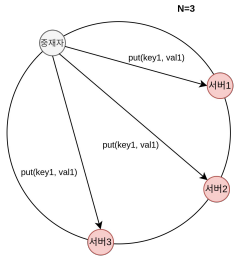


https://velog.io/@bbkyoo/%EA%B0%80%EC%83%81-%EB%A9%B4%EC%A0%91-%EC%82%AC%EB%A1%80%EB%A1%9C-%EB%B0%B0%EC%9A%B0%EB%8A%94-%EB%8C%80%EA%B7%9C%EB%AA%A8-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EC%84%A4%EA%B3%84-%EA%B8%B0%EC%B4%88-6%EC%9E%A5-%ED%82%A4-%EA%B0%92-%EC%A0%80%EC%9E%A5%EC%86%8C-%EC%84%A4%EA%B3%84

데이터 일관성

다중화된 데이터는 적절히 **동기화**되어야 함
정족수의 합(Quorum Consensus) 프로토콜을 사용하면 읽기/쓰기에 일관성 보장 가능

- N | 사본 갯수
- W | 쓰기 연산에 대한 정족수쓰기 연산에 대한 정족수
- R | 읽기 연산에 대한 정족수



https://velog.io/@bbkyoo/%EA%B0%80%EC%83%81-%EB%A9%B4%EC%A0%91-%EC%82%AC%EB%A1%80%EB%A1%9C-%EB%B0%B0%EC%9A%B0%EB%8A%94-%EB%8C%80%EA%B7%9C%EB%AA%A8-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EC%84%A4%EA%B3%84-%EA%B8%B0%EC%B4%88-6%EC%9E%A5-%ED%82%A4-%EA%B0%92-%EC%A0%80%EC%9E%A5%EC%86%8C-%EC%84%A4%EA%B3%84

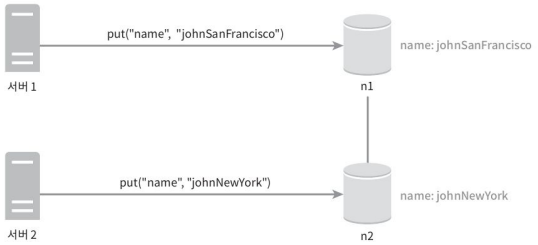
일관성 모델

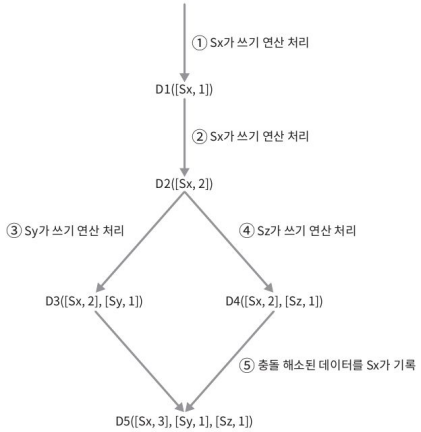
키-값 저장소를 설계할 때, 고려해야 할 요소

- **강한** 일관성 : 가장 최근에 갱신된 결과를 **반드시 반환**
- **약한** 일관성 : 가장 최근에 갱신된 결과를 **반환하지 못할 수 있음**
- **최종** 일관성 : 갱신된 결과들이 결국에는 모든 사본에 반영이 됨

비 일관성 해소 기법 : 데이터 버저닝과 벡터 사계

쓰기 작업에 대해서 높은 가용성(HA)보장





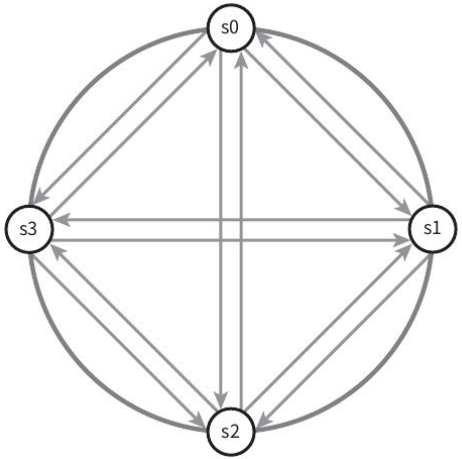
https://velog.io/@bbkyoo/%EA%B0%80%EC%83%81-%EB%A9%B4%EC%A0%91-%EC%82%AC%EB%A1%80%EB%A1%9C-%EB%B0%B0%EC%9A%B0%EB%8A%94-%EB%8C%80%EA%B7%9C%EB%AA%A8-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EC%84%A4%EA%B3%84-%EA%B8%B0%EC%B4%88-6%EC%9E%A5-%ED%82%A4-%EA%B0%92-%EC%A0%80%EC%9E%A5%EC%86%8C-%EC%84%A4%EA%B3%84

장애 처리

우선 **장애 감지(Failure Detection)**과 **장애 해소(Failure Resolution)**의 대책이 필요

- 전체적으로 장애를 어떻게 감지할것인지
- 장애를 어떻게 해소할 것인지

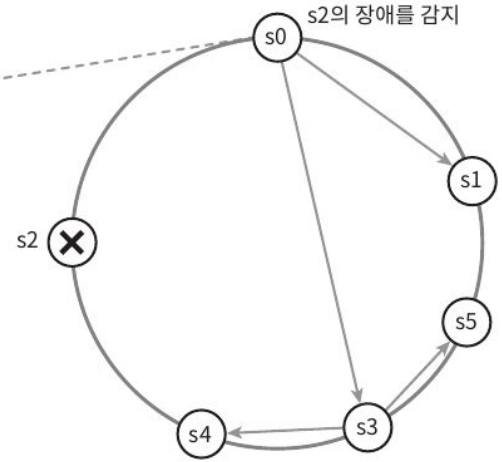
장애 감지



멀티캐스킹
(Multicasting)

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

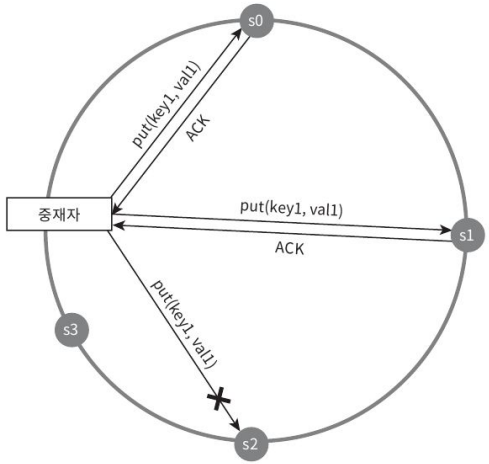


분산형 장애 감지
(Decentralized Failure Detection)

https://velog.io/@bbkyoo/%EA%B0%80%EC%83%81-%EB%A9%B4%EC%A0%91-%EC%82%AC%EB%A1%80%EB%A1%9C-%EB%B0%B0%EC%9A%B0%EB%8A%94-%EB%8C%80%EA%B7%9C%EB%AA%A8-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EC%84%A4%EA%B3%84-%EA%B8%B0%EC%B4%88-6%EC%9E%A5-%ED%82%A4-%EA%B0%92-%EC%A0%80%EC%9E%A5%EC%86%8C-%EC%84%A4%EA%B3%84

일시적 장애 처리

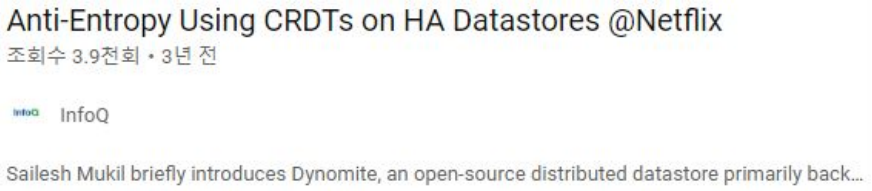
- **엄격한 정족수** : 일시적 장애 발생 시, 쓰기/읽기를 중단
- **느슨한 정족수** : 데이터 다중화를 하면서, 일부 쓰기 작업이 실패 시, 성공한 노드에 **단서 후 임시 위탁(Hint Handoff)**



https://velog.io/@bbkyoo/%EA%B0%80%EC%83%81-%EB%A9%B4%EC%A0%91-%EC%82%AC%EB%A1%80%EB%A1%9C-%EB%B0%B0%EC%9A%B0%EB%8A%94-%EB%8C%80%EA%B7%9C%EB%AA%A8-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EC%84%A4%EA%B3%84-%EA%B8%B0%EC%B4%88-6%EC%9E%A5-%ED%82%A4-%EA%B0%92-%EC%A0%80%EC%9E%A5%EC%86%8C-%EC%84%A4%EA%B3%84

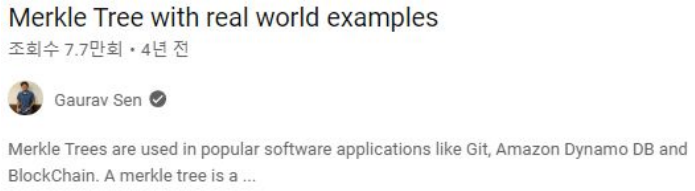
영구 장애 처리

반-엔트로피 프로토콜을 사용하여 동기화 + **머클 트리 알고리즘** 사용



반-엔트로피 프로토콜
(Anti Entropy Protocol)

https://youtu.be/JirCwLXlH_c?si=fAHqX2fYJTaXidry



머클 트리
(Mukle Tree)

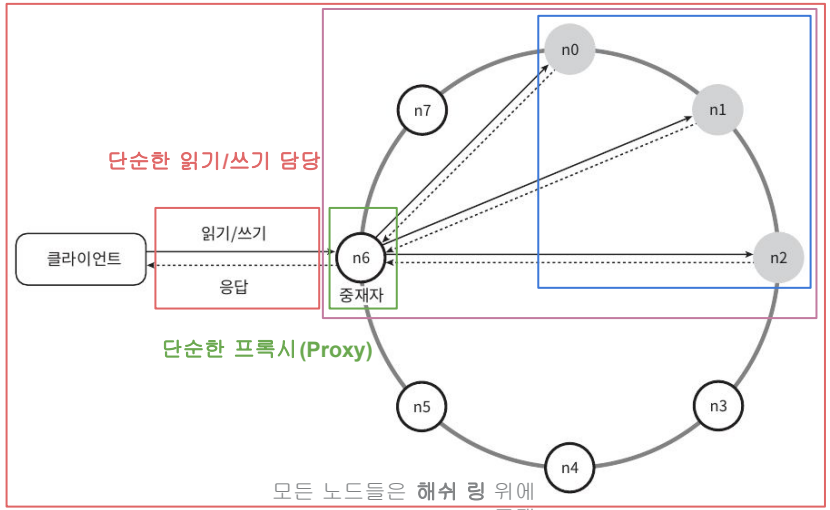
https://youtu.be/qHMLy5JjbjQ?si=rdzdQqVdhobwPVLA

데이터 센터 장애 처리

데이터를 **여러 데이터 센터에 다중화**하는 처리 필요

마스터가 다운되면 슬레이브가 승격되는 처리 필요

시스템 아키텍처 다이어그램



정족수의 합 프로토콜으로
데이터 일관성을 유지

모든 노드는 같은 책임을
지며, **SPoF가 없음**

분산형 장애 감지 기술을 사용하며,
일시적 장애는 느슨한 정족수의 합 프로토콜로
영구적 장애는 반-엔트로피 프로토콜에 머클 트리로
복원

다수의 물리적인 지역에 분산할 필요가 있음

https://velog.io/@bbkyoo/%EA%B0%80%EC%83%81-%EB%A9%B4%EC%A0%91-%EC%82%AC%EB%A1%80%EB%A1%9C-%EB%B0%B0%EC%9A%B0%EB%8A%94-%EB%8C%80%EA%B7%9C%EB%AA%A8-%EC%8B%9C%EC%8A%A4%ED%85%9C-%EC%84%A4%EA%B3%84-%EA%B8%B0%EC%B4%88-6%EC%9E%A5-%ED%82%A4-%EA%B0%92-%EC%A0%80%EC%9E%A5%EC%86%8C-%EC%84%A4%EA%B3%84

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

감사합니다.