6장

강의, 교재명	가상 면접 사례로 배우는 대규모 시스템 설계 기초1편
생성 일시	@2024년 2월 17일 오후 4:12

데이터 다중화

- N개의 서버에 데이터를 다중화하여 높은 가용성과 안정성을 확보한다.

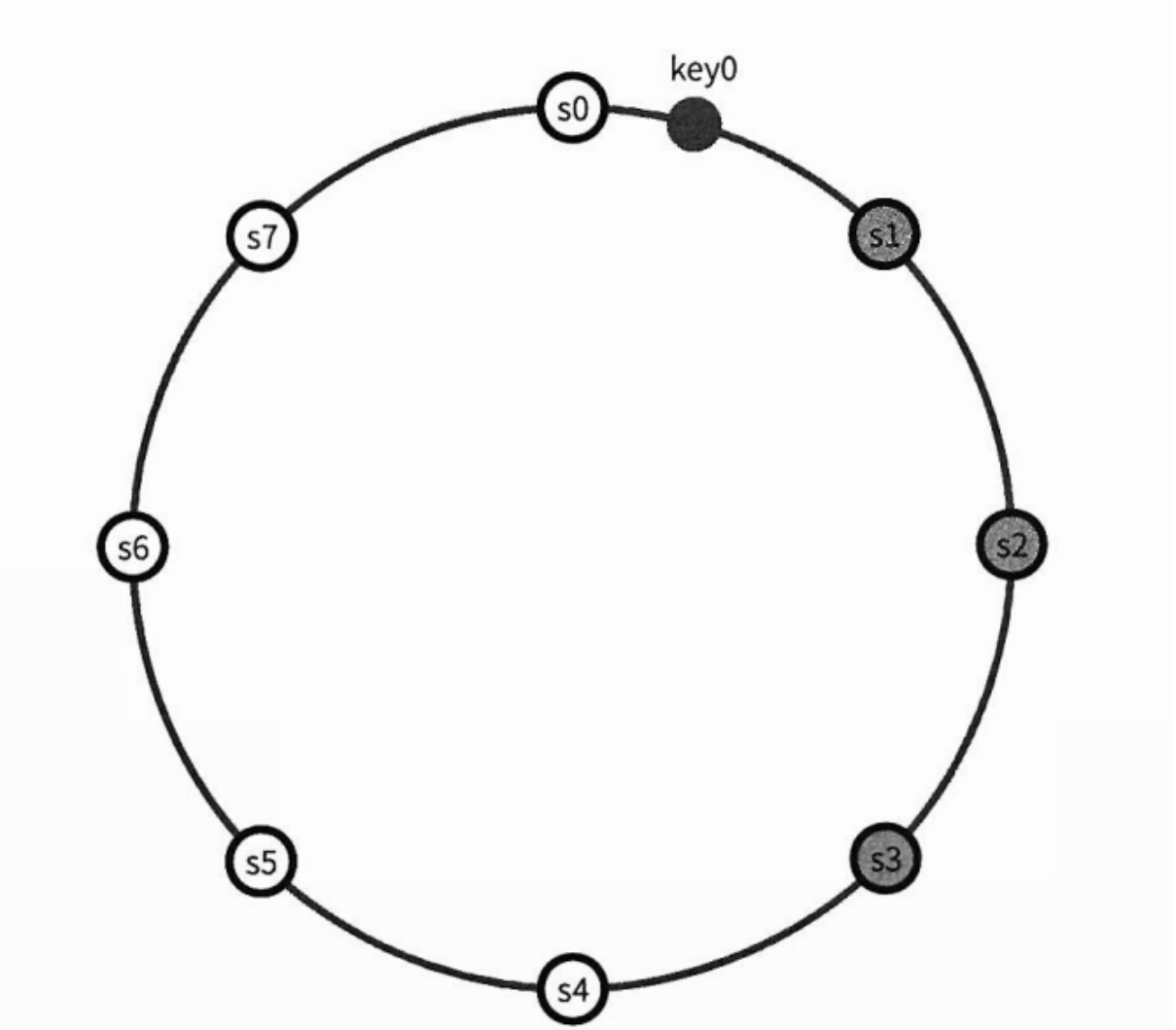


그림 6-5

N = 3일 때 key0은 s1,2,3에 저장된다.

어떤 키를 해시 링 위에 배치 후 시계 방향으로 링을 순회하면서 만나는 첫 N개의 서버에 데이터 사본을 보관

가상 노드를 사용한다면?

- 1, 2, 3 번의 물리 서버가 존재하고 N = 3이라면
- 1번의 가상 노드에만 저장하는 상황이 생길 수도 있다.
- 이렇게 되면 실제 물리 서버 1번에만 저장이 돼고 장애를 대응하기 위해 다중화시키려는 취지와 맞지 않게 된다.
- 같은 물리 서버의 가상 노드를 중복 선택하지 않게끔 해야한다.

데이터 일관성

- 여러 노드에 다중화된 데이터는 적절히 동기화 되어야 한다.
- 정족수 합의 프로토콜을 사용하면 읽기/쓰기 연산 모두에 일관성을 보장한다.

정족수 합의 프로토콜

- N = 사본 개수
- W = 쓰기 연산에 대한 정족수
    - 적어도 W개의 서버로부터 성공했다는 응답을 받아야 성공한 것으로 간주
- R = 읽기 연산에 대한 정족수
    - 적어도 R개의 서버로부터 성공했다는 응답을 받아야 성공한 것으로 간주

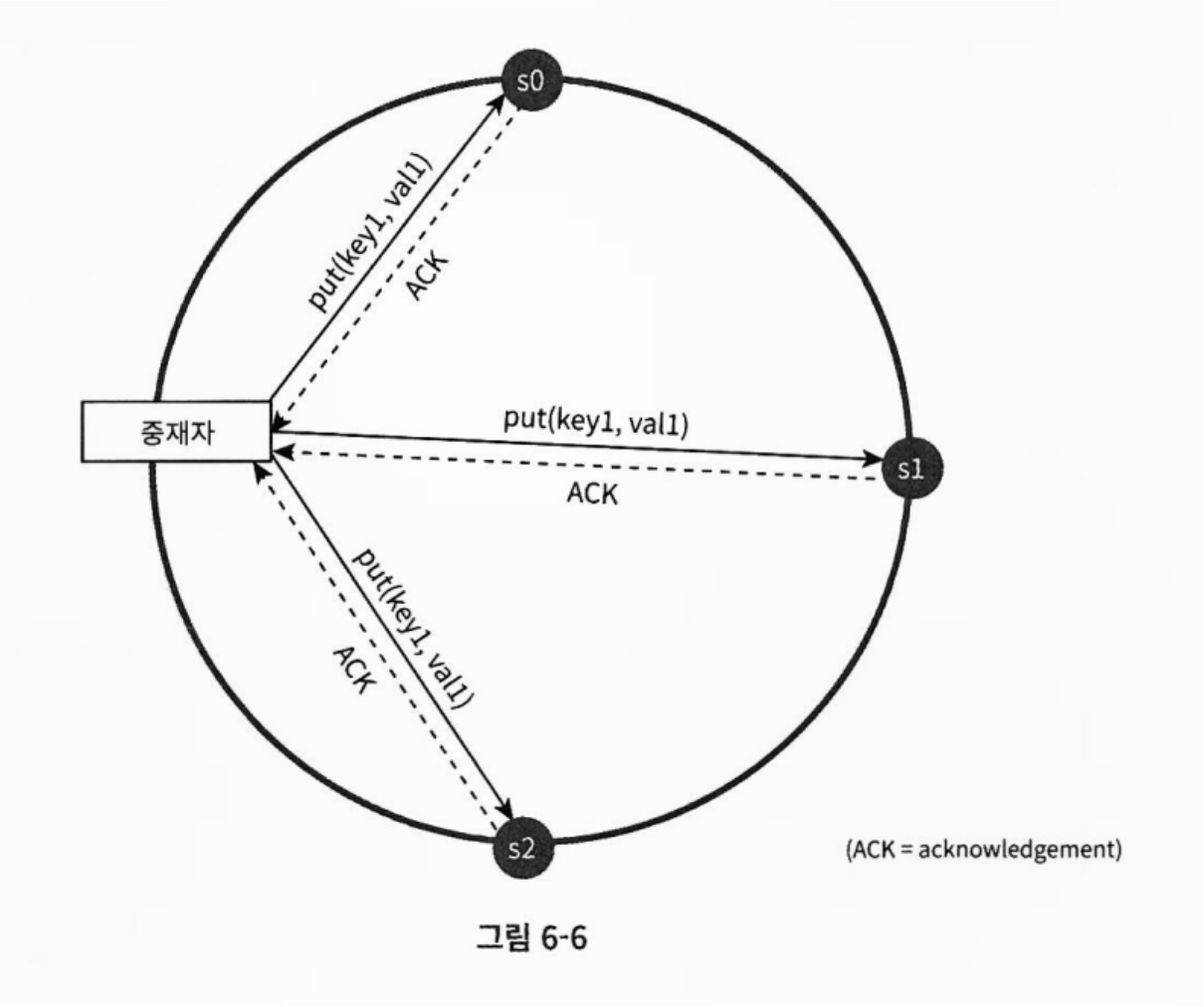


그림 6-6

- W = 1
  - 중재자는 최소 한 대 서버롭터 쓰기 성공 응답을 받아야 한다는 뜻
  - s1으로부터 성공 응답을 받았다면 s0, s2 응답은 기다리지 않아도 됨
  - 중재자는 클라이언트와 노드 사이에서 프록시 역할을 함
- W + R > N (쓰기, 읽기 서버로부터 최소 1개 이상의 ack 갯수 보장)
  - 강한 일관성이 보장된다.
- W + R ≤ N
  - 강한 일관성이 보장되지 않음.
- W = 1, R = N
  - 빠른 쓰기 연산에 최적화된 시스템
- R = 1, W = N
  - 빠른 읽기 연산에 최적화된 시스템

일관성 모델

- 키-값 저장소를 설계할 때 고려해야 할 중요한 요소
  - 강한 일관성 : 모든 읽기 연산은 가장 최근에 갱신(Write)된 결과를 반환
  - 약한 일관성 : 읽기 연산은 가장 최근에 갱신된 결과를 반환하지 못할 수 있다.
  - 최종 일관성 : 약한 일관성의 한 형태로, 갱신 결과가 결국에는 모든 사본에 반영(동기화)되는 모델
- 강한 일관성은 일반적으로 모든 사본에 현재 쓰기 연산의 결과가 반영될 때까지 해당 데이터에 대한 읽기/쓰기를 금지하는 방법이라 고가용성 시스템에는 적잡하지 않다.
- 다이나모, 카산드라는 최종 일관성 모델을 채택
- 최종 일관성 모델은 쓰기 연산이 병렬적으로 발생하면 시스템에 저장된 값의 일관성이 깨질 수 있다.

비 일관성 해소 기법: 데이터 버저닝

- 버저닝
  - 데이터를 변경할 떄마다 해당 데이터의 새로운 버전을 만든다.
-

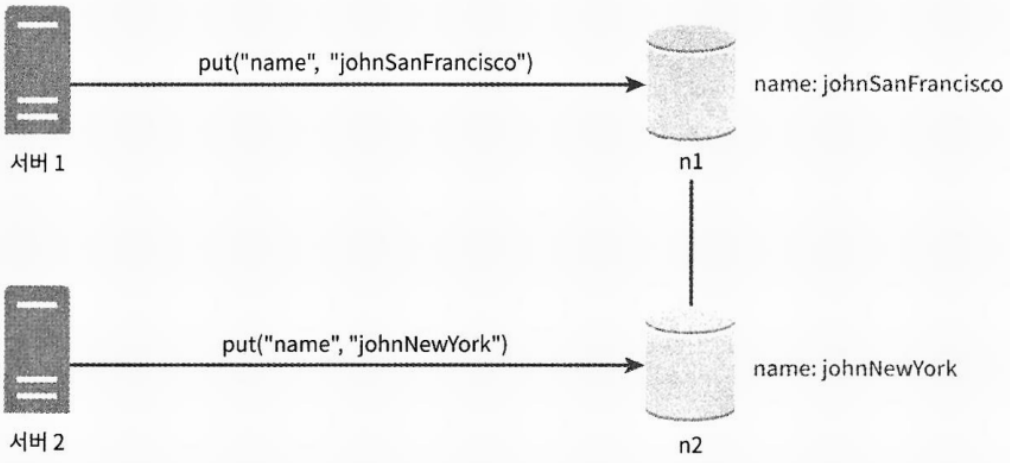


그림 6-8

이 변경이 이루어진 이후에, 원래 값은 무시할 수 있다. 변경이 끝난 옛날 값이어서다. 하지만 마지막 두 버전 v1과 v2 사이의 충돌은 해소하기 어려워 보인다. 이 문제를 해결하려면, 충돌을 발견하고 자동으로 해결해 낼 버저닝 시스템이 필요하다. 벡터 시계(vector clock)는 이런 문제를 푸는데 보편적으로 사용되는 기술이다. 지금부터 그 동작 원리를 살펴보자.

벡터 시계

- [서버, 버전]의 순서쌍을 데이터에 매단 것
- D([S1, v1], [S2, v2], ..., [Sn, vn])과 같이 표현한다고 가정하면
- 만일 데이터 D를 서버 Si에 기록하면, 시스템은 아래 작업을 수행
  - [Si, vi]가 있으면 vi를 증가
  - 그렇지 않으면 새 항목 [Si, 1]를 만든다.

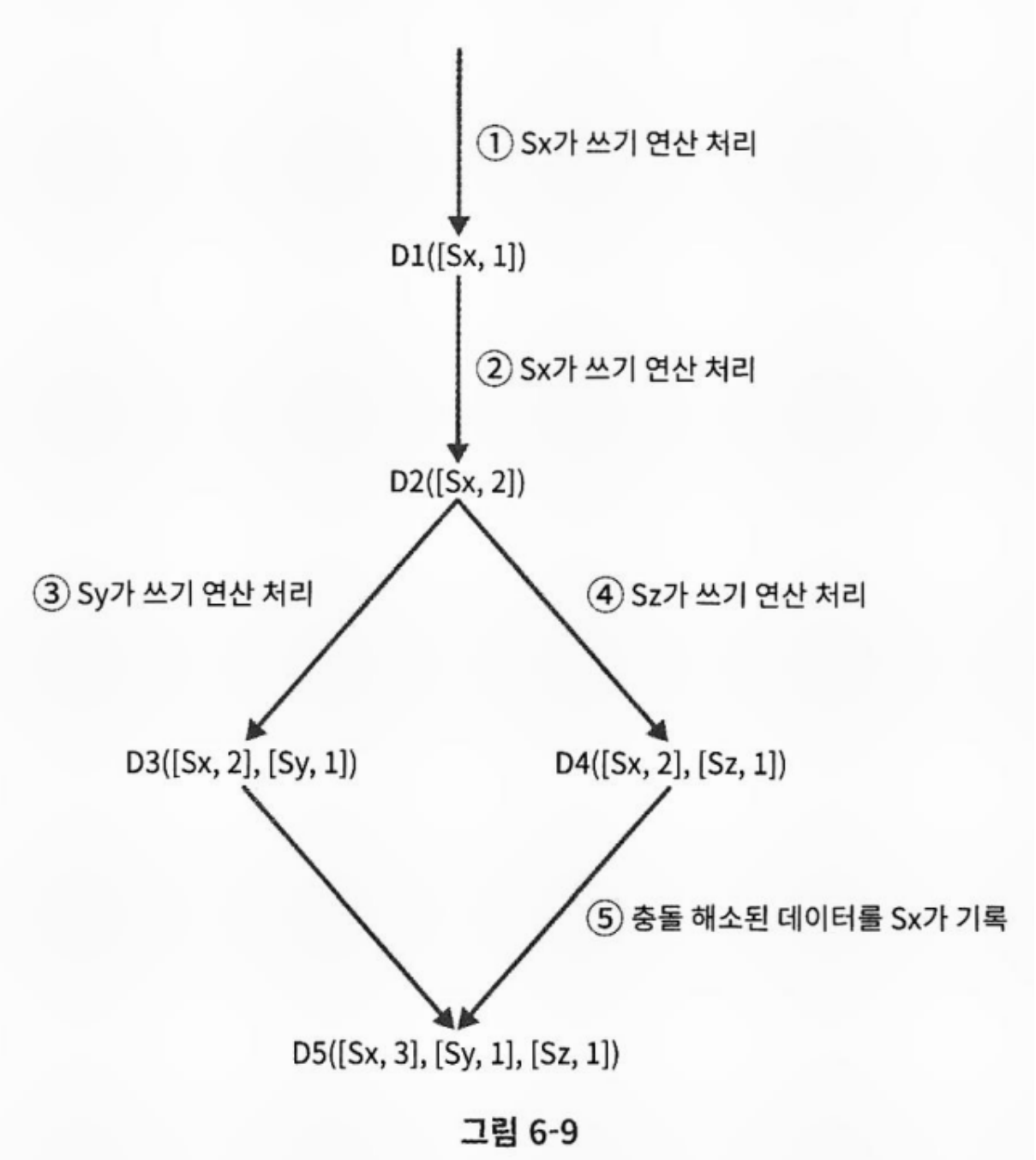


그림 6-9

1. 클라이언트가 데이터 D1을 시스템에 기록하고 쓰기 연산을 처리한 서버는 Sx 따라서 벡터 시계는 **D1[(Sx, 1)]**으로 변함
2. 다른 클라이언트가 데이터 D1을 읽고 D2로 업데이트한 다음 기록하고 D2는 D1에 대한 변경이므로 D1을 덮어쓴다. 이 때 Sx가 쓰기 연산 처리한다면 벡터 시계는 **D2([Sx, 2])**로 바뀜
3. 다른 클라이언트가 D2를 읽어 D3로 갱신한 다음 기록하고 쓰기 연산은 Sy가 처리하면 **D3([Sx, 2], [Sy, 1])**로 바뀜

4. 또 다른 클라이언트가 D2를 읽고 D4로 갱신한 다음 기록하고 쓰기 연산은 Sz가 처리하면 **D4([Sx, 2], [Sz, 1])**일 것이다.
5. 어떤 클라이언트가 D3와 D4를 읽으면 데이터 간 충돌이 있다는 것을 인지
   D2를 Sy와 Sz가 각기 다른 값으로 바꾸었기 때문
   이 충돌은 클라이언트가 해소한 후에 서버에 기록 이때 쓰기 연산은 Sx가 처리하면
   D5[Sx, 3], [Sy, 1], [Sz, 1])로 바뀜

장점

- 버전 Y에 포함된 모든 구성요소의 값이 X에 포함된 모든 구성요소 값보다 같거나 큰지만 보면 되는 벡터 시계를 사용하면 어떤 버전 X가 버전 Y의 이전 버전인지(충돌이 없는지) 쉽게 판단
- D([s0, 1], [s1, 2])와 D([s0, 2], [s1, 1])는 서로 충돌

단점

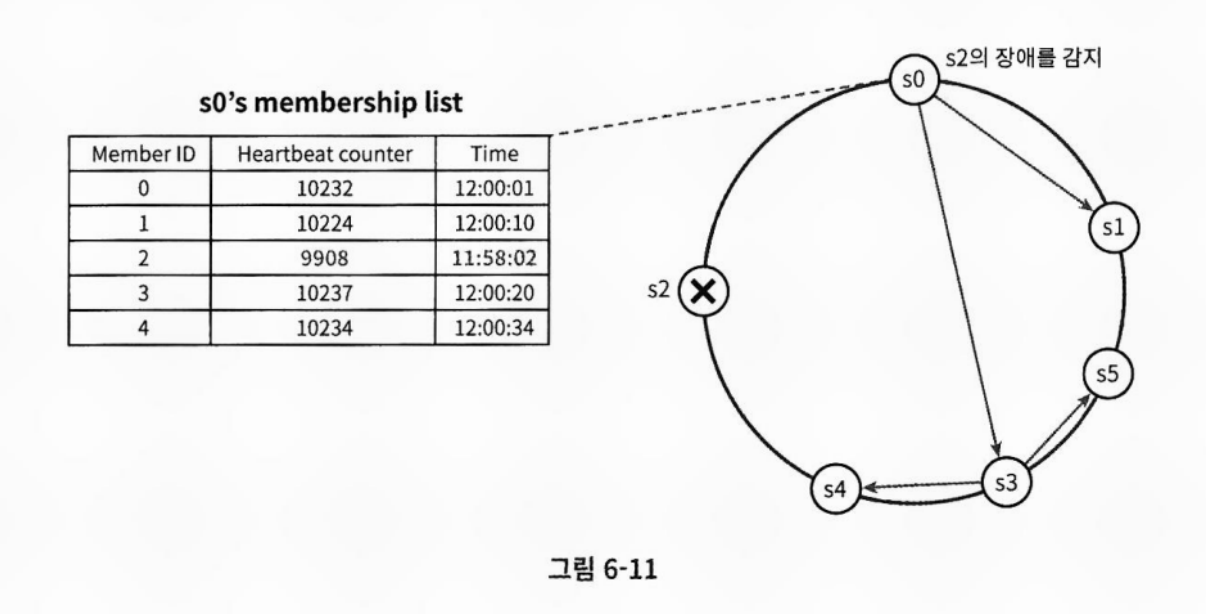
- 충돌 감지 및 해소 로직이 클라이언트에 들어가야 하므로, 클라이언트 구현이 복잡해짐
- [서버, 버전]의 순서쌍 개수가 굉장히 빨리 늘어난다.

가십 프로토콜

- 서버의 장애 감지를 위한 솔루션 중 하나

동작 원리

- 각 노드는 멤버 ID와 박동 카운터를 쌍으로 하는 멤버십 목록을 유지
- 주기적으로 박동 카운터 증가시킴
- 무작위로 선정된 노드들에게 주기적으로 자기 박동 카운터 목록을 보냄
- 목록을 받은 노드는 멤버십 목록을 최신 값으로 갱신
- 어떤 멤버의 박동 카운터 값이 지정된 시간 동안 갱신되지 않으면 해당 멤버는 장애 상태인 것으로 간주

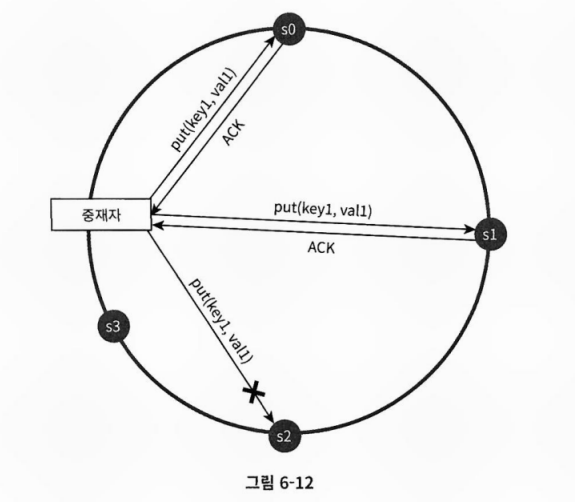


가십 프로토콜 동작 원리

일시적 장애 처리(Hinted handoff)

후임시 위탁 기법

- 정족수는 결정을 내리거나 행동을 취하는 데 필요한 최소한의 참석자 수나 동의 수를 의미
- 느슨한 정족수 접근법을 이용해 쓰기 연산을 수행할 W개의 건강한 서버와 읽기 연산을 수행할 R개의 건강한 서버를 해시 링에서 고른다.
- 이 때 장애 상태인 서버는 무시
- 장애 서버로 가는 요청을 선택해둔 서버에 잠시 동안 위임하여 처리한다.
- 장애 서버가 복구되었을 때 그동안 발생한 변경사항을 일괄 반영하여 일관성 보존함
- 이를 위해 임시로 쓰기 연산을 처리한 서버에는 그에 관한 단서를 남겨둠



후임시 위탁 기법

영구 장애 처리

반-엔트로피 프로토콜

- 사본들을 비교하여 최신 버전으로 갱신하는 과정
- 사본 간의 일관성이 망가진 상태를 탐지하고 전송 데이터의 양을 줄이기 위해서 **머클트리** 사용

머클트리

생성 과정

1. **데이터 조각의 해시 계산**: 먼저 파일을 여러 데이터 조각으로 나눈다. 각 조각에 대해 해시 함수를 사용하여 고유한 해시 값을 계산
2. **해시 쌍 생성**: 계산된 해시 값들을 두 개씩 짝지어 그 쌍의 해시 값을 다시 계산하고 이 과정을 통해 새로운 해시 값들이 생성
3. **해시 쌍의 해시 계산**: 새로 생성된 해시 값들을 다시 두 개씩 짝지어 해시를 계산하고 이 과정을 반복하면서 트리의 상위 레벨로 올라감
4. **루트 해시 도출**: 최종적으로 하나의 해시 값, 즉 루트 해시에 도달할 때까지 이 과정을 반복하고 루트 해시는 전체 파일의 무결성을 나타냄

검증 과정

1. **부분 검증**: 이제 파일의 특정 부분만 검증하고 싶다고 가정해 보면, 해당 데이터 조각의 해시 값과 그것과 직접 연결된 해시 값들만 알고 있으면 된다.

2. **필요한 해시 값들 수집**: 검증하고자 하는 데이터 조각으로부터 시작하여, 루트 해시에 이르기까지 경로에 있는 모든 해시 값을 수집함
3. **해시 값들 비교**: 수집한 해시 값들을 이용해, 데이터 조각에서 시작하여 루트에 이르는 해시 경로를 재현한다. 만약 모든 해시 값들이 일치하면, 데이터 조각이 원본과 정확히 일치함을 의미한다.