6장 키-값 저장소 설계

key-value store

- 키-값 데이터베이스라고도 불리는 비 관계형(non-relational) 데이터베이스
- e.g. redis, memcached, AWS DynamoDB

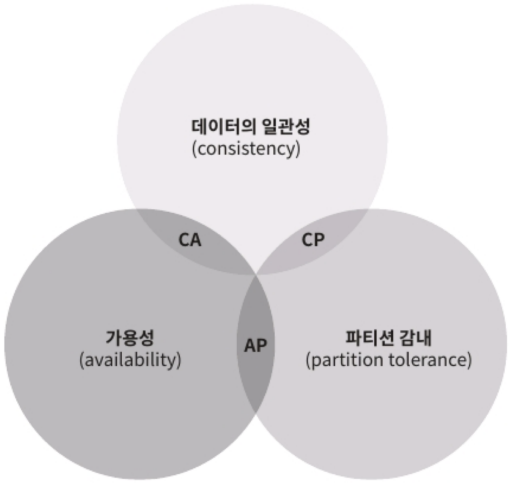
단일 key-value store

- 메모리에 해시 테이블로 저장하는 쉬운 방법이다.
- 규모가 커지면 모든 데이터를 메모리 안에 두는 것은 불가능
  - 데이터 압축
  - 디스크에 분산해 저장
  - 분산 key-value store

분산 key-value store

- CAP 정리
  - `일관성(Consistency)`, `가용성(Availability)`, `파티션 감내(Partition Tolerance)` 모든 요구사항을 동시에 만족하는 분산 시스템을 설계하는 것은 불가능하다.
  - 일관성 : 모든 클라이언트는 어떤 노드에 접속했는지 상관없이 언제나 같은 데이터를 보게 되어야 한다.
  - 가용성 : 일부 노드에 장애가 발생하더라도 항상 응답을 받을 수 있어야 한다.
  - 파티션 감내 : 두 노드 사이에 통신 장애가 발생했더라도 시스템은 계속 동작해야 한다.

- CP : `일관성`, `파티션 감내` 지원
- AP : `가용성`, `파티션 감내` 지원
- CA : `일관성`, `가용성` 지원



시스템 컴포넌트

데이터 파티션

- 데이터를 한 대 서버에 욱여넣기 불가능하기 때문에, 작은 파티션들로 분할한 다음 여러 대에 저장
- 고려 사항 2가지
  - 데이터를 여러 서버에 고르게 분산할 수 있는가
  - 노드가 추가되거나 삭제될 때 데이터 이동을 최소화 할 수 있는가
- Consistent hashing 사용으로 얻는 장점
  - 규모 확장 자동화
  - 다양성 : 서버의 용량에 맞게 가상 노드 수를 조절할 수 있다.

데이터 다중화

- 높은 가용성과 안정성을 확보하기 위해 N개 서버에 비동기적으로 replication이 필요
- 해시 링 위에 키가 먼저 만나는 N개 서버에 데이터 사본을 보관
- 가상 노드 사용 시 실제 replication된 물리 서버의 수가 N보다 작을 수 있음
  → 같은 물리 서버를 중복 선택하지 않도록 해야한다.

데이터 일관성

- 여러 노드에 다중화된 데이터는 적절히 동기화되어야함
- `정족수 합의(Quorum Consensus)` 프로토콜 사용해 일관성 보장
  - **N**: 서버 사본의 개수
  - **W**: 쓰기 연산에 대한 정족수. 쓰기 연산이 성공한 것으로 간주되려면 적어도 W개의 서버로부터 쓰기 연산이 성공했다는 응답을 받아야 한다.
  - **R**: 읽기 연산에 대한 정족수. 읽기 연산이 성공한 것으로 간주되려면 적어도 R개의 서버로부터 응답을 받아야 한다.
- 요구사항에 따라 다음과 같은 일관성 모델을 적절히 선택해야함 (일관성, 가용성 Trade off)
  - `강한 일관성`: 클라이언트는 절대 낡은 데이터를 볼 수 없다.
  - `약한 일관성`: 읽기 연산은 가장 최근에 갱신된 결과를 반환하지 못할 수 있다.
  - `최종 일관성`: 약한 일관성의 한 형태. 결국에는 모든 사본에 최신 데이터가 동기화 되는 모델

비 일관성 해소 기법 : 데이터 버저닝

- 다중화를 통해 가용성이 높아지지만 사본 간 일관성이 깨질 가능성이 높아지는데, 이를 버저닝으로 해결
- `버저닝(versioning)`과 `벡터 시계(vector clock)`을 이용해 데이터를 변경할 때 마다 새로운 버전을 만든다.
  - 각 버전의 데이터는 변경 불가능
  - [서버, 버전] 순서쌍을 붙여 버저닝
- 한계
  - 충돌 감지, 해소를 클라이언트에서 처리해야함
  - [서버, 버전] 순서쌍이 굉장히 빠르게 늘어날 수 있음
    - 임계치 설정 후 오래된 순서쌍 제거
    - 순서쌍 제거시 이론상 선후 관계 정확하지 않아 충돌 해소 과정에 문제 발생 가능

→ DynamoDB에서 실제로 그런 문제가 발생한 것을 발견한 적이 없다고 함

장애 감지

- 모든 노드 사이에 멀티캐스팅 채널은 비효율적인 방법
- 가십 프로토콜로 장애 감지
  - 맴버십 목록을 유지해 그들의 박동 카운터를 저장한다.
  - 박동 카운터를 무작위 노드들에게 주기적으로 전송
  - 특정 노드의 박동 카운터가 오랜 시간 갱신되지 않으면 장애 파악 후 전파

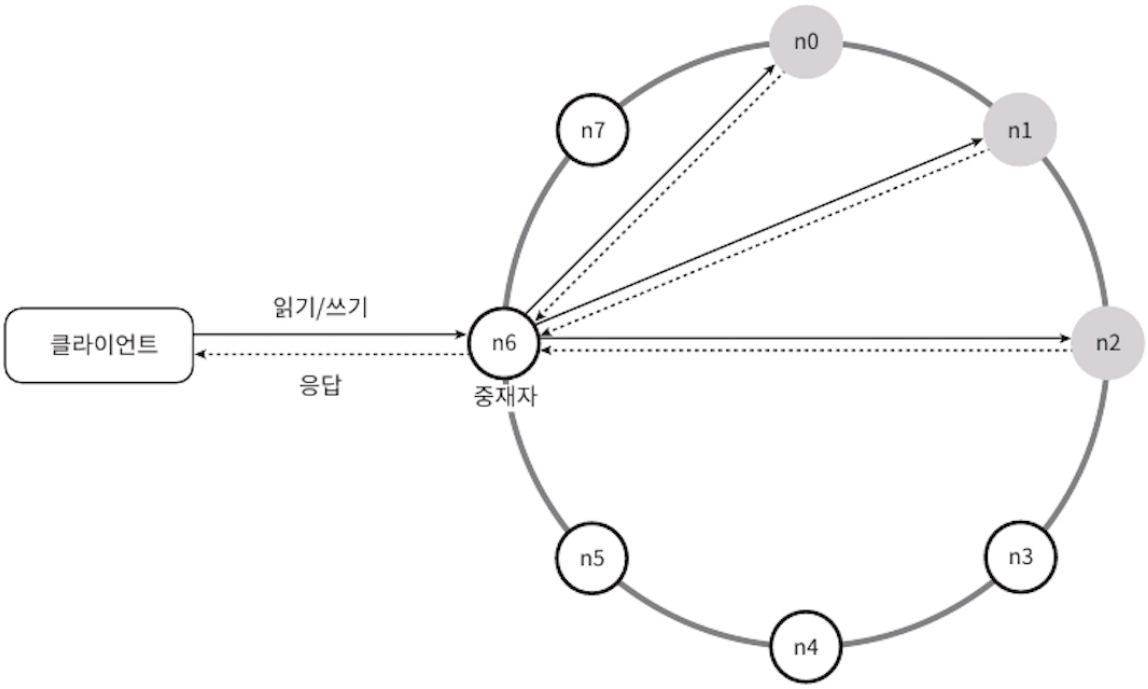
일시적 장애 처리

- 가십 프로토콜을 통해 장애를 감지한 시스템은 가용성 보장을 위해 조치를 취해야 한다.
- 엄격한 정족수/느슨한 정족수 접근법에 따라 건강한 읽기, 쓰기 연산 금지/상태의 서버 사용 조취를 취한다.
- 데이터 일관성을 보존하기 위해 장애 서버가 복구되었을 때 그동안 발생한 변경사항을 일괄 반영함
  - 이를 위해 서버에는 그에 관한 단서(hint)들을 남기고 이런 장애 처리 방안을 `단서 후 임시 위탁(hinted handoff)` 기법이라 부름

영구 장애 처리

- 영구 장애 상태에는 `anti-entropy` 프로토콜을 구현하여 사본들을 동기화
  - 사본 간 일관성이 망가진 상태를 탐지하고 전송 데이터 양을 줄이기 위해 `머클(Merkle) 트리` 사용

최종 아키텍처 다이어그램



- 클라이언트와 단순한 API로 통신: get(key), put(key, value)
- 중재자는 클라이언트에게 key-value store에 대한 proxy역할을 하는 노드
- 노드는 consistent hash 위에 존재
- 노드를 자동으로 추가, 삭제 가능하도록 시스템은 완전히 분산
- 데이터는 여러 노드에 다중화되어 있다.
- 모든 노드가 같은 책임을 지므로 SPOF(Single Point Of Failure)는 존재하지 않는다.