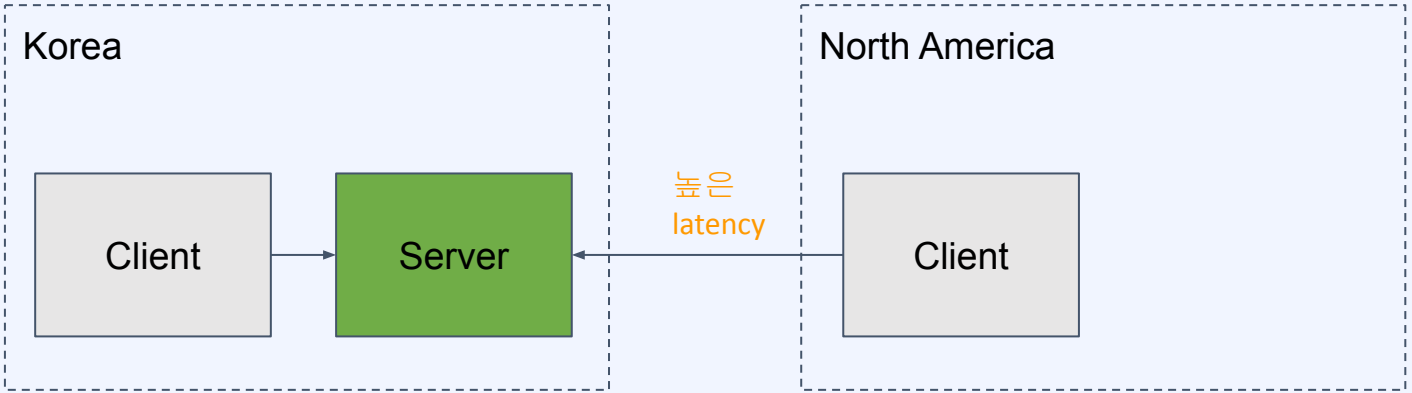
글로벌 서비스를 위한 Active-Active Architecture

1 글로벌 서비스에서 보다 빠른 서비스를 위한 Active-Active 구조

글로벌 서비스 혹은 다중 지역(multi-region) 서비스의 어려움

- 한 지역에 서버를 두고 서비스하면 멀리 떨어진 곳에서는 latency 문제가 있음
- 여러 지역에 서버를 두면 데이터 일관성에 문제가 있음



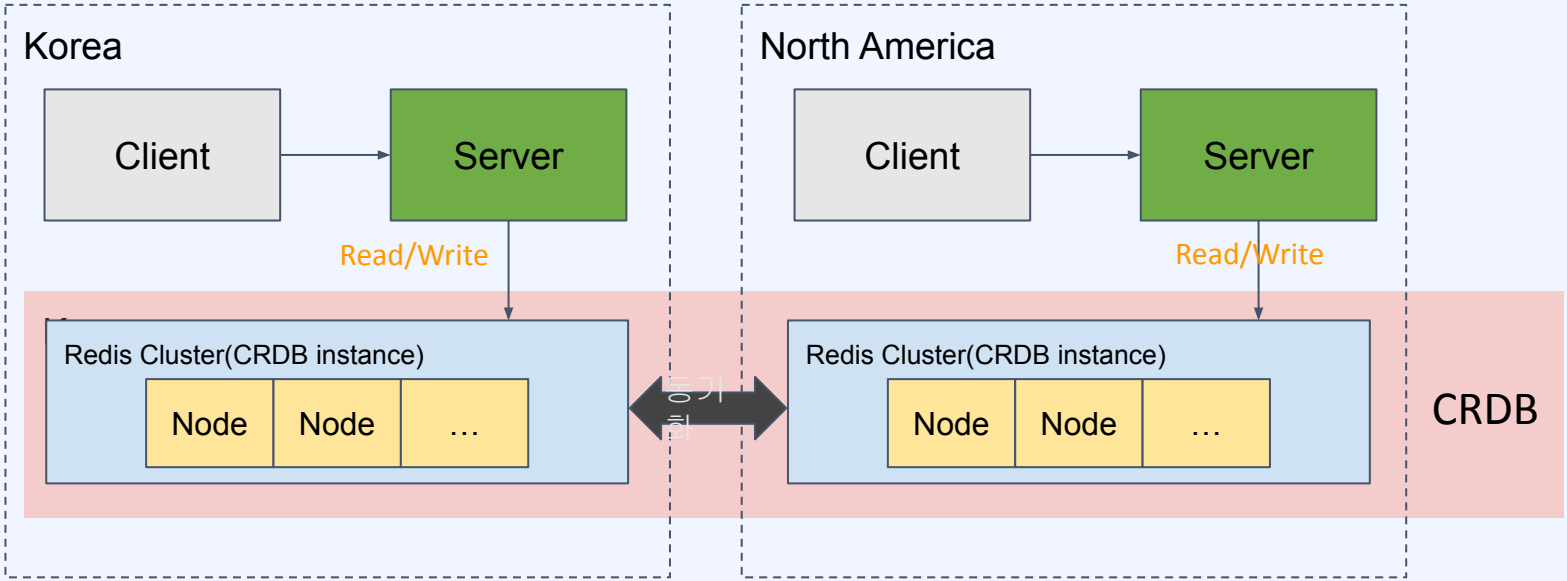
Redis Enterprise

- Enterprise급 기능을 제공하는 유료 제품
- Redis Labs에 의해 제공됨
- on-premise와 cloud 환경 둘 다 지원
- 제한 없는 선형 확장성, 향상된 고가용성, 추가 보안 기능, 기술 지원 등의 이점이 있음
- **Active-Active 아키텍처를 지원**

Active-Active Architecture

- 지역적으로 분산된 글로벌 데이터베이스를 유지하면서, 여러 위치에서 동일한 데이터에 대한 읽기/쓰기를 허용
- multi-master 구조로 생각할 수 있음
- 지역적으로 빠른 latency를 확보하면서도 데이터 일관성을 유지하는 형태
- 학술적으로 입증된 CRDT(Conflict-Free Replicated Data Types)를 활용해 자동으로 데이터 충돌을 해소
- 여러 클러스터에 연결되어 글로벌 데이터베이스를 이루는 것을 CRDB(Conflict-Free Replicated Database)라고 지칭

Active-Active Architecture



Active-Active Architecture의 장점

- 분산된 지역의 수와 상관없이 지역적으로 낮은 latency로 읽기/쓰기 작업을 수행
- CRDTs를 이용한 매끄러운 충돌 해결
- CRDB의 다수 인스턴스(지역 DB)에 장애가 발생하더라도 계속 운영 가능한 비즈니스 연속성 제공

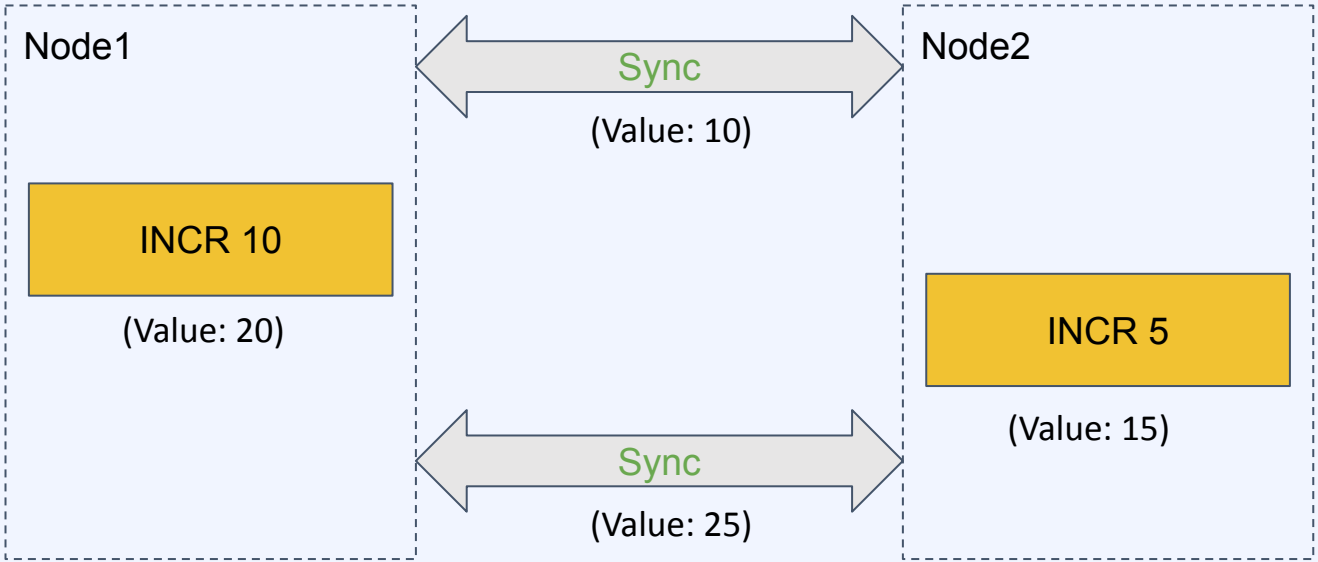
글로벌 서비스를 위한 Active-Active Architecture

2 데이터 충돌을 최소화하는 CRDTs

CRDT(Conflict-Free Replicated Data Type)란?

- 분산 환경에서 여러 노드들 간에 복제되는 데이터 구조로 아래 3개 특성을 가짐
  - 각 노드는 로컬에서 독립적으로 값을 업데이트할 수 있음
  - 노드간에 발생할 수 있는 데이터 충돌은 해당 데이터 타입에 맞는 알고리즘이 해결
  - 동일 데이터에 대해 노드들간에 일시적으로 다른 값을 가질 수 있지만 최종적으로는 같아짐
- 2011년에 등장했으며, 공유 문서 동시 편집 문제를 해결하려고 고안됨

CRDT의 동작



Redis의 충돌 해결

- 각 CRDB 인스턴스는 각자의 데이터셋에 vector clock을 유지함(vector clock: 데이터 일관성 관리를 위한 버전 정보)
- 동기화 요청이 왔을 때 해당 데이터의 vector clock을 비교해 old, new, concurrent로 분류함
- concurrent일 때는 아래처럼 충돌 해소 로직을 수행
  - CRDT인 경우에는 바로 해결 가능 (Ex: Counter)
  - non-CRDT인 경우 LWW(Last Write Wins)를 적용 (Ex: String)
  - 같은 데이터에 대해 서로 다른 Command가 충돌하면 미리 정의된 규칙에 따름

Redis의 충돌 해결 규칙 예제(Command 충돌 시)

- APPEND vs DEL: update 동작인 APPEND가 이김.
- EXPIRE vs PERSIST: 긴 TTL을 가지는 PERSIST가 이김.
- SADD vs SREM: 데이터 삭제보다 업데이트 동작인 SADD가 이김.

글로벌 서비스를 위한 Active-Active Architecture

3 Docker를 사용해 Active-Active 아키텍처 구성해보기

실습 절차

1) Docker 이미지 redislabs/redis를 이용해 3개의 노드를 띄우고 각각 클러스터를 생성
2) 3개의 노드를 네트워크로 연결
3) 3개의 클러스터를 묶어서 CRDB 생성
4) 데이터 쓰기 테스트
5) 네트워크 단절 후 데이터 쓰기
6) 네트워크 복구 후에 동기화 결과 확인