CP5. 복제

Part 2. 분산 데이터

▼ 개요

- Part 1 에서는 단일 장비에서 데이터를 저장할 때 적용하는 데이터 시스템 측면을 다뤘다.
  Part 2 에서는 저장소와 데이터 검색에 여러 장비가 관여하면 무슨일이 발생할까? 주제 중심으로 다룬다.
- 여러 장비 간 분산된 DB를 필요하는 이유는 여러가지다.
    - 확장성
      데이터 볼륨, 읽기 부하, 쓰기 부하가 단일 장비에서 다룰 수 있는 양보다 커지면 부하를 여러 장비로 분배할 수있다.
    - 내결함성/고가용성
      장비 하나(또는 여러 장비나 네트워크, 전체 데이터센터)가 죽더라도 애플리케이션이 계속 동작해야 한다면 여러 장비를 사용해 중복성을 제공할 수 있다. 장비 하나가 실패하면 다른 하나가 이어 받는다. (fail over)
    - 지연시간
      전 세계 사용자가 있다면 사용자와 지리적으로 가까운 곳의 데이터센터에서 서비스를 제공하기 위해 전 세계 다양한 곳에 서버를 두고 싶을 것이다. 이를 통해 사용자는 네트워크 패킷이 지구를 반 바퀴 돌아서 올 때까지 기다릴 필요 없다.
- **고부하로 확장**
    - **공유 메모리 아키텍쳐**
        - 고부하 확장이 필요하다면 더 강한 장비를 구매하는게 가장 단순하다(수직 확장, 용량 확장) 많은 CPU, 메모리, 디스크를 하나의 운영체제로 함께 결합할 수 있다. 그래서 빠른 상호 연결로 모든 CPU 가 메모리나 디스크의 모든 부분에 접근 할 수 있다.
        - 공유 메모리 아키텍처에는 모든 구성 요소를 단일 장비처럼 다룰 수 있다.
        - 문제점은 비용이 선형적인 추세보다 훨씬 빠르게 증가한다.
          시스템 성능이 두 배를 내기 위해서는 비용이 두 배 이상이 소요된다.
          또한 병목 현상 때문에 두 배 크기의 장비가 반드시 두 배의 부하를 처리할 수 있는 것은 아니다.
        - 공유 메모리 아키텍처는 제한적인 내결함성을 제공한다. (장비를 중단 시키지 않고 스케일 업 할 수 있다) 하지만 완전히 하나의 지리적 위치로 제한된다.
    - **공유 디스크 아키텍쳐**
        - 공유 메모리 아키텍쳐와는 다른 접근 방식이다. 독립적인 CPU와 RAM 을 탑재한 여러 장비를 사용하지만 데이터 저장은 장비 간 공유하는 디스크 배열을 한다.

        - 여러 장비는 고속 네트워크로 연결된다. 일부 데이터 웨어하우스 작업부하이 이 아키텍쳐를 사용하지만 잠금 경합과 오버헤드가 공유 디스크 접근 방식의 확장성을 제한한다.
    - **비공유 아키텍쳐 (수평 확장, 규모 확장, 스케일 아웃)**
        - DB 소프트웨어를 수행하는 각 장비나 가상 장비를 **노드**라고 부른다.
          각 노드는 CPU, RAM, 디스크를 독립적으로 사용한다.
          노드 간 코디네이션은 일반적인 네트워크를 사용해 소프트웨어 수준에서 수행한다.
        - 비공유 시스템은 특별한 하드웨어를 필요하지 않아 가격 대비 성능이 가장 좋은 시스템을 사용할 수 있다. 잠재적으로 지리적인 영역에 걸쳐 데이터를 분산해 사용자 지연 시간을 줄이고 전체 데이터센터의 손실을 줄일 수 있다.
        - Part 2 에서는 비공유 아키텍쳐에 중점을 둔다.
          비공유 아키텍쳐를 사용시 애플리케이션 개발자가 반드시 주의해야 하는 점이 있기 때문,
          데이터를 여러 노드에 분산하려면 분산 시스템에서 발생하는 제약 조건과 트레이드오프를 알고 있어야 한다. DB 스스로 이런 점을 숨길 수 없다
        - 대개 장점이 많지만, 애플리케이션 복잡도를 야기하고 때로는 데이터 모델의 표현을 제한한다. 경우에 따라 간단한 단일 스레드 프로그램이 100개 이상의 CPU 코어를 ㅅ용하는 클러스터 보다 효율적일 수 있다. 하지만 비공유 시스템은 매우 강력하다.
    - **복제 대 파티셔닝**
        - 여러 노드에 데이터를 분산하는 방법은 일반적으로 두 개다.
            - 복제
                - 같은 데이터 복사본을 잠재적으로 다른 위치에 있는 여러 노드에 유지한다.
                - 복제는 중복성을 제공한다. 일부 노드가 사용 불가능한 상태라면 해당 데이터는 남은 다른 노드를 통해 여전히 제공될 수 있다. 복제는 성능 향상에도 도움된다.
            - 파티셔닝
                - 큰 DB를 파티션이라는 작은 서브셋으로 나누고 파티션은 각기 다른 노드에 할당한다. (**샤딩**)
        - 복제와 파티셔닝은 다른 매커니즘이지만, 서로 관련있다.
            - 파티셔닝과 복제를 같이 사용해 분산 시스템에서 필요한 어려운 트레이드오프(트랜잭션, ACID)를 설명할 수 있다.
            - 트랜잭션을 이해하면 데이터 시스템에 발생하는 많은 문제를 설명하는 데 도움을 준다.
            - 이후 장에서 복잡한 애플리케이션의 요구사항을 만족하기 위해 어떻게 다양한 (분산된) 데이터 저장소를 가져와 대규모 시스템을 통할할 수 있는지 설명한다.

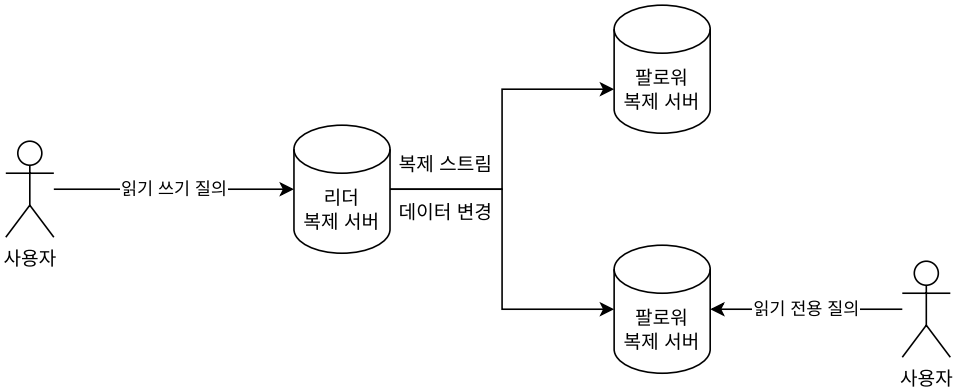
5. 복제
▼ 개요

- 복제
    - 복제란 네트워크로 연결된 여러 장비에 동일한 데이터의 복사본을 유지한다는 의미.
    - 복제가 필요한 이유
        - 지리적으로 사용자와 가깝게 데이터를 유지해 지연 시간 감소
        - 시스템 일부 장애 발생하더라도 지속적 동작 가능 (HA)

        - 읽기 질의를 제공하는 장비의 수를 확장해 읽기 처리량 증가
    - 복제 중인 데이터가 시간이 지나도 변경되지 않는다면 복제는 쉽다. 한번에 모든 노드에 데이터를 복사하며 된다. 복제의 어려움은 복제된 데이터의 변경 처리이다.
    - 노드 간 변경을 복제하기 위한 세가지 복제 알고리즘
        - 단일 리더 (single-leader)
        - 다중 리더 (multi-leader)
        - 리더 없는 (learderless)
    - 복제시 고려해야 할 많은 트레이드오프가 존재함.
        - 동기식 복제, 비동기식 복제
        - 잘못된 복제본의 처리
    - 분산 DB 에 대한 내용
        - 최종적 일관성
        - 쓰기 읽기 보장
        - 단조 읽기 보장

▼ 리더와 팔로워

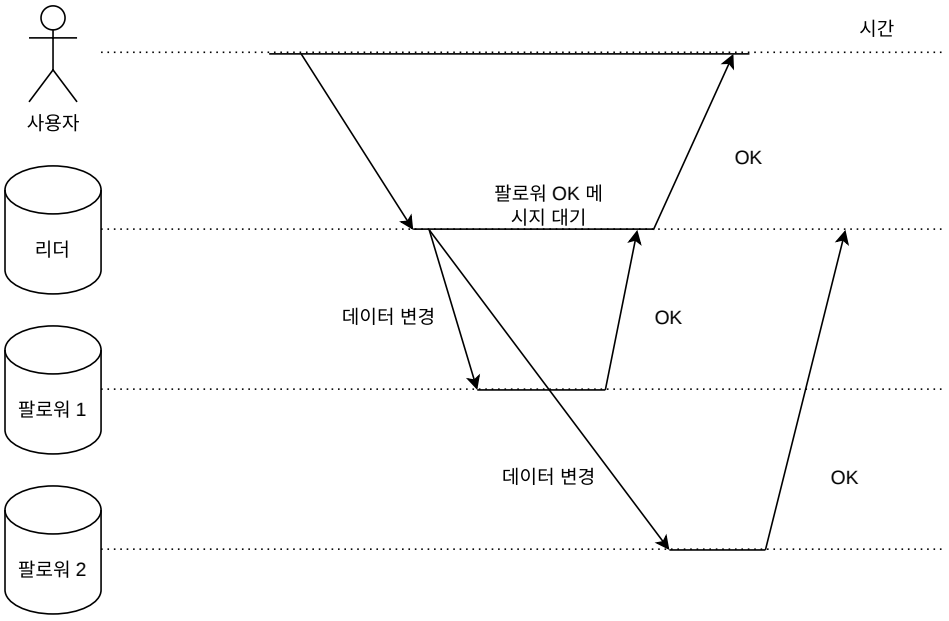
- **복제 서버(replica)** (DB 복사본)
    - 모든 복제 서버에 모든 데이터가 있다는 사실을 어떻게 보장할까?
        - DB의 모든 쓰기는 모든 복제 서버에서 처리되어야 한다. (그렇지 않으면 복제 서버는 더 이상 동일한 데이터를 유지할 수 없음)
        - 일반적인 해결책은 **리더 기반 복제 (leader-based replication) (능동/수동 복제) (마스터 슬레이브 복제)**
- **리더 기반 복제 (leader-based replication) (능동/수동 복제) (마스터 슬레이브 복제)**



    - 복제 서버 중 하나를 **리더(leader)(master, primary)** 로 지정.
    - 다른 복제 서버 **팔로워 (follower)(읽기 복제 서버)(슬레이브, secondary, hot standby)**
    - 쓰기 (클라이언트의 쓰기는 반드시 리더만 허용된다.)
        - 클라이언트가 DB에 쓰기를 할 때 클라이언트는 요청을 리더에게 보내야 한다.
        - 리더는 먼저 로컬 저장소에 새로운 데이터를 기록한다.

        - 리더가 로컬 저장소에 새로운 데이터를 기록할 때마다 데이터 변경을 **복제 로그**, **변경 스트림**의 일부로 팔로워에게 전송한다.
        - 각 팔로워가 리더로부터 로그를 받으면 리더가 처리한 것과 동일한 순서로 모든 쓰기를 적용해 그에 맞게 DB 의 로컬 복사본을 갱신한다.
    - 읽기
        - 클라이언트가 DB로 읽기를 할 때 리더 or 임의 팔로워에게 질의 가능하다.

- **동기식 대 비동기식 복제**



    - 동기식 (팔로워 1의 복제)
        - 리더는 팔로워 1이 쓰기를 수신했는지 확인해 줄 때까지 대기한다.
          확인이 끝나면 사용자에게 성공 응답을 해주고 다른 클라이언트에게 해당 쓰기를 보여준다.
        - 장점
            - 리더와 일관성 있게 최신 데이터 복사본을 가지는 것을 보장
            - 리더가 동작하지 않아도 데이터는 팔로워에게 계속 사용할 수 있음을 보장
        - 단점
            - 동기 팔로워가 응답하지 않으면 쓰기 처리 불가능
            - 리더는 모든 쓰기를 차단하고 동기 복제 서버가 다시 사용할 수 있을 때까지 대기됨.
            - 위 이유로 모든 팔로워가 동기식 방식은 비현실적이다.
              그래서
              반동기식을 사용한다. (하나의 팔로워는 동기식 나머지는 비동기식)
    - 비동기식 (팔로워 2의 복제)
        - 리더는 메시지를 전송하지만 팔로워의 응답을 기다리지 않음.
    - DB 복제 시간은 보장할 수 없다.
        - 장애를 극복 중

        - 시스템 최대 가용량 근처에서 동작
        - 노드 간 네트워크 문제 등등..
- **새로운 팔로워 설정**
    - 복제 서버 수를 늘리거나, 장애 노드 대체를 위함
      새로운 팔로워가 리더의 데이터 복제본을 정확히 가지고 있는지 어떻게 보장함?
        - 한 노드에서 다른 노드로 데이터 파일을 복사하는 것만으로는 대개 충분하지 않다.
        - 클라이언트는 지속적으로 DB에 기록하고 데이터는 항상 유동적이기 때문에 표준 파일 복사본은 다른 시점에 DB의 다른 부분을 보게 된다. 즉, 복사 결과가 유효하지 않을 수 있다.
        - DB를 lock 해서 디스크 파일을 일관성 있게 만들 수 있지만, 고가용성 목표에 부합하지 못한다. 다행히 팔로워 설정은 대개 중단시간 없이 수행할 수 있다.
    - 새로운 팔로워 추가 과정
        1. 가능하다면 전체 DB를 잠그지 않고 리더 DB의 스냅숏을 일정 시점에 가져온다. 대부분의 DB는 백업이 필요하기 때문에 이 기능이 있다.
        2. 스냅숏을 새로운 팔로워 노드에 복사한다.
        3. 팔로워는 리더에 연결해 스냅숏 이후 발생한 모든 데이터 변경을 요청한다.이것은 스냅숏이 리더의 복제 로그의 정확한 위치와 연관돼야 한다.
        4. 팔로워가 스냅숏 이후 데이터 변경의 미처리분을 모두 처리했을 때 따라잡았다고 한다.
- **노드 중단 처리**
    - 시스템의 모든 노드는 장애로 인해 중단될 수 있지만 계획된 유지보수로 인해 중단될 수도 있다.
    - 중단시간 없이 개별 노드를 재부팅할 수 있다는 점은 운영과 유지보수에 큰 장점이다.
    - 따라서 개별 노드의 장애에도 전체 시스템이 동작하게끔 유도하고 노드 중단의 영향을 최소화하는 것이 목표다.
    - **팔로워 장애: 따라잡기 복구**
        - 각 팔로워는 리더로부터 수신한 데이터 변경 로그를 로컬 디스크에 보관한다.
        - 팔로워가 죽어 재시작하거나 리더와 팔로워 사이의 네트워크가 일시적으로 중단된다면 팔로워는 매우 쉽게 복구할 수 있다.
        1. 먼저 보관된 로그에서 결함이 발생하기 전에 처리한 마지막 트랜잭션을 알아낸다.
        2. 그러면 팔로워는 리더에 연결해 팔로워 연결이 끊어진 동안 발생한 데이터 변경을 모두 요청할 수 있다.
        3. 이 변경이 다 적용되면 리더를 따라잡게 되고 이전과 같이 데이터 변경의 스트림을 계속 받을 수 있다.
    - **리더 장애: 장애 복구(failover)**
        - 리더의 장애를 처리하는 것은 까다롭다.
          팔로워 중 하나를 새로운 리더로 승격해야 하고 클라이언트는 새로운 리더로 쓰기를 전송하기 위해 재설정이 필요하며 다른 팔로워는 새로운 리더로 부터 데이터 변경을 소비하기 시작해야 한다. 이 과정을
          장애 복구(failover) 라 한다.
        - 장애 복구는 수동 이나 자동으로 진행된다.

- 자동 장애 복구 과정
  1. 리더가 장애인지 판단
     - 고장, 정전, 네트워크 문제 등 잠재적으로 여러 가지가 문제 일 수 있다.
     - 무엇이 잘못됐는지 발견할 수 있는 확실한 방법이 없기 때문에 대부분의 시스템은 단순히 타임아웃을 사용한다.
     - 노드들은 자주 서로 메시지를 주고 받으며 일정 시간 동안 노드를 응답하지 않으면(타임아웃) 죽은 것으로 간주한다.
  2. 새로운 리더 선택
     - 산출 과정(리더가 나머지 복제 서버의 대다수에 의해 선택) 통해 되거나 이전에 선출된 제어 노드에 의해 새로운 리더가 임명될 수 있다.
     - 가장 적합한 후보는 보통 이전 리더의 최신 데이터 변경사항을 가진 복제 서버다
  3. 새로운 리더 사용을 위한 시스템 재설정
     - 클라이언트는 이제 새로운 쓰기 요청을 새로운 리더에게 보내야 한다
     - 이전 리더가 돌아오면 여전히 자신이 리더라 믿을 수 있어야 하고 다른 복제 서버들이 자신을 리더에서 물러나게 한 것을 알지 못한다.
     - 시스템은 이전 리더가 팔로워가 되고 새로운 리더를 인식할 수 있게끔 해야한다.
- 복구 과정에서 잘못될 수 있는 과정 (이 문제들에 대한 쉬운 해결책은 없다)
  - 비동기식 복제를 사용한다면 새로운 리더는 이전 리더가 실패하기 전에 이전 쓰기 일부를 수신하지 못할 수 있다. 새로운 리더가 선출된 다음 이전 리더가 클러스터에 다시 추가된다면 이 쓰기를 어떻게 해야 할까? 그 동안 새로운 리더가 충돌하는 쓰기를 수신했을지도 모른다. 가장 일반적인 해결책은 이전 리더의 복제되지 않은 쓰기를 단순히 폐기하는 방법이다. (내구성을 기대할 수 없다)
    - 쓰기를 폐기하는 방법은 DB 외부의 다른 저장소 시스템이 DB 내용에 맞춰 조정돼야 한다면 특히 위험하다.
      ex)
      깃허브 유요하지 않은 마이 SQL 팔로워 승격된 사례
  - **스플릿 브레인(split brain)**
    - 특정 결함 시나리오에서 두 노드가 모두 자신이 리더라고 믿을 수 있다.
    - 매우 위험한 상황이고 두 리더가 쓰기를 받으면서 충돌을 해소하는 과정을 거치지 않으면 데이터가 유실되거나 오염된다. 일부 시스템에서는 안전 장치로 두 리더가 감지되면 한 노드를 종료하는 매커니즘이 있다. (잘 못하면 두 리더 모두 종료 될 수도 있다.)
  - 리더가 분명히 죽었다고 판단 가능한 적절한 타임아웃은 얼마일까?
    - 긴 타임아웃은 리더가 작동하지 않을 때부터 복구까지 오랜 시간이 소요된다는 뜻이다.
    - 하지만 타임아웃이 너무 짧으면 불필요한 장애 복구가 있을 수 있다.
      - ex) 일시적인 부하 급증으로 노드 응답 시간이 타임아웃보다 커지거나 네트쿼으 고장으로 패킷이 지연되는 경우
  - 노드 장애, 불안정한 네트워크, 복제 서버 일관성과 관련된 트레이드오프, 지속성, 가용성, 지연 시간 등의 문제는 사실 분산 시스템에서 발생하는 근본적인 문제다.