대용량 시스템 설계 기초

6.분산 키 - 값 저장소

OolongTea620
김혜린

2024.03.24

들어가기 전에

이런 내용이지 않을 까...?

👉대용량 시스템에서 사용되는 서버 캐시 입장에서
비관계형 데이터베이스 저장 전략을 이야기 하는 것 같다.

들어가기 전에

우리가 아는 대표적인 Key-Value 데이터베이스 들...

Amazon DynamoDB, memcachedm redis...

책에서는 무엇을 해결하고 싶은데?

- 키-값 쌍의 크기는 10K이하
- 큰 데이터를 저장할 수 있어야 한다. (분산 파일 저장을 이야기 하는 듯)
- 높은 가용성을 제공해야하 한다, (서버가 장애가 생기더라도 빨리 응답할 수 있어야 한다)
- 높은 규모 확장성을 제공해야한다(트래픽 양에 따라서 서버 증설/삭제가 이루어져야 한다)
- 데이터 일정 수준은 조정이 가능 해야 한다.
- 응답 지연시간(latency)이 짧아야 한다

👉 많은 데이터를 저장하기 위해서 분산 키-값 저장소가 필요

분산 키-값 저장소

분산 키-값 저장소는 분산 해시 테이블이라고 불린다.
키-값쌍 데이터를 여러 서버에 분산 저장한다.

CAP 정리

데이터 일관성(consistency), 가용성(availability), 파티션 감내(partion tolerance)
세가지 요구 사항을 만족하는 분산 시스템을 설계하는 것은 불가능하다

CAP 정리

데이터 일관성(C) :
분산 시스템에 접속하는 모든 클라이언트는 어떤 노드에 접속했느냐에 관계없이 언제나 같은 데이터를 보게 되어야 한다.

가용성(A) :
분산 시스템에 접속하는 클라이언트는 일부 노드에 장애가 발생 하더라도 항상 응답을 받을 수 있어야 한다.

파티션 감내(P) :
(파티션은 두 노드 사이에 통신 장애가 발생하였음을 의미한다)
파티션 감내는 네트워크에 파티션이 생기더라도 시스템은 계속 동작하여 한다는 것을 의미한다.

👉 셋 중 하나는 반드시 포기.

CAP에 따른 저장소 유형 분류

CP 시스템 :
일관성과 파티션 감내를 지원하는 키-값 저장소|

AP 시스템 :
가용성과 파티션 감내를 지원하는 키-값 저장소|

CA 시스템 :
일관성과 가용성을 지원하는 키-값 저장소
하지만, 분산 시스템은 반드시 파티션 문제를 감당할 수 있도록 설계되어야 하므로, CA시스템은 존재하지 않는다

사례1. 세 대의 복제 노드 n1,n2,n3

이상적 상태

- 네트워크가 파티션되는 상황은 절대로 일어나지 않음
- n1에 기록된 데이터는 n2, n3에 복제된다(데이터 일관성, 데이터 가용성 만족)

사례1. 세 대의 복제 노드 n1,n2,n3

실세계 분산 시스템

n3에 문제 발생
- n1,n2에 기록된 데이터는 n3에 전달되지 않는다.
- n3에 기록되었으나, n1,n2로 전달되지 못한 데이터가 있을 수 있다.

CP시스템(일관성)선택 시
- 세 서버 사이에 생길 수 있는 데이터 불일치 문제를 해소하기 위해서 n1,n2 서버의 쓰기 연산 중단
- 가용성을 결론적으로 포기하게 된다.

AP(가용성)선택 시
- 낡은 데이터를 반환할 위험이 있더라도, 쓰기 연산을 허용해야한다.
- n1,n2는 쓰기연산을 허용할 것, 파티션 문제가 일어난 뒤, 새 데이터를 n3에 전송할 것

결론

시스템 성격에 맞게 분산 시스템 방식을 선택할 수 있어야 한다.

시스템 컴포넌트

키-값 저장소 구현에 사용되는 핵심 컴포넌트 및 기술

- 데이터 파티션
- 데이터 다중화(replication)
- 일관성(consistency)
- 일관성 불일치 해소(inconsistency resolution)
- 장애 처리
- 시스템 아키텍처 다이어그램
- 쓰기 경로(write path)
- 읽기 경로(read path)

시스템 컴포넌트-데이터 파티션

대규모 어플리케이션의 경우, 데이터를 작은 파티션들로 분할하고, 여러 대 서버에 저장한다.

고려할 상황 :
- 데이터를 여러 서버에 고르게 분산할 수 있을까?
- 노드가 추가되거나, 삭제될 때, 데이터의 이동을 최소화 할 수 있을까?

안정 해시(consistent hash)도입 장점 :

- 규모 확장 자동화(automatic scaling):
  - 시스템 부하에 따라 서버가 자동으로 추가되거나 삭제되도록 만들 수 있다.

- 다양성(heterogeneity):
  - 각 서버의 용량에 맞게 가상 노드(virtual node)의 수를 조정할 수 있다.
  - 고성능 서버는 더 많은 가상 노드를 갖도록 설정 할 수 있다.

시스템 컴포넌트-데이터 다중화(replication)

높은 가용성과 안정성을 확보하기 위해서 데이터를 N개 서버에 비동기적으로 다중화할 필요가 있다.

N: 튜닝 가능한 값

- 어떤 키를 해시 링 위에 배치한 후, 그 지점으로 부터 시계방향으로 링을 순회하면서 첫 N개 서버에 데이터 사본을 보관한다.
- 가상 노드 사용 시, N개의 노드가 대응될 실제 물리 서버의 개수가 N보다 작아질 수 있다.
- 노드를 선택할 때, 값은 물리 서버를 중복선택하지 말아야 한다.

시스템 컴포넌트- 데이터 센터

같은 데이터센터에 속한 노드는 정전, 네트워크, 자연 재해 등을 동시에 겪을 가능성이 있다.
안정성을 담보하기 위해서 데이터의 사본은 다른 센터의 서버에 보관되고 센터들은 고속 네트워크로 연결한다.

시스템 컴포넌트- 일관성(consistency)

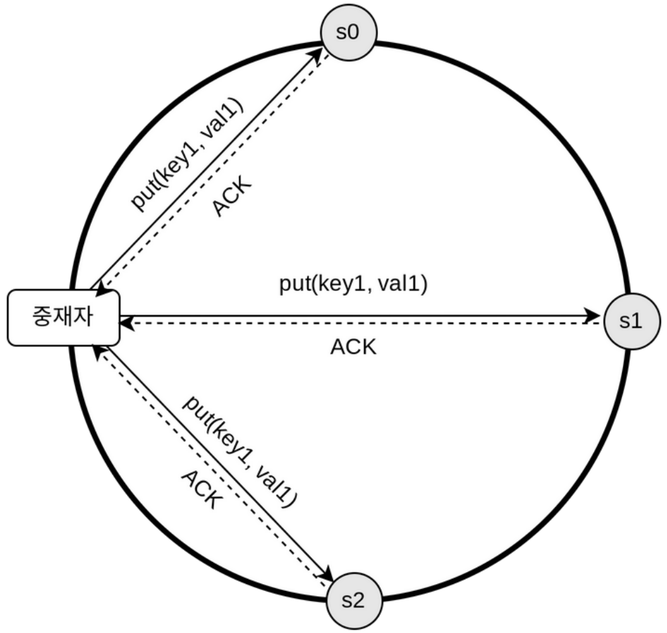
정족수 합의 프로토콜:
데이터 일관성을 보장해주기 위한 노드간 동기화 프로토콜

정의
N : 사본 개수
W : 쓰기 연산에 대한 정속수

- **적어도 W개의 서버로부터 쓰기 연산이 성공했다는 응답이 있어야한다**

R : 일각 연산에 대한 정족수

- **적어도 R개의 서버로부터 응답을 받아야 한다**



예시시
N = 1
쓰기 연산이 성공했다고 판단하기 위해서 중재자는 최소 한대 서버로부터 응답을 받아야 한다.
❗중재자는 클라이언트와 노드 사이의 프록시(Proxy)역할을 한다.

시스템 컴포넌트- 일관성(consistency)

W + R > N 인 경우
강한 일관성(Strong consistency):
강한 일관성 보장
일관성을 보장할 최신데이터를 가진 노드가 최소 1개는 겹친다.

R = 1, W = N:
빠른 읽기 연산에 최적화된 시스템

W = 1, R = N :
빠른 쓰기 연산에 최적화된 시스템

W + R <= N :
강한 일관성이 보장되지 않음

👉 요구되는 일관성 수준에 따라서 W, R, N을 바꾸면 된다.

시스템 컴포넌트- 일관성 모델 (consistency model)

키-값 저장소를 설계할 때, 고려해야 할 또 하나의 중요한 요소
일관성의 수준을 결정한다.

👉 요구되는 일관성 수준에 따라서 W, R, N을 바꾸면 된다.

시스템 컴포넌트- 일관성 모델 (consistency model)

강한 일관성(Strong consistency) : 모든 읽기 연산은 가장 최근에 갱신된 결과를 반환

- 클라이언트는 절대로 낡은 데이터를 알지 못한다.
- 고가용성 시스템에 적합하지 않다(새로운 요청의 처리가 중단되기 때문

모든 사본에 현재 쓰기 연산의 결과가 반영될 때 까지
해당 데이터에 대한 읽기/쓰기를 금지한다.

- 약한 일관성(Weak consistency) : 읽기 연산은 가장 최근에 가장 최근에 갱신된 결과를 반환하지 못할 수 있다.

- 최종 일관성(Eventual consistency) : 약한 일관성의 한 형태로, 갱신 결과가 결국에는 모든 사본에 반영(동기화)되는 모델이다.
  - 다이나모, 카산드라 저장소
  - 병렬적인 쓰기 연산이 일어나면, 일관성이 깨질 수 있다.
  - 해당 문제를 클라이언트가 해결해야 한다.
    - 클라이언트에서 데이터 버전 정보를 활용하여 일관성이 깨진 데이터를 읽지 않도록 해아한다.

시스템 컴포넌트- 비 일관성 해소 기법 : 데이터 버저닝

데이터 다중화 : 가용성은 높어지지만, 사본 간 일간성이 깨질 가능성이 올라간다.
데이터 비일관을 해소할 때, 버저닝(versioning), 벡터 시계(vector clock)을 이용 할 수 있다.

버저닝
데이터를 변경할 때 마다, 해당 데이터의 새로운 버전을 만드는 것.
각 버전의 데이터는 변경 불가능이다.

벡터 시계
[서버, 버전]의 순서쌍을 데이터에 매단 것.
어떤 버전이 선행 버전인지, 후행 버전인지, 아니면 다른 버전과 충돌이 있는지 판별하기 위해 사용한다.

시스템 컴포넌트- 비 일관성 해소 기법 : 데이터 버저닝 - 벡터시계

벡터 시계

[서버, 버전]의 순서쌍을 데이터에 매단 것.
어떤 버전이 선행 버전인지, 후행 버전인지, 아니면 다른 버전과 충돌이 있는지 판별하기 위해 사용한다.

장점

- 어떤 버전 X, 버전 Y 중 어떤 버전이 이전 버전인지 알 수 있다 (충돌이 있나 없나)

단점

- 클라이언트 구현 복잡해짐
  - 충돌감지 및 해소로직이 클라이언트에 구현되어 있어야 한다.
- 순서쌍 갯수로 인한 저장 용량 임계치의 한계
  - [서버:버전]의 순서쌍 갯수가 굉장히 빨리 늘어난다.
    - 효율성이 낮아 질 수 있다.

시스템 컴포넌트- 장애 처리

장애 감지(failure detection)
보통 두 대 이상의 서버에서 똑같은 장애 A가 보고 될 때, 실제로 해당 서버에 장애가 났다고 간주한다

솔루션 들
멀티 테스킹
- 서버가 많아질 수록 비효율적인 방법이 된다.

가십프로토콜(gossip protocol)
각 노드는 맴버십 목록을 유지한다.
 - 멤버십 목록은 각 멤버의 ID와 그 박동 카운터쌍의 목록이다.
- 각 노드는 주기적으로 자신의 박동 카운터를 증기시킨다.
- 각 노드는 무작위로 선정된 노드들에게 주기적으로 자기 박동 카운터 목록을 보낸다.
- 박동 카운터 목록을 받은 멤버는 멤버십 목록을 최신 값으로 갱신한다.
- 어떤 멤버의 박동 카운터 값이 지정된 시간 동안 갱신되지 않으면, 해당 멤버는 장애 상태인 것으로 간주.

시스템 컴포넌트- 장애 처리

장애 해소 전략들(failure resolution)

후임시위탁기법 : 일시적 장애 처리

- 네트워크, 서버 문제로 장애 상태인 서버로 가는 요청은 다른 서버가 잠시 맡아서 처리하는 것
- 그 상황에서 발생하는 변경사항은 해당 서버가 복구될 때 일관 반영하여 데이터 일관성 보존
- 임시 서버에 그에 대한 단서(hint)를 남겨둔다.

반-엔트로피 프롵토콜 구현 : 영구적인 노드의 장애 상태 처리 방법

- 사본들을 비교하여 최신 버전의 데이터로 갱신하는 과정이다.
- `머클트리(merkle)`:
  - 사본간의 일간성이 망가진 상태를 탐지하고 전송 데이터의 양을 줄이기 위한 방식

시스템 컴포넌트- 장애 처리

머클트리?

- 각 노드에 자식 노드들에 보관된 값의 해시(자식 노드가 종단인 경우)
- 자식 노드들의 레이블로부터 계산된 해시값을 레이블로 붙여두는 트리
- 대규모 자료구조의 내용을 효과적이면서도 보안상으로 안전한 방법으로 검증(verification)할 수 있다.

시스템 컴포넌트 - 시스템 아키텍처 다이어그램

- 클라이언트는 키-값 저장소가 제공하는 두 가지 단순한 API 즉, get(key) 및 put(key, value)와 통신한다
- 중재자(coordination)은 클라이언트에게 키-값 저장소에 대한 프록시(proxy)역할을 하는 노드이다.
- 노드는 안정 해시의 해시 링 위에 분포한다.
- 노드를 자동으로 추가 / 삭제할 수 있도록 시스템은 완전히 분산한다.
- 데이터는 여려 노드에 다중화한다.
- 모든 노드가 같은 책임을 진다. SPOF(Single Point Of Failure)는 존재하지 않는다.

시스템 컴포넌트 - 노드에 쓰기, 읽기가 요청 될 때

쓰기 경로(write path)

쓰기 요청이 특정 노드에 전달 된다면?

1. 쓰기 요청이 커밋 로그(commit log)파일에 기록된다.
2. 데이터가 메모리 캐시에 기록된다.
3. 메모리 캐시가 가득차거나, 사전에 정의한 어떤 임계치에 도달하면 데이터는 디스크에 있는 SSTable에 기록된다.

시스템 컴포넌트 - 노드에 쓰기, 읽기가 요청 될 때

읽기 경로(read path)

- **읽기 요청 발생 시, 노드는 데이터가 메모리 캐시에 있는지부터 확인한다.**
- **메모리 캐시에 데이터가 있는 경우, 해당 데이터를 메모리에서 가져와서 클라이언트에게 반환한다.**

- **메모리에 없는 경우, 디스크에서 가져온다.**
- **key가 존재하는 SSTable을 찾기 위해서 블룸 필터가 흔히 사용된다.**

읽기 요청이 특정 노드에 전달 된다면?

1. 데이터가 메모리에 있는지 검사한다 없으면 2번 부터 시작이다.
2. 데이터가 메모리에 없으므로, 블룸 필터를 검사한다.
3. 블룸 필터를 통해서 어떤 SSTable에 키가 보관되어 있는지 알아낸다.
4. SSTable에서 데이터를 가져온다.
5. 해당 데이터를 클라이언트에게 반환한다.

END