Redis Week5

Scaling

레디스 클러스터를 사용하면 다음과 기능을 사용할 수 있다.

- 여러 노드 간에 데이터 세트를 자동으로 분할
- 노드의 하위 집합에 장애가 발생하거나 나머지 클러스터와 통신 할 수 없어도 작업을 수행할 수 있다

Redis cluster TCP port

모든 레디스 클러스터 노드에는 두 개의 개방형 TCP 연결이 필요하다. 클라이언트에 서비스를 제공하는 데 사용되는 Redis TCP port(6379)와 cluster bus port로 알려진 두번째 포트이다. 기본적으로 클러스터 버스 포트는 데이터 포트에 10000를 더해서 설정된다.(16379)

클러스터 버스는 대역폭과 처리 시간이 적어 노드 간 정보 교환에 더 적합한 바이너리 프로토콜을 사용하는 노드 간 통신 채널이다. 노드는 장애 감지, 구성 업데이트, 장애 조치 권한 부여 등에 클러스터 버스를 사용한다. 클라이언트는 클러스터 버스 포트와 통신을 시도하지 말고 Redis 명령 포트를 사용해야 한다. 그러나 방화벽에서 두 포트를 모두 열어야 한다. 그렇지 않으면 Redis 클러스터 노드가 통신할 수 없다.

레디스 클러스터가 제대로 동작하려면 각 노드에 대해 다음이 필요하다.

- 클라이언트와 통신하는 데 사용되는 클라이언트 통신 포트(일반적으로 6379)는 클러스터에 도달해야 하는 모든 클라이언트와 키 마이그레이션을 위해 클라이언트 포트를 사용하는 다른 모든 클러스터 노드에 개방된다.
- 클러스터 버스 포트는 다른 모든 클러스터 노드에서 도달할 수 있어야 한다.

Cluster Data Sharding

레디스 클러스터는 일관된 해싱을 사용하지 않고 모든 키가 개념적으로 해시 슬롯이라고 부르는 것의 일부인 다른 형태의 샤딩을 사용한다.

레디스 클러스터에는 16384개의 해시 슬롯이 있으며 주어진 키에 대한 해시 슬롯을 계산하기 위해 다음과 같이 계산하면 된다.

```
HASH_SLOT = CRC16(key) mod 16384
```

레디스 클러스터의 모든 노드는 해시 슬롯의 하위 집합을 담당한다. 에를들어 3개의 노드가 있는 클러스터가 있다고 하면

- 노드 A에는 0에서 5500까지의 해시 슬롯이 있다.
- 노드 B에는 5501에서 11000까지의 해시 슬롯이 있다.
- 노드 C에는 11001에서 16383까지의 해시 슬롯이 있다.

이렇게 하면 클러스터 노드를 쉽게 추가하고 제거할 수 있다. 예를 들어 새 노드 D를 추가하려면 노드 A, B, C에서 D로 일부 해시 슬롯을 이동해야 한다. 마찬가지로 클러스터에서 노드 A를 제거하려면 해시 슬롯을 이동하면 된다. A에서 B와 C로 서비스를 제공한다. 노드 A가 비어 있으면 클러스터에서 완전히 제거할 수 있다.

한 노드에서 다른 노드로 해시 슬롯을 이동하는 데 작업을 중지할 필요가 없다. 따라서 노드를 추가 및 제거하거나 노드가 보유한 해시 슬롯의 비율을 변경하는 데 다운타임은 없다.

Redis 클러스터는 단일 명령 실행(또는 전체 트랜잭션 또는 Lua 스크립트 실행)에 관련된 모든 키가 동일한 해시 슬롯에 속하는 한 여러 키 작업을 지원한다. *사용자는 해시 태그* 라는 기능을 사용하여 여러 키를 동일한 해시 슬롯의 일부로 만들 수 있다 .

해시 태그는 Redis 클러스터 사양에 문서화되어 있지만 요점은 키의 {} 괄호 사이에 하위 문자열이 있는 경우 문자열 내부에 있는 것만 해시된다는 것이다. 예를 들어, 키 `user:{123}:profile`와 키 `user:{123}:account`는 동일한 해시 태그를 공유하기 때문에 동일한 해시 슬롯에 있음이 보장된다. 따라서 동일한 다중 키 조작으로 이 두 키를 조작할 수 있다.

master-replica model

Redis Cluster는 별도의 가용성을 지원하지는 않고 이전시간에 배웠던 Redis Replication의 개념을 Cluster에도 사용할 수 있게 한다. 만약 클러스터 내부 노드 A, B, C로 이루어진

상태에서 각각 레플리카로 A1, B1, C1을 구성하면 마스터노드인 B에 장애가 나더라도 복제본인 B1을 통해서 Redis Cluster를 계속적으로 사용 가능한 상태로 만든다. 그러나 만약 B와 B1이 둘다 장애가 난다면 Redis Cluster는 사용이 불가능한 상태가 된다.

Redis Cluster consistency guarantees

Redis Cluster는 강력한 일관성을 보장하지 않는데 다음과 같은 시나리오를 예를 들어보겠다.

1. **B는 B1, B2, B3라는 복제본을 갖고 있다.**
2. **사용자는 B에 데이터 aaa를 입력했다.**
3. **B와 B1, B2, B3 사이에 replication은 비동기로 이루어지기때문에 B1, B2, B3에 데이터 전파가 이루어지지 않더라도 사용자에게 'OK' 라는 메시지를 보낸다.**
4. **비동기로 데이터를 복제하는 도중 aaa 데이터가 어떤 환경에 의해 B1, B2, B3에 들어가지 않았다.**
5. **이후에 B의 장애가 발생했을때 B1, B2, B3에는 aaa 라는 데이터가 없다.**

따라서 이런 문제에 대해서는 주의해야 한다.

Redis persistence

persistence는 SSD(Solid-State Disk)와 같은 내구성 있는 스토리지에 데이터를 쓰는 것을 의미한다. Redis는 다양한 지속성 옵션을 제공한다. 여기에는 다음이 포함된다.

- **RDB** (Redis 데이터베이스): RDB persistence는 지정된 간격으로 데이터 세트의 특정 시점 스냅샷을 수행한다.
- **AOF**(Append Only File): AOF persistence는 서버에서 수신한 모든 쓰기 작업을 기록한다. 이러한 작업은 서버 시작 시 다시 재생되어 원본 데이터 세트를 재구성할 수 있다. 명령은 Redis 프로토콜 자체와 동일한 형식을 사용하여 기록된다.
- No persistence : persistence를 완전히 비활성화할 수 있습니다. 캐싱할 때 가끔 사용된다.
- **RDB + AOF** : 동일한 인스턴스에서 AOF와 RDB를 모두 결합할 수도 있다.

RDB

장점

- RDB는 레디스 데이터의 매우 작은 단일 파일의 특정 시점을 나타낸다. RDB 파일은 백업에 적합하다. 예를 들어 최근 24시간 동안 매시간 RDB 파일을 보관하고 30일 동안 매일 RDB 스냅샷을 저장할 수 있다. 이를 통해 재해 발생 시 다양한 버전의 데이터 세트를 쉽게 복원할 수 있다.
- RDB는 원거리 데이터 센터 또는 Amazon S3(암호화 가능)로 전송할 수 있는 단일 압축 파일이므로 재해 복구에 매우 유용하다.
- RDB는 Redis 상위 프로세스가 지속하기 위해 수행해야 하는 유일한 작업이 나머지 모든 작업을 수행할 자식을 포크하는 것이므로 Redis 성능을 최대화한다. 상위 프로세스는 디스크 I/O 등을 수행하지 않는다.
- RDB는 AOF에 비해 큰 데이터 세트로 더 빠르게 재시작할 수 있다.

단점

- Redis가 작동을 멈춘 경우(예: 정전 후) 데이터 손실 가능성을 최소화해야 하는 경우 RDB는 좋지 않다. RDB가 생성되는 다른 *저장 지점*을 구성할 수 있다 (예: 데이터 세트에 대해 최소 5분 및 100회 쓰기 후 여러 저장 지점이 있을 수 있음). 그러나 일반적으로 5분 이상마다 RDB 스냅샷을 생성하므로 Redis가 어떤 이유로든 올바른 종료 없이 작동을 중지하는 경우 최신 데이터를 잃을 수 있도록 준비해야 한다.
- RDB는 하위 프로세스를 사용하여 디스크에서 지속되기 위해 자주 fork()해야 한다. fork()는 데이터 세트가 큰 경우 시간이 많이 소요될 수 있으며 Redis가 몇 밀리초 또는 데이터 세트가 매우 크고 CPU 성능이 좋지 않은 경우 1초 동안 클라이언트 서비스를 중지할 수 있다. AOF는 또한 fork()가 필요하지만 덜 빈번하며 내구성에 대한 절충 없이 로그를 다시 작성하는 빈도를 조정할 수 있다.

AOF

장점

- AOF Redis를 사용하면 훨씬 더 내구성이 있다. 다른 fsync 정책을 가질 수 있다. fsync는 백그라운드 스레드를 사용하여 수행되며 기본 스레드는 fsync가 진행 중이지 않을 때

쓰기를 시도하므로 1초 분량의 쓰기만 손실될 수 있다.

- AOF 로그는 추가 전용 로그이므로 정전 시 검색이나 손상 문제가 없다. 어떤 이유로(디스크가 꽉 찼거나 다른 이유로) 로그가 절반만 작성된 명령으로 끝나더라도 redis-check-aof 도구를 사용하면 쉽게 수정할 수 있다.
- Redis는 AOF가 너무 커지면 백그라운드에서 자동으로 다시 작성할 수 있다. Redis가 이전 파일에 계속 추가하는 동안 현재 데이터 세트를 생성하는 데 필요한 최소한의 작업 세트로 완전히 새로운 파일이 생성되고 이 두 번째 파일이 준비되면 Redis가 두 파일을 전환하고 추가를 시작하므로 재작성은 완전히 안전하다.
- AOF에는 이해하기 쉽고 구문 분석하기 쉬운 형식으로 모든 작업의 로그가 하나씩 포함되어 있다. AOF 파일을 쉽게 내보낼 수도 있다. 예를들어 실수로 모든 항목을 플러시하더라도 그 동안 로그를 다시 쓰지 않는 한 서버를 중지하고 최신 명령을 제거한 다음 Redis를 다시 시작하기만 하면 데이터 세트를 계속 저장할 수 있다.

단점

- AOF 파일은 일반적으로 동일한 데이터 세트에 대한 동등한 RDB 파일보다 크다.
- AOF는 정확한 fsync 정책에 따라 RDB보다 느릴 수 있다. 일반적으로 fsync를 *매초*로 설정해도 성능은 여전히 매우 높으며, fsync를 비활성화하면 높은 부하에서도 정확히 RDB만큼 빨라야 한다. 여전히 RDB는 엄청난 쓰기 부하가 있는 경우에도 최대 대기 시간에 대해 더 많은 보장을 제공할 수 있다.