3.샤딩, 샤드 클러스터 구성

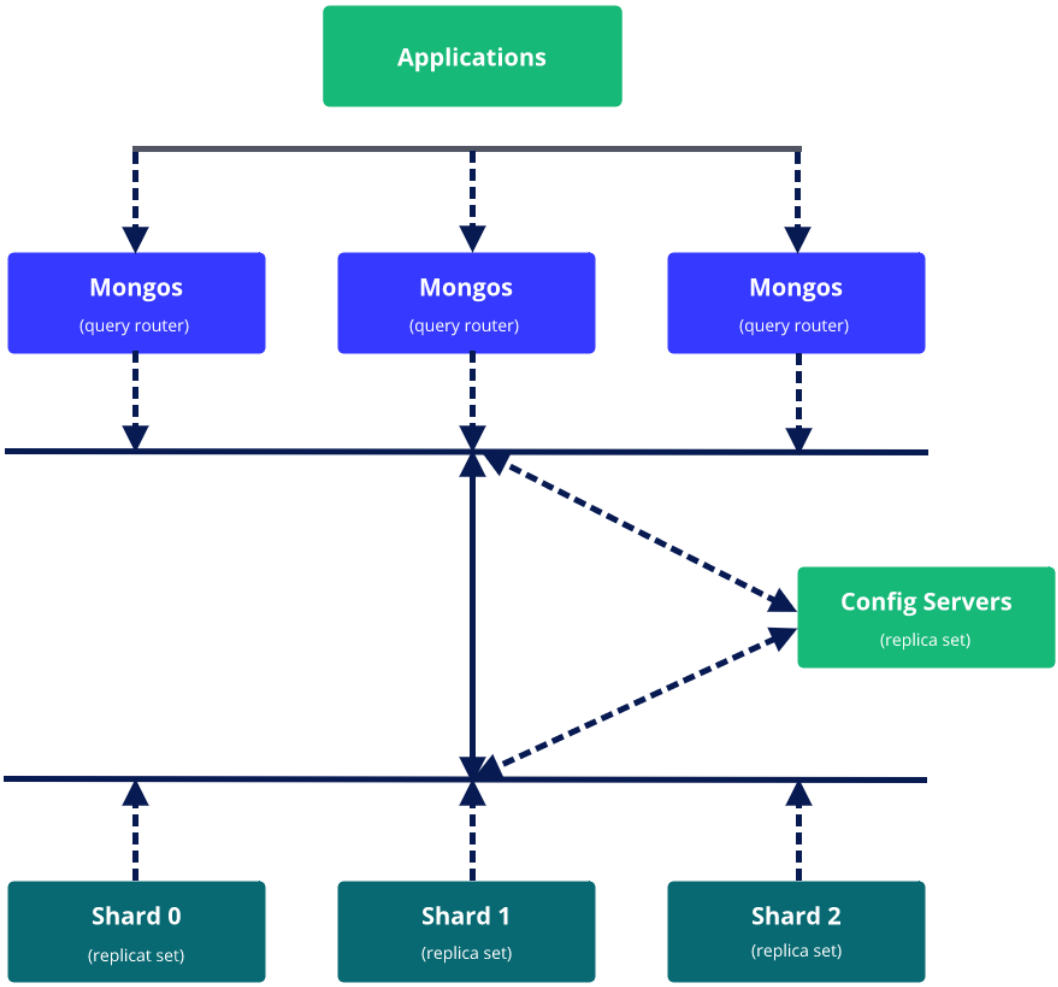
Sharding system이란

. 샤딩의 가장 큰 목적은 파티셔닝을 통한 데이터 분산 처리와 성능 향상을 위한 Load Balancing

. 빅데이터의 효율적인 관리와 백업 및 복구 전략 수립을 위한 솔루션

. Scale out의 개념

. Mongodb는 콜렉션 단에서 데이터를 샤딩해서 클러스터 샤드에 콜렉션 데이터를 분배



구성 요소

- Shard : 샤드 데이터(Shareded data)의 집합, 샤드는 replica set이 될 수 있다.
- Mongos : 쿼리 라우터 같은 역할, 애플리케이션과 샤드 클러스터간의 인터페이스 제공
- Config Servers : 설정 서버에서는 클러스터의 메타 데이터와 설정들을 저장

샤딩의 목적

1. 데이터의 분산저장
   . 초당 발생하는 빅데이터를 디스크에 저장할 때 발생하는 Write Scaling 문제는 애

플리케이션의 성능 저하 뿐만 아니라 샤딩 시스템 전체 성능 저하현상을 유발

2. 빠른 성능
   . 분산 처리는 여러 개의 프로세스가 여러 개의 CPU를 통해 동시 작업을 수행했을 때 가장 이상적
3. 데이터의 백업과 복구 전략의 일환
   . 하나의 서버에 데이터를 관리 했을 때 장애 발생 시 유실 되는 데이터 양이 매우 많아지고 복구 시에도 많은 시간과 비용이 요구 됨

샤딩 구축을 위한 적정 시스템 환경

- 샤딩 시스템은 데이터의 적절한 분산처리를 통한 효용성 향상이 가장 큰 목적이므로 3대 이상의 샤드 서버 구축을 권장(최소 2대 이상이지만 가장 적절한 성능이 보장되기 위한 적정 서버 수)
- 싱글 노드를 운영 할 때 보다 요구 되는 메모리 영역보다 최소 20 ~ 30%이상의 추가 매모리 영역이 요구(샤드 시스템 구축할 때 활성화 되는 MongoS와 OpLog 그리고 Balancer 프로세스가 사용하게 될 추가 메모리에 대한 고려 등)
- 샤드 시스템 구축할 때 요구되는 Config서버는 최소 3대 이상 활성화 할 것을 권장
- Config 서버는 샤드 시스템 구축과 관련된 메타 데이터를 저장 관리하며 빅데이터의 빠른 검색을 위한 인덱스 정보를 저장, 관리하고 있기 때문에 샤드 서버와는 별도 서버에 구축하는 것이 원칙(장애 발생 시 spare를 위해 1대 이상의 config 서버가 필요하며 샤드 서버보다 저사양 시스템으로 구축 가능)

Config 서버

- 3개의 mongod로 구성할 것을 권장(최소 1대)
- 2 Phase Commit을 수행
- 하나의 Config Server가 다운되면 나머지는 Read Only상태
- shard에 대한 Meta Data 정보를 가짐
- MongoDB 3.4버전 부터 Config Server는 복제서버(ReplicaSets)로 구성, 3.2이하 버전은 각각 StandAlone 서버로 구축 후 설정

Config 서버의 특징

- Config 서버는 샤딩 시스템의 필수 구조 중에 하나이고 최소 1대가 요구되며 예기치 못한 시스템 장애로 인해 서비스가 수행 되지 못하는 경우를 대비해서 추가로 Config 서버의 설정이 필요하여 하나의 시스템에 Config 서버를 설정하게 되면 시스템에 장애가 발생 할 때 문제가 될수 있으므로 별도 시스템 또는 샤드 서버 쪽에 분산 배치
- 1번 Config 서버에 저자오딘 Meta Data는 2번, 3번과 동일하게 관리
- 2대 중 하나의 Config 서버에 장애가 발생하면 나머지 서버는 읽기 전용이 되기 때문에 최소한의 샤딩 시스템운영이 가능
- Config 서버는 각 샤드 서버에 어떤 데이터들이 어떻게 분산 저장되어 있는지에 대한 Meta Data가 저장되어 있으며 MongoS가 데이터를 쓰고/읽기 작업을 수행할 때 Config 서버를 통해 수집

MongoS 프로세스 주요 특징

- 빅 데이터를 샤드 서버로 분배해 주는 프로세스
- 하나 이상의 프로세스가 활성화
- Application Server에서 실행 가능
- Config 서버로부터 Meta-Data를 캐시

. Config 서버가 설치 된 Node에서 각 Node가 상호 연결 될 수 있도록 mongos에 등록해야함

. MongoDB의 샤딩은 Collection 단위로 수행되며 해당 Collection의 특정 필드 값을 기준으로 분산됨

. 샤딩을 위해 샤드키에는 반드시 인덱스의 생성이 요구

Sharding 시스템 구축 시 고려 사항

- Shard Key
  1. Shard key를 이용하여 컬렉션의 document를 샤드에 배포
  2. 별도로 지정하지 않으면 샤드 키는 Object_id(_id)가 되며, 해당 컬렉션의 모든 document에 존재하는 field index라면 샤드키로 지정 가능
  3. 하나 이상의 Field로 샤드 키를 구성할 수 없다.(최대 512byte를 초과할 수 없다.)
  4. 샤딩한 컬렉션의 샤드 키는 변경할 수 없다.
  5. 샤드 키로 지정된 필드의 value는 변경 될 수 ㅇ벗다.

6. 샤드 키는 Partition과 Load Balancing의 기준이 되므로 MongoDB의 데이터 저장과 성능에 절대적 영향을 미친다.
7. 샤드 키를 설정하는것은 Cardinality, 데이터의 분산 저장 , Query Isolation, 신뢰도, Index Locality등을 고려하여 결정

- Shard Key Index
  1. 데이터를 분산하는 기준, Chunk Migration 횟수와 빈도를 결정, 데이터의 빠른 검색을 윟나 기준
  2. 샤드 키는 반드시 하나의 인덱스를 생성해야 하는데 이것은 여러대의 서버에 데이터가 Chunk 단위로 분산 저장되는 경우 데이터의 빠른 검색을 위해 필요
  3. 인덱스를 기준으로 설명 시 Shard Key는 인덱스의 생성을 고려할 수 있는 필드, 빅 데이터의 빠른 검색이 요구되는 필드 즉 좋은 카디널리티를 가진 필드를 Shard Key Index로 선정해야 한다.

- Hash Index에 대한 고려
  1. Hashed Index는 MongoDB V2.4에서 제공
  2. 사용자가 정의한 샤드 키가 적절하게 분산 되지 않는 문제 해결을 위해 Mongodb가 제공하는 해시 알고리즘을 적용
  3. 인덱스 대상 필드 값이 해시 값으로 변환 되어 저장되기 때문에 불필요한 변환이 발생함으로 사용자가 해당 기술에 대한 정확한 이해를 바탕으로 사용하되 적절한 샤드 키가 없거나 기술에 대한 이해가 부족한 경우 적용하는 것을 권장
  4. Compound Index, Unique Index, Multi-Key Index에는 적용 할 수 없음
  5. 연속적으로 저장된 범위의 값을 해시 인덱스로 생성하게 되면 암호화된 값으로 변환 되어 저장됨으로써 인접하지 않은 저장 공간에 저장되어 연속된 값을 검색 시 IO가 증가할 가능성이 있어 성능 지연 문제 발생
  6. 부분 데이터 검색 시에는 여러대의 샤드 서버로 부터 부분 데이터를 read, Mongos가 수집 후 데이터를 Return하므로 불필요한 IO가 발생하나 특정 field라면 특정 샤드 서버만 데이터를 검색하여 불필요한 IO를 줄일 수 있음(full scan인 경우에는 다를 수 있음)

- Chunk Size & Migraion 임계 값
  1. MongoS는 각 Shard의 Balance에 불균형(20개 이상의 Chunk 개수)이 발생하면 Chunk의 Migration 작업이 발생
  2. Migration이 발생 할 때 Mongos에 Lock 발생

※ Chunk : 하나의 서버에만 데어터가 저장될 시 성능 저하 현상이 발생하기 때문에 하나의 서버에 저장되는 데이터들을 여러개의 논리적 구조로 분할 저장 했다가 일정한 데이터 양에 도돨 했을때 다른 서버로 분한 데이터 일부를 이동 저장 하는데 해당 단위를 Chunk 라고 함. 기본적으로 64MB로 분할 되며 어떤 필드를 샤드키로 설정 했느냐에 따라 64MB보다 큰 Size로 분할 될 수 있음.

해당 크기에 따라 Migraion 횟수와 빈도가 결정 되기 때문에 데이터 양과 도큐멘트 크기에 따라 적절한 size를 결정 해야함. Migration이 빈번하다면 Chunk Size를 늘리고 데이터가 집중 되고 골고루 분산되지 않는 경우 Size를 작게 설정해야함

Chunk Migration 임계 값

Aa 이름	Chunk 개수	Chunk Migration 수
제목 없음	1개 이상 20개 미만	2
제목 없음	21개 이상 80개 미만	4
제목 없음	80개 이상	8

- 샤드-키 값의 변경
  1. MongoDB 4.2버전 부터 Shard Key Value의 Update 지원
  2. _id 필드가 샤드키로 선정 된 경우를 제외한 모든 경우 사용가능
  3. 샤드 키 값의 Update는 반드시 mongos에서 하나의 트랜잭션 또는 Retryablewrite기능을 통해 실행
  4. 해당 메소드를 통해 변경 가능
     . db.collection.updateOnd / db.collection.update ← {multi : false} 옵션절 사용
     . db.collection.findAndModify / db.collection.findOneAndUpdate
     . db.collection.bulkWrite
  5. Shard Server에서 직접적인 Update작업을 해서는 안됨

Shard 서버 추가와 삭제

```
1. Shard Server의 추가
>sh.addShard("<hostname>:<port>")
또는
>db.runCommand({addshard : "<hostname>:<port>"});
```

```
2. Shard Server의 삭제
>db.runCommand({removeshard : "<Shard Server명>"});
```

```
3. Shard Server의 이동
>db.runCommand({movePrimary: "<대상 Shard>", to:"<이동될 Shard>"});
```

- Shard 시스템의 장점은 수평적 확장이 용이

Chunk 크기 관리

```
1. Shard Server의 List

>db.runCommand({listShards : 1})
```

```
2. Chunk Size의 변경
>use config
>db.settings.save({_id:"chunksize",value <size>})
```

```
3. Chunk의 분리
>sh.splitFind("<컬렉션명>": {"<Field>": values});
```

```
4. Balancer의 Lock Check
>db.locks.find({_id : "balancer"}).pretty()

>sh.getBalancerState()  ##Balancer의 작동여부 상태 확인
```

- Mongos(balancer)프로세스가 샤드서버 데이터를 분산하다 보면 작업의 일관성, 연속성을 위해 잠금 현상이 발생 할 수있음

```
5. Balancer Window 변경
>db.settings.update(
{_id : "balancer"},{set : {activeWindow :
{ start : "<start-time>",stop : "<stop-time>"}}},true)

>db.settings.update(
{_id : "balancer"},{$unset : {activeWindow : true})
```

- 마이그레이션 시 성능 저하 현상 발생 때문에 유휴 시간대에서만 작동하도록 밸런스 위 윈도우를 설정

```
6. Balancer의 시작/종료
>sh.stopBalancer()
>sh.startBalancer()
```

샤딩 System의 문제점

- 하나의 Shard 서버에 데이터가 집중 되고 Load Balancing이 되지 않는 경우
  . 샤드 키로 설정 된 필드의 카디널리티가 낮으면 64MB보다 큰 Chunk 크기로 생성
  → 하나의 서버에 데이터가 집중
- 특정 Shard 서버에 IO트래픽이 증가하는 경우
  . CRUD가 빈번하게 발생하는 데이터인 Hot Chunk가 특정 서버에 집중되어 있을 때 해당 서버에 IO트래픽이 증가
- 샤드 서버 클러스터의 밸런스가 균등하지 않는 경우
  . 사용자 필요에 따라 특정 서버의 데이터를 삭제, 이동 시
  . 낮은 카디널리티의 샤드 키를 설정 시
  . 하나의 컬렉션에 데이터가 빠르게 입력 되는 것 보다 shard 서버에 분산 저장되는 속도가 늦는 경우
  . 빅데이터의 빠른 저장에 비해 Chunk Migraion이 빈약한 네트워크 대역폭으로 빠르게 이동되지 못할 때
  . Balance Window를 너무 길게 설정 하는 경우 Chunk Migration이 드물게 발생하여 밸런스가 깨짐
- 과도한 Chunk Migration이 클러스트 작동을 멈추는 경우
  . 빈번한 Chunk Migration은 일시적으로 클러스터 서버 전체 성능 지연이 발생하므로 밸런싱 윈도우 기능을 통해 피크 시간을 피해 밸런싱
  . 큰 Chunk Size는 네트워크 트래픽을 증가 시키고 resource 효율성을 저하시키므로 Chunk size를 줄여 과부하를 줄임
  . 샤드 서버의 밸런싱이 적절하지 않은 것은 데이터 발생양에 비해 샤드 서버 수가 부족하기 때문이므로 샤드 서버 수를 늘려줘야함
- 쓰기 성능이 지연되고 빠른 검색이 안되는 경우
  .하나의 컬렉션은 여러개의 익스텐트 구조로 생성되는데 익스텐트가 너무 작게 생성되어 있으면 alloc 하는 불필요한 대기 시간이 발생하여 쓰기 성능 지연시키므로 적절한 익스텐트 크기로 생성
  .일반적으로 RDBMS는 데이터 중복을 방지하고자 여러개의 테이블로 분리 설계하는 것이 보편적 이지만 MongoDB에서ㄴ는 하나의 Embedded 도큐멘트 구조로 설계하는것이 빠른 쓰기 성능에 도움(rich Document 구조)
  . 싱글 노드에서 요구되는 시스템 메모리와 클러스트 노드(샤딩)에 요구되는 메모리 요구사항이 다를 수 밖에 없기 때문에 추가적인 메모리가 요구

Zone Sharding

. MongoDB 3.6부터 제공 Auto Sharding을 통해 분산 저장 되는 데이터를 사용자가 직접 원하는 Shard Server에 저장되도록 설계하는 방법

. Auto Sharding은 Chunk 단위로 데이터를 분산 저장하기 때문에 사용자는 원하는 조건의 데이터를 특정 서버에 저장할 수 없으므로 데이터의 분산 정도를 제어할 수 없는 단점

수동 샤딩

- RDBMS에서 데이터 분산 저장을 위해 샤딩을 구축하는건 비용과 관리문제로 어려웠던 반면 Auto Sharding을 통해 MongoDB는 편리하게 샤딩을 구성해 분산 저장 가능
- 쟈동 샤딩은 사용자의 의도대로 데이터를 분산, 저장할 수 없어 선택인 옵션기능 (MongoDB에서 제공하는 알고리즘에 의해 분산, 저장)
- 자동 샤딩은 Shard-Key를 기준으로 여러대의 샤드 서버에 Chunk 단위로 데이터를 분산, 저장하는 개념인 반면 수동 샤딩은 여러대의 서버에 특정 범위의 데이터가 배치될 수 있도록 사용자가 직접 설계
- 데이터의 생성량 참조 빈도를 조사하여 사용자가 균등하게 분산 저장
- mongos를 통해 자동 처리 할 수 없기 때문에 SW를 통해 저장되어 있는 서버를 검색해야 하므로 개발에 추가적인 resource가 요구됨
- 사용자가 데이터를 원하는 데이터 서버에 직접 저장 및 배치가 가능함(Zone Sharding방법과 유사)

추가 자료

- 4.2버전 signing key 이슈
  - 90일 주기로 config server stop down 권고 → 180일 후 mongodb cluster가 hang 걸리는 이슈가 발생함(application에서도 접속 불가)
  - replica set 방식에서는 이슈가 없음
  - 최신 3.6 ~ 4.2 버전 까지는 발생했으나 릴리즈 버전(4.2.12)에서는 bug fix된걸
  - 참고 자료 : https://jira.mongodb.org/browse/SERVER-52654