6. Sharding System

차례

1. Sharding 개념과 정의

- 샤딩의 목적
  - 데이터의 분산

    한 대의 서버에 빅데이터를 저장하는 것은 불가능
    - 서버의 성능 저하 유발 : 초당 발생하는 엄청난 양의 Insert 동작시 Write scaling 문제 발생
    - 디스크를 사용하는 하드웨어 한계성

데이터를 분산하여 순차적으로 저장한다면 한 대 이상에서 트래픽을 감당하기 때문에 부하를 분산하는 효과가 있다.

1. Sharding 개념과 정의

- 샤딩의 목적
  - 백업과 복구 전략

    데이터 분산이라는 샤딩의 가장 대표적인 기능을 통해 얻는 효과
    - 시스템의 성능 향상
    - 데이터 유실 가능성으로부터 보호

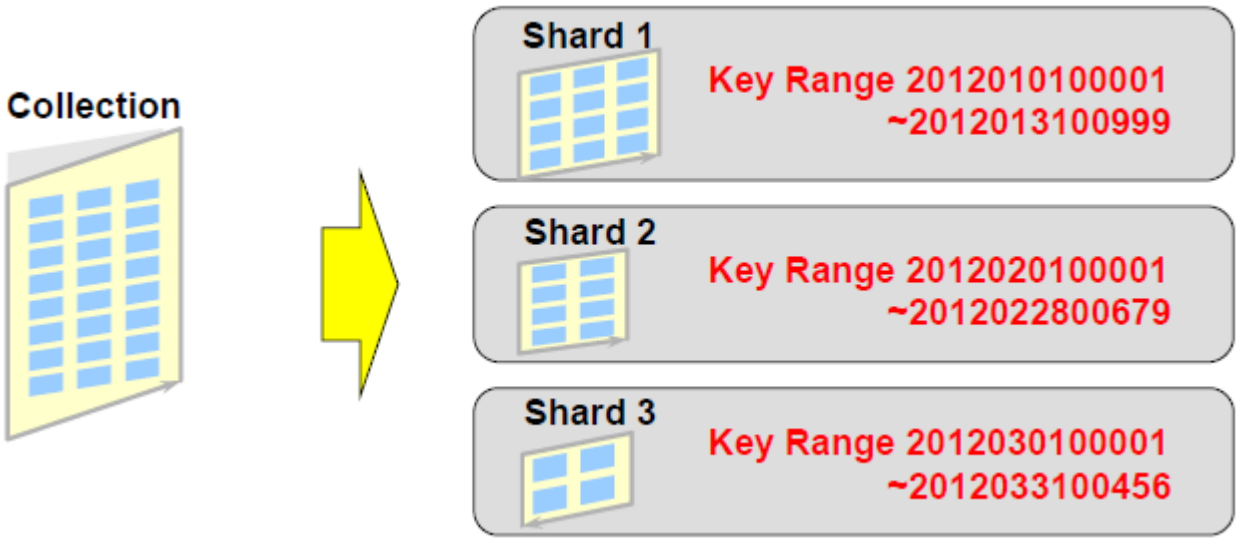
    서버의 데이터가 유실된다면 그 데이터 양은 상상을 초월할 것이고 시스템 복구에 엄청난 시간과 비용이 소요됨

> 미리 데이터를 분산하여 저장해둔다면 리스크로 부터 보호받고
> 효과적인 시스템 운영이 가능해진다.

1. Sharding 개념과 정의

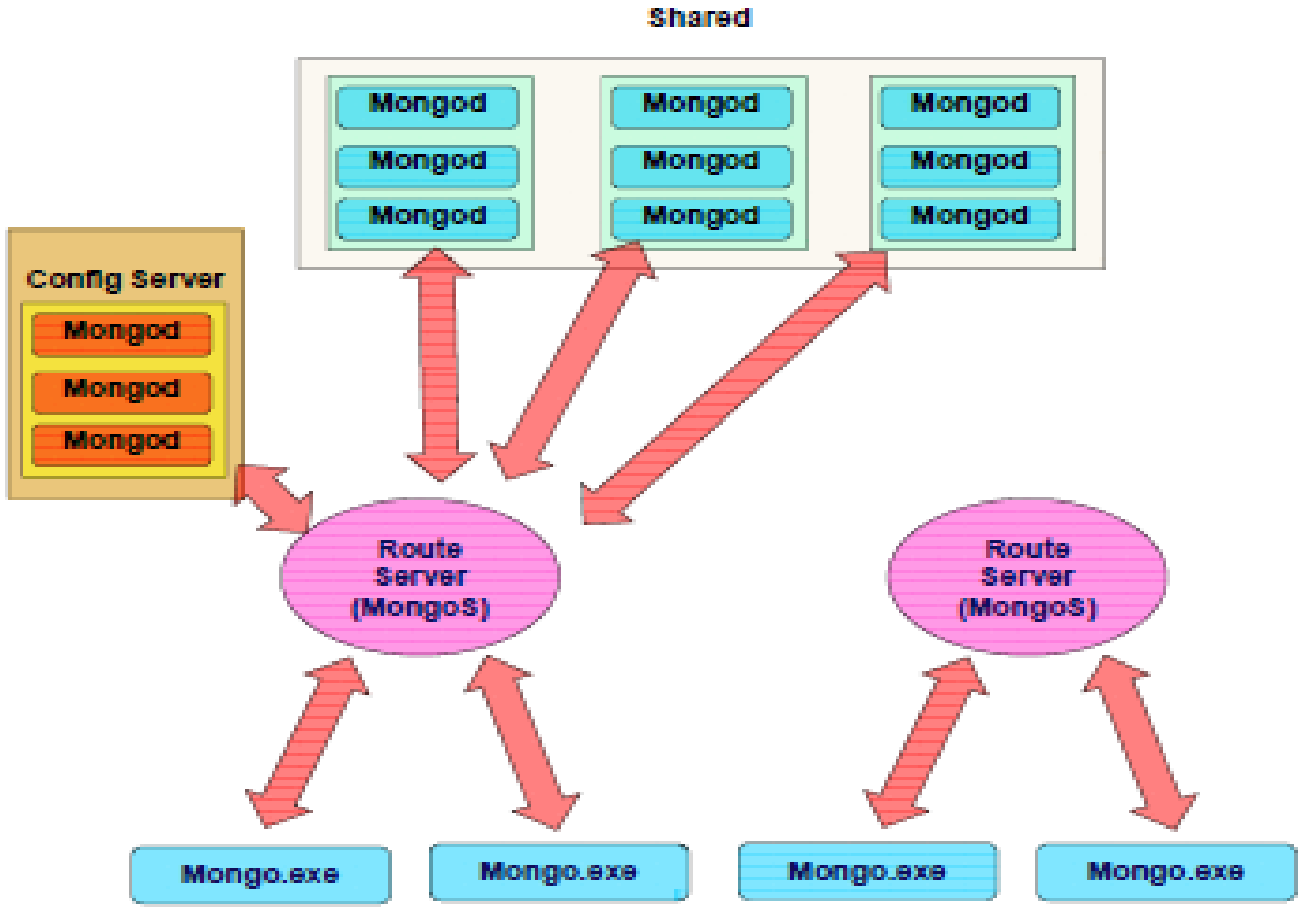
- 샤딩의 목적
  - 빠른 성능

    여러 대의 독립된 프로세스가 병렬오 작업을 동시에 수행하기 때문에 이상적으로 빠른 처리 성능 보장



1. Sharding 개념과 정의

- Sharding 시스템 구조



1. Sharding 개념과 정의

- Sharding 구축을 위한 시스템 환경
  - 3대 이상의 샤드 서버로 구축 권장
  - 싱글 노드를 운영할 때는 요구되는 메모리 영역보다 20~30%% 이상 추가 메모리 요구(샤드 시스템 구축할 때 활성화되는 Mongos와 OpLog, Balancer 프로세스가 사용할 추가 메모리에 대한 고려)
  - 샤드 시스템 구축할 때 요구되는 config 서버는 최소 3대 이상 활성화 할 것을 권장

1. Sharding 개념과 정의

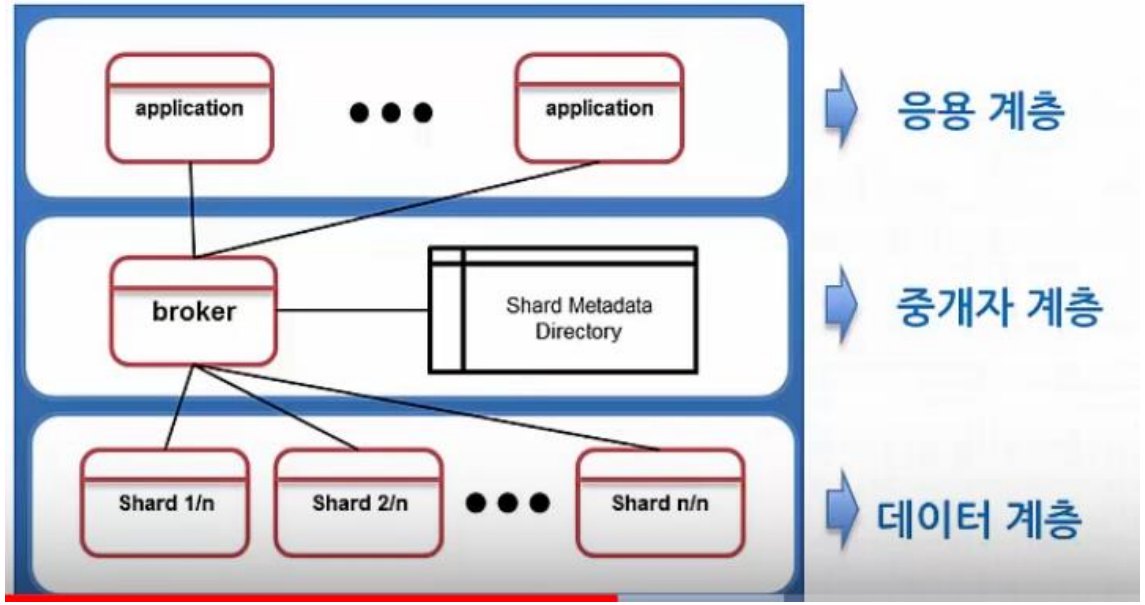
- Config Server 주요특징
  - **config 서버는 샤드 시스템에대한 메타 데이터 저장/관리 역할**
  - **샤드 서버의 인덱스 정보를 빠르게 검색 가능하게 함**
  - **샤드 서버와 별도의 서버에 구축이 기본**
  - **장애 발생에 대비해 최소 3**대 이상 사용(최소 1대만으로도 운영 가능) 만약 하나의 Config Server가 다운되면 나머지는 Read Only 됨
  - **Shard**서버에 비해 저사양 서버 사용 가능

1. Sharding 개념과 정의

- Mongos 프로세스의 주요 특징
  - 하나 이상의 프로세스가 활성화 됨
  - **Application Server**에서 실행 가능
  - Config Server로부터 Meta-Data를 Cache한다
  - 빅데이터를 샤드 서버로 분산해주는 프로세스이다.
    - Config 서버는 각 샤드 서버에 어떤 데이터들이 어떤 식으로 분산 저장되어 있는지에 대한 Meta 데이터가 저장되어 있는데, mongos 서버를 통해 데이터를 쓰고 읽는 작업이 가능
    - Mongos는 각 서베에서 어떤 일을 하는지 개발자가 모르게 해주는 역할을 한다.
    - Mongos를 통해 샤딩 상태인지 리플리케이션 상태인지 개발자는 알 필요가 없다.

1. Sharding 개념과 정의

- 샤딩 시스템의 계층
  - 중계자 계층은 샤딩 시스템의 가장 핵심적인 부분으로, 샤드 메타 정보를 저장하여 응용계층으로부터 전달된 질의를 분석하여 적절한 샤드에 명령을 수행 시킨 다음 그 결과를 응용계층으로 전달



5. MongoDB Sharding 작업

- config 서버 생성 및 실행
  - 리플리카 셋으로 config 서버를 구성한다.
  - config 서버 폴더 생성

```
mkdir d:\mongo\shard\config01
mkdir d:\mongo\shard\config02
mkdir d:\mongo\shard\config03
```

  - config 서버 실행

```
mongod --configsvr --replSet configRepl --dbpath d:\mongo\shard\config01 -port
20001
mongod --configsvr --replSet configRepl --dbpath d:\mongo\shard\config02 -port
20002
mongod --configsvr --replSet configRepl --dbpath d:\mongo\shard\config03 -port
20003
```

- config 서버 접속 및 리플리카 셋 설정

```
mongo localhost:20001 (config01 server 접속)

var config = {
    _id : "configRepl", members : [
        {_id : 0, host : 'localhost:20001'},
        {_id : 1, host : 'localhost:20002'},
        {_id : 2, host : 'localhost:20003'}
    ]
}

- 설정 후 접속한 서버가 primary 서버로 바뀐 것을 확인 할 수 있다.

rs.initiate(config)
```

5. Shard 서버 테스트

1) Shard Server 폴더 생성

- 각각의 **Shard Mongo** 서버 역시 리플리카 셋으로 구성한다.

```
mkdir d:\mongo\shard\shard1\shardRep1\data
mkdir d:\mongo\shard\shard1\shardRep2\data
mkdir d:\mongo\shard\shard1\shardRep3\data

mkdir d:\mongo\shard\shard2\shardRep1\data
mkdir d:\mongo\shard\shard2\shardRep2\data
mkdir d:\mongo\shard\shard2\shardRep3\data

mkdir d:\mongo\shard\shard3\shardRep1\data
mkdir d:\mongo\shard\shard3\shardRep2\data
mkdir d:\mongo\shard\shard3\shardRep3\data
```

mongodb 3.4 버전부터 **Config Server** 는 복제서버**(ReplicaSets)**로 구성해야 함

- shard 서버 구성을 위한 리플리카 셋 설정
  - 서버 실행: 총 9개의 서버를 실행시켜준다

```
mongod --shardsvr --replSet shardRep1 --dbpath d:\mongo\shard\shard1\shardRep1\data -port 30011
mongod --shardsvr --replSet shardRep1 --dbpath d:\mongo\shard\shard1\shardRep2\data -port 30012
mongod --shardsvr --replSet shardRep1 --dbpath d:\mongo\shard\shard1\shardRep3\data -port 30013

mongod --shardsvr --replSet shardRep2 --dbpath d:\mongo\shard\shard2\shardRep1\data -port 30021
mongod --shardsvr --replSet shardRep2 --dbpath d:\mongo\shard\shard2\shardRep2\data -port 30022
mongod --shardsvr --replSet shardRep2 --dbpath d:\mongo\shard\shard2\shardRep3\data -port 30023

mongod --shardsvr --replSet shardRep3 --dbpath d:\mongo\shard\shard3\shardRep1\data -port 30031
mongod --shardsvr --replSet shardRep3 --dbpath d:\mongo\shard\shard3\shardRep2\data -port 30032
mongod --shardsvr --replSet shardRep3 --dbpath d:\mongo\shard\shard3\shardRep3\data -port 30033
```

- 각각의 리플리카 셋 서버 접속 및 설정

```
mongo localhost:30011
var config = {
    _id : "shardRep1", members : [
        {_id : 0, host : 'localhost:30011'},
        {_id : 1, host : 'localhost:30012'},
        {_id : 2, host : 'localhost:30013'}
    ]
}
rs.initiate(config)
```

```
mongo localhost:30031
var config = {
    _id : "shardRep3", members : [
        {_id : 0, host : 'localhost:30031'},
        {_id : 1, host : 'localhost:30032'},
        {_id : 2, host : 'localhost:30033'}
    ]
}
rs.initiate(config)
```

```
mongo localhost:30021
var config = {
    _id : "shardRep2", members : [
        {_id : 0, host : 'localhost:30021'},
        {_id : 1, host : 'localhost:30022'},
        {_id : 2, host : 'localhost:30023'}
    ]
}
rs.initiate(config)
```

rs.status() 명령어로 설정된 리플리카 셋 설정 정보를 확인 할 수 있다.

Mongos(shard 서버) 설정

- 위에서 설정한 config 서버를 각 실행해둔다.

```
mongos --configdb configRepl/localhost:20001,localhost:20002,localhost:20003
```

Sharding 설정

- 클라이언트가 자동으로 데이터 서버를 접근하도록 하기 위한 샤딩 서버(중개 계층) 설정을 해준다.
  - mongo 입력하면 mongos로 접속이 된다.

```
mongo
```

  - mongos에서 각 shard를 설정한다.
  - mongo로 접속하면 자동으로 mongos로 접속되는 것을 확인할 수 있다.

- 샤드 설정

```
sh.addShard("shardRep1/localhost:30011")
sh.addShard("shardRep2/localhost:30021")
sh.addShard("shardRep3/localhost:30031")
```

- 샤드 DB 등록

```
sh.enableSharding("test")
```

- 샤딩 시킬 collection의 인덱싱 설정

```
use test
db.things.createIndex({ empno : 1})
```

- 샤드 시킬 콜렉션 설정(admin)

```
use admin
sh.shardCollection("test.things", {empno : "hashed"})
```

- 테스트 데이터 삽입

```
use test

for(var n = 100000; n<=110000; n++){

    db.things.insert({empno : n, ename : 'test', sal : 1000})
}

db.things.count() //데이터 갯수 확인
```

- 각 mongo 서버 데이터 확인

```
use admin

db.runCommand({ listshards })
```

```
mongo localhost:30011
use test
db.things.count()
exit

mongo localhost:30021
use test
db.things.count()
exit

mongo localhost:30031
test
db.things.count()
```

6. Sharding 시스템 구축시 고려사항

- Shard Key

```
db.runCommand( { shardcollection : “sales.order”,
                 key : { idate : 1 }})
```

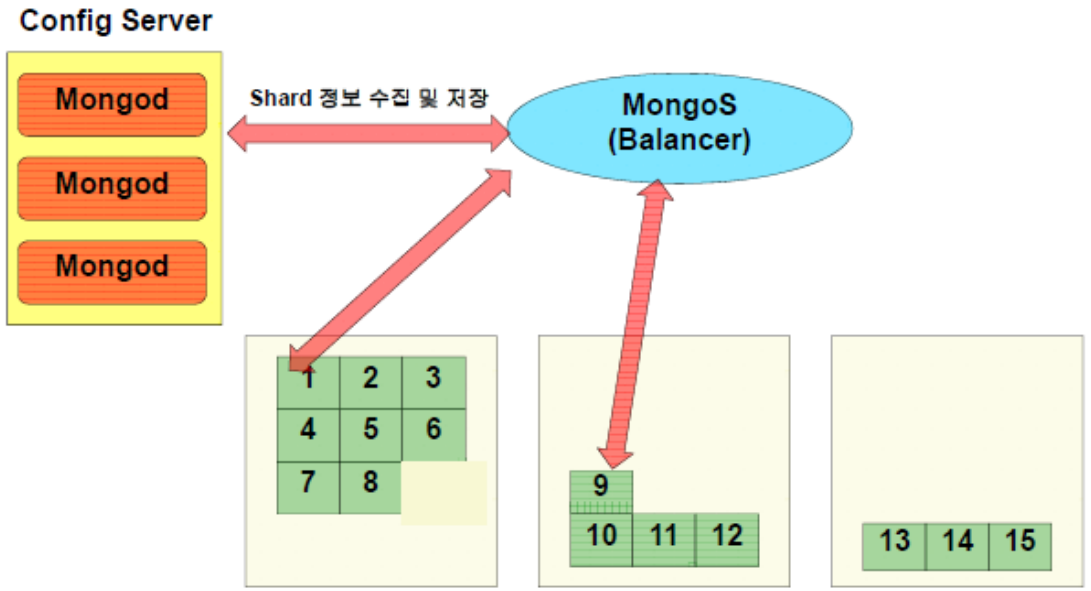
  - Shard Key는 여러 개의Shard Server로 분할될기준 Field이므로 충분한 고려를 해야 한다.
  - Shard Key는 Partition과 Load Balancing의 기준이되므로 MongoDB의 데이터 저장과 성능에 절대적 영향을 미친다.
  - Shard Key를 선정하는 것은 Cardinality, 데이터의 분산 저장, Query Isolation, 신뢰도, Index Locality 등을 고려하여결정
  - 하나 이상의 Field로 Shard Key 를 구성할 수 있다.(최대 512를 초과할 수 없다.)

- Hashed Index

```
> db.employees.ensureIndex( { deptno: "hashed" } )
```

  - **Hashed Index**는 **MongoDB V2.4**에서 제공
  - 사용자에 의해 정의된 Shard Key가 적절하게 분산되지 않는 문제 해결을 위해 MongoDB에서 제공하는 해시 알고리즘을 적용할수 있다.
  - 인덱스대상 필드의값이 해시값으로변환되어저장되기때문에 필요한 변환이 발생함으로 사용자가 해당 기술에 대한 정확한 이해를바탕으로사용하되적절한Shard Key가 없거나기술에 대한 이해가 부족한 경우 적용하는 것이 좋다.
  - **Compound Index, Unique Index, Multi-Key Index**에는 적용할 수 없다.

- Chunk Migration



1) MongoS는 각 **Shard**의 **Balance**에 불균형**(8**개 이상의 **Chunk** 개수**)**이 발행하면 **Chunk**의 **Migration** 작업을 수행한다**.**

2) Migration이 발생할 때 **Mongos**에 **Lock**이 발생한다**.**

3) 실제로는 **Move**가 아니라 데이터를 복사하고 소스 **Chunk** 데이터는 삭제된다.

7. Shard 서버의 추가와 삭제

- Shard Server의 추가

```
> sh.addShard( "<hostname>:<port>" )
또는
> db.runCommand( { addshard  : “hostname>:<port>" } );
```

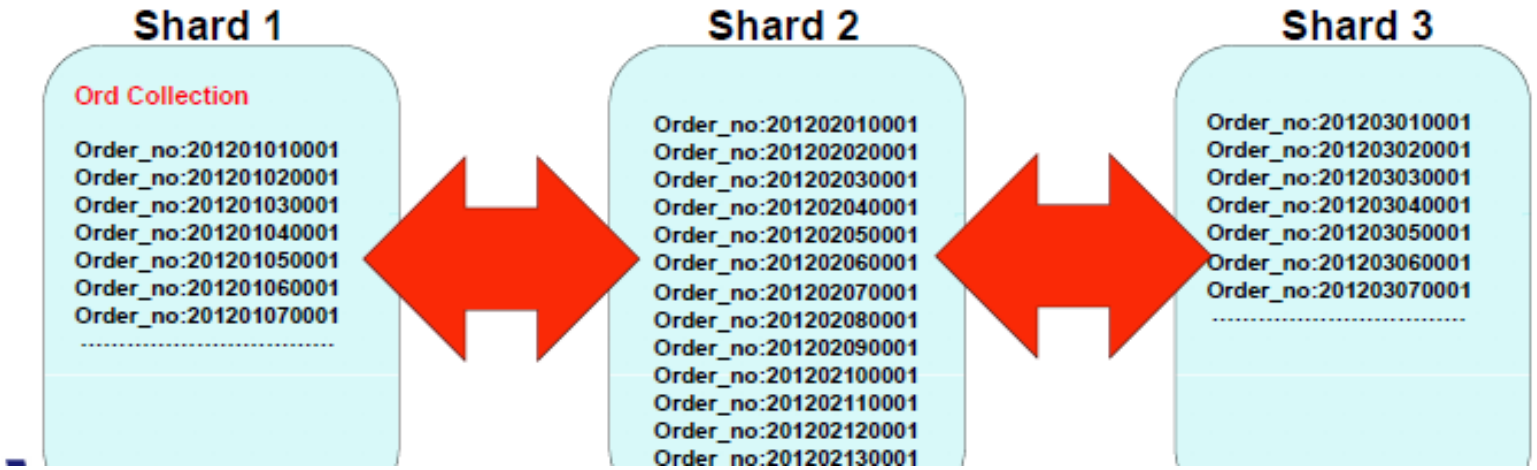
- Shard Server의 삭제

```
> db.runCommand( { removeshard: “<Shard Server명" } )
```

- Shard Server의 이동

```
> db.runCommand(
  { movePrimary: “< 대상 Shard>, to: "<이동될 Shard>" })
```

8. Shard System의 문제점



1) 하나의 **Shard** 서버에 데이터가 집중되고 **Load Balancing**이 되지않는 경우

2) 특정 **Shard** 서버에 **IO** 트래픽이 증가하는 경우

3) 샤드 서버 클러스터의 밸런스가 균등하지 않는 경우

4) 과도한 **Chunk Migration**이 클러스트 작동을 멈추는 경우