월간 10억 PV를 지지하기 위한 MongoDB Tip

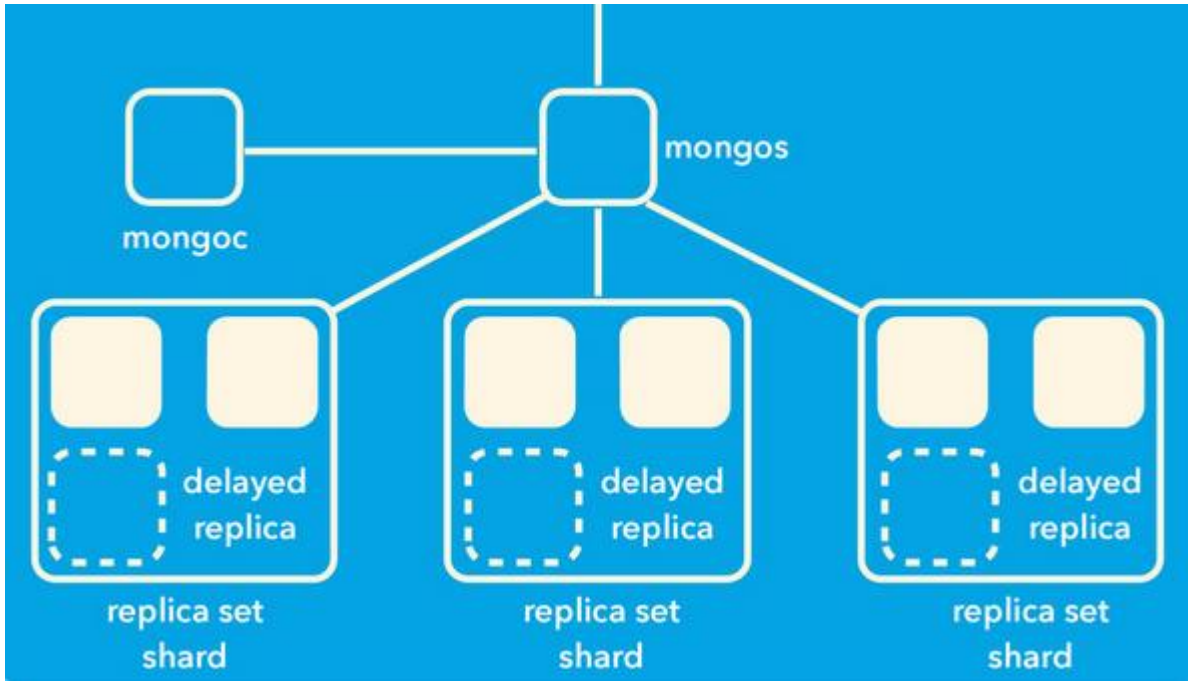
최흥배

아래 두 개의 문서 내용을 정리한 것

http://www.slideshare.net/yujiosaka/10pvmongo-db

https://speakerdeck.com/yujiosaka/yue-jian-10yi-pvkaraxue-ndamongodbantipatan

- 합계 월간 PV 10억
- 초당 편균 400PV
- 평균 동시 접속 수 5억
- 월간 보존 데이터 량 10TB



architecture:

front_ends: [
  “Nginx”
  “Ruby on Rails”
  “MySQL”
  “memcached”
  “AWS”
]

back_ends: [
  “Node.js”
  “CoffeeScript”
  “MongoDB”
  “Redis”
  “AWS”
]

스키마 디자인

```
timeline:
 _id: "yujiosaka"
 tag: "#dbtechshowcase"
 tweets: [
  {text: "MongoDB Rocks!"}
 ]
```

읽기 중시
1 쿼리 1 히트
코멘트 수에 따라서 쓰기가 무겁게 된다.

```
timeline:
 _id: "yujiosaka"
 tag: "#dbtechshowcase"

tweet:
 _id: 1234
 user_id: "yujiosaka"
 text: "MongoDB Rocks!"
```

쓰기 중시
2 쿼리 N+1 히트
쓰기 확장성

```
timeline:
 _id: "yujiosaka-20141111"
 tag: "#dbtechshowcase"
 tweets: [
  {text: "MongoDB Rocks!"}
 ]
```

밸런스 타입
1 쿼리 날수만큼 히트
쓰기는 1일분 정도만 무겁게 된다.

"스키마 레스"는 스키마 정의가 필요하지 않다가 아니다.

RDBMS 보다 선택기가 많아서 어렵다.

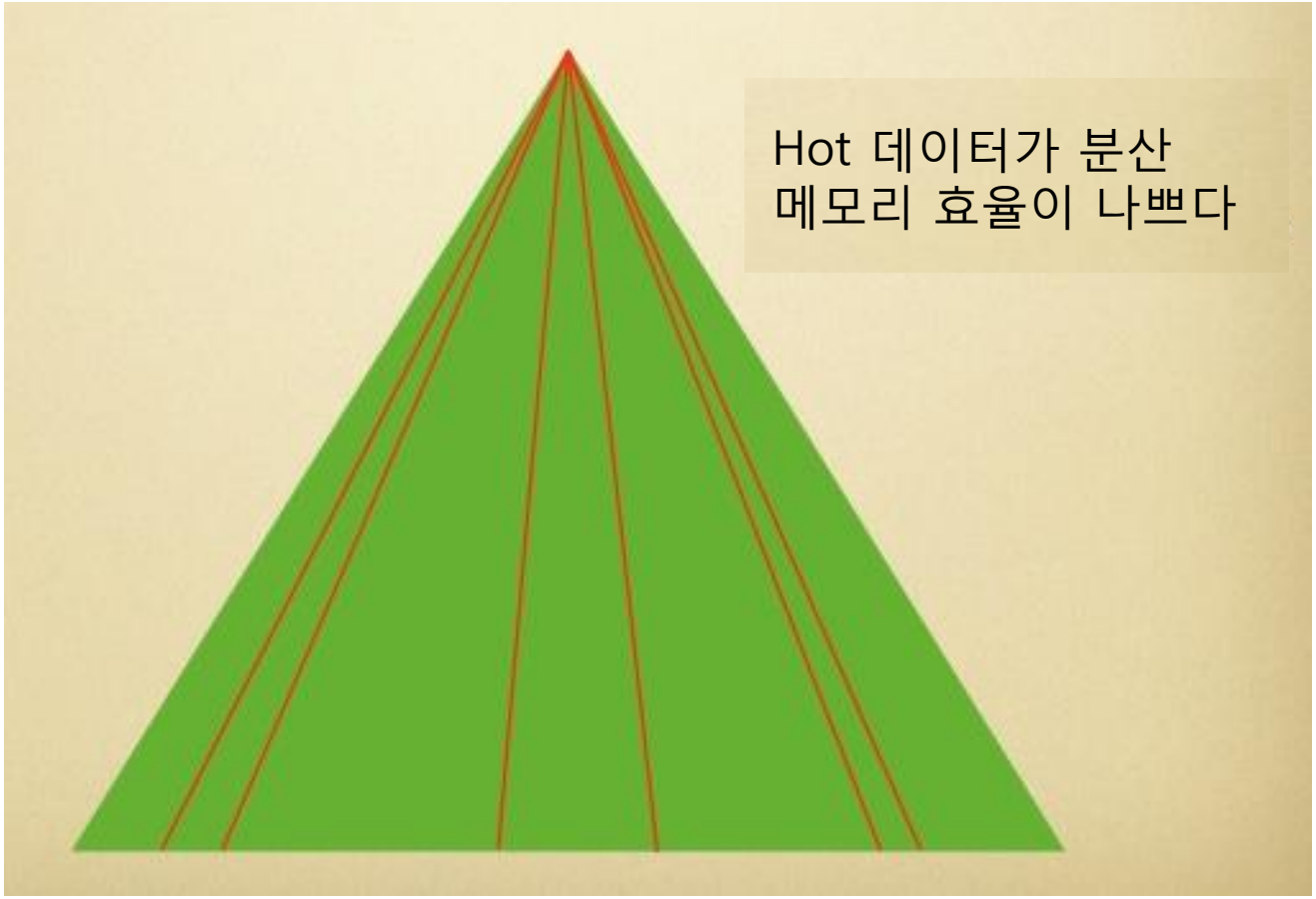
인덱스에 대해서 잘 알아야 한다

- 메모리 상에 올라가면 고속
  - 인덱스 정보가 메모리 상에 올라가 있지 않으면 느리게 된다.
  - 컬렉션에서 필드를 삭제할 때 사용하지 않는 인덱스가 남지 않도록 조심.

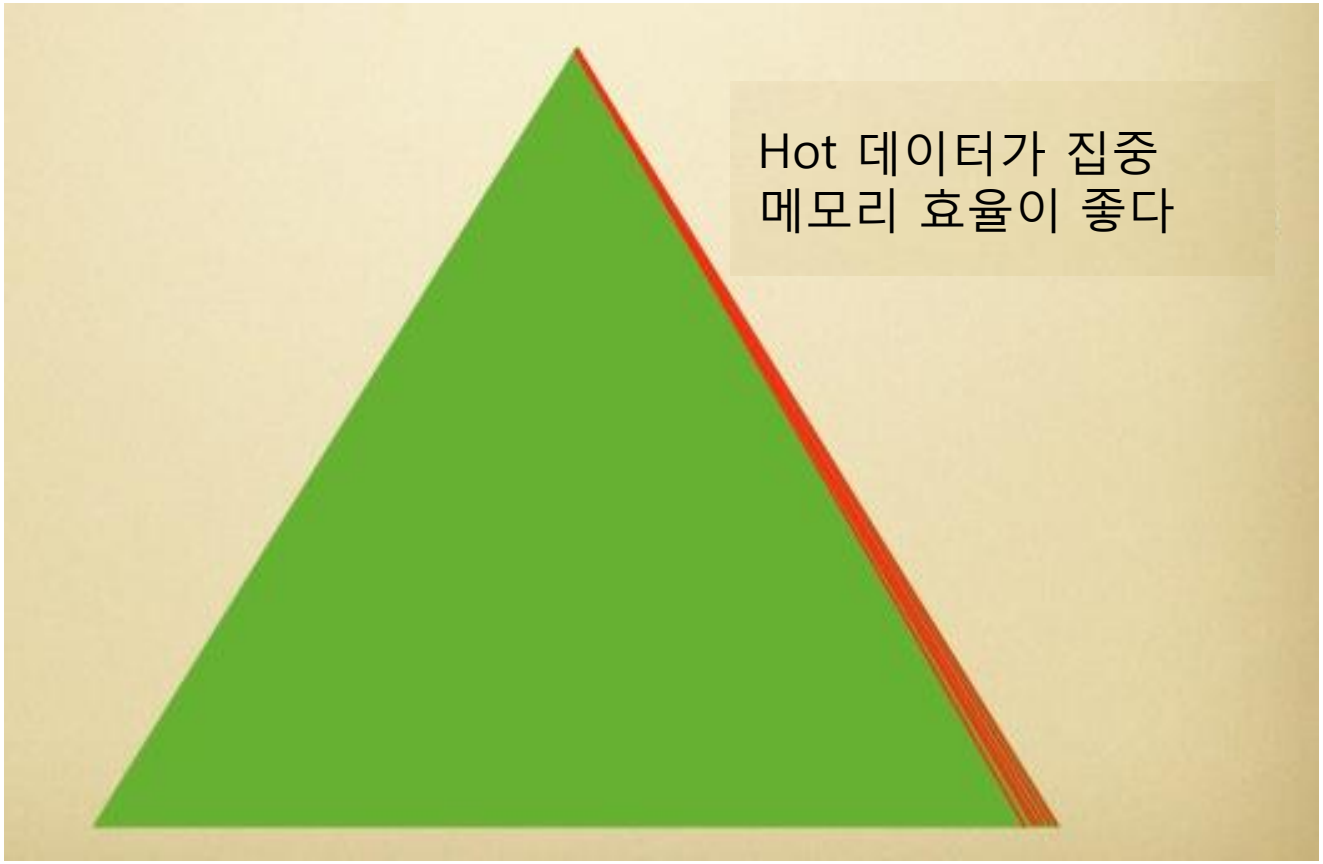
- 1쿼리 당 1 인덱스

- 인덱스의 구조
  - 인덱스에 B-Tree 자료 구조 사용
  - B-Tree 보통 균등하게 분포하려고 한다

문자열 인덱스



ObjectId * 시간 인덱스



가능한 ObjectId를 사용하는 것이 좋다.

필드를 지정하지 않는 쿼리는 피한다

```
user: {_id: "yujiosaka", first: "Yuji", last: "Isobe", city: "Tokyo"}
index: {_id:1, city: 1} // compound index
```

```
var user = db.users.findOne({_id: "yujiosaka"});
```

SQL에서 'select *'
불 필요한 네트워크 비용
불 필요한 디스크 접근

```
var user = db.users.findOne({_id: "yujiosaka"}, {city: 1});
```

SQL에서 'select city'
네트워크 비용 절약
covered index를 활용할 수 있다.

covered index를 잘 사용하면 디스크 접근 없이 메모리만으로 정보를 읽을 수 있다.

쓰기 성능 올리기

- mongoDB는 데이터가 증가하는 Update는 느리다.
- Bulk Inserts/Operations를 사용하자.

```
db.col.update({_id: 1234}, {$push: {points: {x: 151, y: 100}}});
db.col.update({_id: 1234}, {$push: {points: {x: 151, y: 179}}});
db.col.update({_id: 1234}, {$push: {points: {x: 151, y: 266}}});
...
db.col.update({_id: 1234}, {$push: {points: {x: 151, y: 340}}});
```

redis를 사용하여 빈번하게 업데이트 하지 않도록 한다

DB 설계

- 용도 마다 DB를 나눈다.
  - mongoDB 2.6 기준으로 DB 단위로 lock 한다.
  - mongoDB 2.8에서 도큐먼트 단위 lock으로 바뀔 예정.

- Hot 데이터/Clod 데이터를 나눈다.
  - Hot 데이터의 lock을 피한다.
  - Hot 데이터 용으로 대용량 메모리.
  - Cold 데이터 용으로 대용량 스토리지.

- 오래된 데이터를 삭제한다.
  - S3에 백업.
  - 통계 데이터를 저장.
  - 정기적으로 자동 삭제

Update 쿼리

```
var user = db.users.findOne({_id: "yujiosaka"});
user.count += 1;
user.save();
```

꼭 쿼리가 실행된다.
쓸데없는 네트워크 비용 발생
atomic 하지 않은 처리

```
db.users.update({_id: "yujiosaka"}, {$inc: {count: 1}});
```

update와 $inc를 같이 사용
1회 업데이트로 끝
atomic한 처리

Update 쿼리 (계속)

```
var user = db.users.findOne({_id: "yujiosaka"});
if (user) {
  user.count += 1;
  user.save();
} else {
  db.users.insert({_id: "yujiosaka", count: 1});
}
```

데이터를 발견하면 갱시하고 없다면 만든다.
복잡한 코드

```
db.users.update({_id: "yujiosaka"}, {$inc: {count: 1}}, {upsert: true});
```

{update:true}를 사용하여 처리를 모은다.
보다 간단한 코드

튜닝

- 메모리는 많은면 많을 수록 좋다.
- 디스크 접근도 빠르면 좋다. SSD 사용 추천

AWS EC2 인스턴스 선택 방법

- 일반적 목적(m3). 보통 처음에 선택하는 인스턴스
- 메모리 최적화(r3). 메모리와 CPU 밸런스가 좋다.
- 저장 최적화(i2/hs). SSD를 사용. 고부하에 견딜 수 있다.

r3.2xlarge
-> vCPU: 8
-> Memory: 61GiB
-> SSD: 160GB
-> Cost: $0.840 per Hour

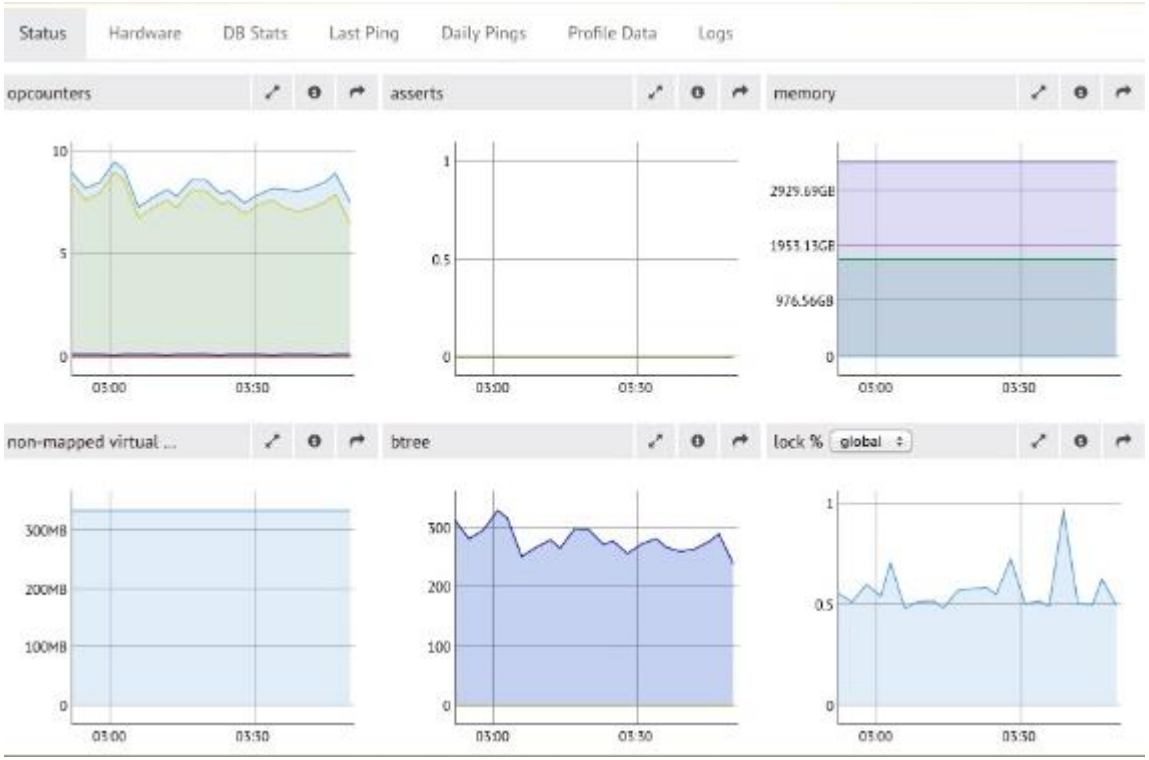
i2.2xlarge
-> vCPU: 8
-> Memory: 61GiB
-> SSD: 800GB x 2
-> Cost: $2.001 per Hour

모니터링- mongoHQ

- 호스팅 서비스의 대시보드
- 매일 건강 체크 용
- 과거의 정보는 참조할 수 없다.

모니터링- MMS

- 정기적인 건강 진단용
- 실 시간 감시는 MongoHQ, 과거 상태 조사는 MMS
- 경고와 백업도 할 수 있다.
- 도입이 간단하므로 먼저 사용해 보는 것을 추천



느린 쿼리 감시

- explain(true)
- system.profile.find()
- MongoDB professor
- Dex
- MongoHQ Slow Queries

ObjectId를 사용한 쿼리를 실행했을 때
100ms 이상 걸린다면 **위험 신호**

MongoDB 최적화 하기

- 인덱스 사양을 이해한다.
- 쓰기 횟수를 줄인다.
- DB를 용도별로 나눈다.
- 메모리를 늘린다.
- 계속적으로 감시한다.