장금

MongoDB 의 장금

MongoDB 에서도 여러 계층의 데이터베이스 오브젝트들에 대한 동시 처리를 위해 인텐션 락(Intention Lock)과 다중 레벨의 잠금(Multi Granulariry Locking) 을 채용하고 있다.

mongodb 3.0 이전에는 mmapv1 만을 사용할 수 있었고, 서버의 잠금(global, database, collection) 에만 의존해 왔었다.

mongodb 3.2 에서 wiredTiger 가 도입된 이후 부터 동시처리에 대한 성능을 위해 인텐션 락이 채용되었다.

장금 의 종류

명시적 장금

사용자가 명시적으로 장금을 할 수 있는 것은 글로벌 장금뿐이다. 글로벌 장금은 다음과 같은 특징이 있다.

- instance 전체의 쓰기 작업을 금지 시키며 조회만 가능하게 된다.
- 쓰기 장금이 되더라고 저널링으로 인한 동기화 및 wiredTiger 의 스냅샷 으로 인해 데이터가 변경 될 수 있다.
    - fsyncLock() 을 통한 백업시 유의
    - 명시적 장금은 replication 도 차단 되므로 주의 해야 한다.

```
> db.fsyncLock({fsync:1, lock:true})
{
        "info" : "now locked against writes, use db.fsyncUnlock() to unlock",
        "lockCount" : NumberLong(1),
        "seeAlso" : "http://dochub.mongodb.org/core/fsynccommand",
        "ok" : 1
}
> db.fsyncUnlock();
{ "info" : "fsyncUnlock completed", "lockCount" : NumberLong(0), "ok" : 1 }
>
```

묵시적 장금

사용자의 operation 에 따라 4가지의 장금이 묵시적으로 발생하며 적절한 시점에 해제된다. 묵시적 장금은 global, database, collection 별로 아래의 lock 상태를 가질 수 있다.

wiredTiger 에서는 record lock 을 지원하는데 현재 mongodb 에서 스토리지 레이저는 별도로 구분되어 record lock 에 대해서 wiredTiger 에 일임하며, lock상태 조회시에도 조회되지 않는다.

lock 모드	설명
R	shared (S) lock
W(대문자)	Exclusive (X) lock

lock 모드	설명
r	intent shared (IS) lock
w	intent Exclusive (IX) lock

mongodb 는 읽기와 쓰기에 대해 큐잉되어 순차적으로 처리 되지만 최적화를 위해 서로 용인되는 장금에 대해서는 후 순위의 장금이 먼저 수행 되기도 한다.

```
원래 순서 : IS - IS - X - X - S - IS
최적 순서 : IS - IS - S - IS - X - X
```

다음은 operation 에 따른 장금의 종류를 설명한 표이다.

operation	database	collection
issue a query	r(intent shared)	r(intent shared)
insert data	w(intent exclusive)	w(intent exclusive)
remove data	w(intent exclusive)	w(intent exclusive)
update data	w(intent exclusive)	w(intent exclusive)
perform aggregation	r(intent shared)	r(intent shared)
create an index(foreground)	W(Exclusive)	
create an index(background)	w(intent exclusive)	w(intent exclusive)
list collections	r(intent shared)	
map-reduce	W(Exclusive) and R(shared)	w(intent Exclusive) and r(intent shared)

궁금점

다음은 createIndex(foreground) 에 대해 로그에서 발취한 내용인데 Collection 에 대해서만 Exclusive 가 생성되었다. (db.currentOp() 는 확인 못함)

```
{
  "type": "command",
  "ns": "test.sparse",
  "appName": "MongoDB Shell",
  ... ...
    "Global": {
      "acquireCount": {
        "w": 4
      }
    },
    "Database": {
      "acquireCount": {
        "w": 3
```

```
        }
      },
      "Collection": {
        "acquireCount": {
          "r": 1,
          "w": 1,
          "W": 1
        }
      },
    ... ...
}
```

장금 yield

RDBMS 는 한번 획득한 장금은 operation 이 끝날때 해제하지만 mongodb 는 동시성 처리를 위해 특정 조건을 만족하는 경우 획득한 장금을 해체하고 os 스레드 스케쥴링에 따라 재획득 후에 지속적으로 처리를 하는데 이를 장금 yield 라고 한다.

> 장금을 yield 할때 atomic 이 유지 되는지???

장금 yield 조건 확인

```
> db.runCommand( {getParameter: '*'} )
{
        ... ...
        "internalQueryExecYieldIterations" : 1000,
        "internalQueryExecYieldPeriodMS" : 10,
        ... ...
}
```

Transaction

wiredTiger ACID

wiredTiger 가 제공하는 트랜잭션의 ACID 에 대한 특징은 다음과 같다.

- 최고 레벨의 격리 수준은 Snapshot(repeatable-read)
  - read-uncommitted, read-committed, snapshot 을 지원하지만 mongodb 에서 snapshot 으로 고정해서 사용한다.
- 트랜잭션의 commit 과 checkpoint 2가지 형태로 영속성 보장
- 커밋되지 않은 변경 데이터는 공유 캐시 크기보다 작아야 함

단일 ,멀티 다큐먼트 트랜잭션

mongodb 3.4 에서는 단일 다큐먼트 크랜잭션만 지원한다.
mongodb 4.0 에서는 리플리카 셋에서 멀티 다큐먼트 트랜잭션을 지원했으며

mongodb 4.2 에서는 sharded 클러스터 에서 분산 트랜잭션을 지원하며 이는 멀티 다큐먼트 트랜잭션을 포함하는 개념이다.

mongodb 4.2 multi-document transaction 에 대해 메뉴얼 상의 의문점

```
# atomic 하다
Multi-document transactions are atomic (i.e. provide an "all-or-nothing"
proposition):

# 아래 operation 은 atomic 하지 않다.?
Multi-Document Transactions
When a single write operation (e.g. db.collection.updateMany()) modifies multiple
documents, the modification of each document is atomic, but the operation as a
whole is not atomic.
```

멀티 다큐먼트 트랜잭션 사용시 주의점

멀티 다큐먼트는 임베디드 다큐먼트와 어레이를 통해 최소화 하는것이 성능상 매우 중요하다.

분산 트랜잭션의 visible

분산 트랜잭션에서 복수의 샤드에서 트랜잭션이 발생할때 read concern "local" 라면 전체 트랜잭션이 완료되지 않아도 완료된 샤드의 데이터를 볼 수 있다.