1. MongoDB 설치, WiredTiger 스토리지 엔진

MongoDB 설치

ReplicaSet 구성

mongodB 는 replicaset 을 최대 50대 까지 구성이 가능하다. 멤버 중에서 primary 선출은 7개까지만 가능하다.

프라이머리(primary)

- 데이터를 변경할 수 있는 유일한 멤버
- oplog 를 통해 멤버들이 데이터를 동기화 한다.
- read preference 를 통해 세컨더리 에서도 조회가 가능하다.

세컨더리(secondary)

- 프라이머리 유실시 세컨더리 멤버 중 선출을 통해 새로운 프라이머리를 정하게 된다.
    - promotion, step-up 이라 명명
- 세컨터리 중 read 할 멤버를 정할 수는 없지만 제외하는 것은 가능하다.(백업, 통계용으로 활용)

아비터(arbiter)

- 프라이머리 선출에는 관여하지만 데이터는 가지지 않는다.
    - oplog도 가져오지 않음
- 1 세컨더리 + 1 아비터 로 (p-s-a) 구성으로 많이 활용됨
    - 하나의 replicaset 은 과반수(majority) 이상 투표해야 프라이머리 선출 가능
- 아비터 들만 모아놓은 전용서버를 구축하기도 한다.

프라이머리 선출

프라이머리 선출 및 조건

- replica set 에서 프라이머리가 멤버가 없다고 판단될때 발생 한다.
- 프라이머리 선출은 투표권을 가진 최대 7개의 멤버에 의해 결정된다.
- 프라이머리는 전체 멤버 중 절반이상의 동의를 얻어야 하며 최소 2개 멤버의 동의를 얻어야 한다.(자신포함)

프라이머리 자격 반납

- replica set 에서 절반 이상을 동의 얻지 못하면 자격을 반납한다.
    - split-brain 방지
    - write operation 이 금지 된다.

프라이머리 선출 방식

- MongoDB 3.0 이전 (Protocol Version 0)
    - 각 서버가 인지하는 시각에 의존, 일정 시간 주기로 프라이머리 선출을 시도

- MongoDB 3.2 이후 (Protocol Version 1)
    - 논리적인 시간을 도입하여 짧은 주기로 프라이머리 선출

프라이머리 텀

- 프라이머리 선출 시도시마다 증가하는 논리적 시간값
    - protocol version1, version0 에는 primary term 개념이 없었다.
- oplog 에도 사용되어 어느시점의 데이터인지 유추 할 수 있다.

프라이머리 스텝 다운

- 관리자가 의도적으로 프라이머리를 교체하는 행위

```
// stepDownSecs: 다음 프라이머리가 선출되기까지 현재 프라이머리가 대기하는 시간
// 시간이 지나도 선출이 되지 않으면 다시 프라이머리가 된다.
// secondaryCatchUpPeriodSecs: 세컨더리가 oplog 를 다 적용할때까지 선출을 미루고 기다
리는 시간
rs.stepDown(stepDownSecs, secondaryCatchUpPeriodSecs)

// 내부적으로 stepDown 과 같지만 secondaryCatchUpPeriodSecs 가 적용되지 않음
var cfg = rs.conf()
cfg.members[1].priority = 2
rs.reconfig(cfg)

// replSetStepDown: stepDownSecs 동일
// secondaryCatchUpPeriodSecs: 동일
db.adminCommand( {
    replSetStepDown: 60,
    secondaryCatchUpPeriodSecs: 10,
    force: true
} )
```

프라이머리 선출시 정족수(Quorum)의 의미

프라이머리 선출시 투표권을 가진 멤버 중 정족수를 채워야 프라이머리에 선출 될 수 있다. (정확히 이해를 못함)

롤백

- 전제
    - transaction 의 rollback 과는 상관 없다.
    - replicaset 상황

```
# oplog 상황
prim: [1] [2] [3]
sec1: [1] [2]
sec2: [1]
```

```
# 1 prim 장애, [3] 은 아무도 못 받음
# 2 sec2 가 primary 로 선출되고 새로운 oplog 쌓음
sec1: [1] [2] [4]
sec2: [1] [2] [4]
# 3 prim 이 다시 참가
prim: [1] [2] [4] # [3] 을 삭제하며 이를 rollback 이라함
sec1: [1] [2] [4]
sec2: [1] [2] [4]
```

롤백된 데이터를 별도의 파일로 기록된다.

롤백 후 재처리

롤백된 데이터를 bsondump 를 이용해서 json 으로 변활 할 수있고 이를 재처리 할 수 있다.

복제 아키텍쳐

복제 로그(oplog) 구조

필드명	설명
ts(timestamp)	unix epoch 의 초단위, 두번째는 동시간대(초)의 논리적인 순서값
t(primary term)	primary term
h(hash)	random hash
v(version)	oplog document version
op(operation type)	i(insert),d(delete),u(update),c(command),n(no operation)
ns(namespace)	데이터베이스명과 컬렉션이름의 조합
o(operation)	변경내용
o2(operation 2)	update 시에만 대상 document 를 식별하기 위해 "_id" 값을 제공

local 데이터베이스

- 정의
  - 복제대상이 되지 않는 데이터베이스
  - 초기에 자동으로 생성된다.
  - 사용자도 collection 을 만들 수 있다.
  - 백업이력, 모니터링 등
  - oplog 도 local 데이터베이스에 저장되어 중복으로 이중화 되지 않는다.
  -

초기 동기화

새로운 노드를 리플리카셋에 처음 참가 시킬 경우 초기 적재를 하는 행위를 말한다.

- 수동 동기화

    - 완전한 세컨더리 하나를 셧다운하여 모든 데이터 파일을 수동 복사하여 진행하는 방식
    - 서비스가 운영중이라면 셧다운 이후 기록되는 oplog 를 가지고 있는 노드가 하나는 반드시 있어야 한다.
- 자동 동기화
    - 멤버를 추가해주면 데이터 부터 인덱스 까지 모두 복제하지만 속도가 매우 느리다.
- 특이점
    - 싱글스레드로 동작
    - 중간에 중지되면 처음부터 다시 시작

실시간 복제

(미준비)

리플리카 셋 설정

하트비트 메시지 주기와 프라이머리 선출 타입아웃

(미준비)

리플리카 셋 멤버 설정

(미준비)

리플리카 셋 배포

리플리카 셋 멤버의 수

(미준비)

DR(Disaster Recovery) 구성

(미준비)

리플리카 셋 배포시 주의 사항

(미준비)

WiredTiger

WiredTiger 스토리지 엔진 저장 방식

- 레코드 스토어
    - OLTP 용도에 적합
    - 일반적은 RDBMS 가 사용하는 저장 방식
    - 테이블의 레코드를 함께 저장
    - B-Tree
- 컬럼 스토어
    - 대용량의 분석 용도로 주로 사용 OLTP 용도로는 부적합
    - 컬럼 단위 혹은 컬럼의 그룹 단위로 파일을 분리해서 저장

- LSM
    - HBase 난 카산드라 같은 NoSQL 에서 자주 사용된다.
    - 데이터를 읽기 보다는 쓰기에 중점을 두었다.
    - LSM-Tree

MongoDB 에서는 현재 레코드 스토어만 지원한다.

데이터 파일 구조

WiredTiger

WiredTiger 스토리지 엔진 버전

```
sudo cat /var/lib/mongo/WiredTiger
```

storage.bson

WiredTiger의 디렉토리 구조를 설정하는 옵션의 내용이 저장 storage.bson 파일은 MongoDB 서버의 데이터 파일을 처음 생성했던 시점의 storage.deirectoryPerDB 옵션과 storage.wiredTiger.directoryForIndexes 옵션의 값을 저장

storage.deirectoryPerDB

WiredTiger 스토리지 엔진이 컬렉션의 데이터 파일을 어느 디렉터리에 위치시킬지 결정

storage.wiredTiger.directoryForIndexes

WiredTiger 스토리지 엔진이 컬렉션의 인덱스 저장 파일을 어느 디렉터리에 위치시킬지 결정 한번 데이터 파일이 초기화되면 변경할 수 없다.

sizeStorer.wt

메타 데이터 파일 WiredTiger 스토리지 엔진을 사용하는 컬렉션의 전체 도큐먼트 건수와 각 컬렉션의 데이터 파일 크기를 저장 특정 이유에서 정확한 컬렉션의 도큐먼트 건수를 보여주지 않을 수 있다. -> 조건을 포함하는 count() 명령으로 출력

wiredTiger.lock

MonogDB 서버가 사용하는 데이터 파일들에 대해 다른 MongoDB 서버 인스턴스가 동시에 사용하지 못하도록 잠금 역할을 하기 위한 파일 WiredTiger 스토리지 엔진이 정상적으로 셧다운 됐는지 판단하는 데 참조되는 파일

WiredTiger.turtle

WiredTiger 스토리지 엔진의 설정 내용을 담고 있다. 엔진의 옵션 설정 설정을 변경한 내용이 적용 됐는지 확인. (db.serverStatus로 확인할 수 있다.)

WiredTiger.wt

WiredTiger 스토리지 엔진의 메타 데이터를 저장하는 컬렉션의 데이터 파일

_mdb_catalog.wt

MongoDB 서버가 가지고 있는 WiredTiger 스토리지 엔진 컬렉션과 인덱스의 목록 그리고 각 인덱스나 컬렉션이 사용하는 데이터 파일의 목록을 관리하는 메타 데이터를 관리하는 파일

diagnostic.data

MongoDB 서버가 정보들을 1초에 한번씩 모아서 기록하는 파일을 저장하는 디렉토리

- 운영체제의 상태 정보(/proc/stats, /proc/meminfo, /sys/block/*/stat)
- serverStatus
- replSetGetStatus
- local.oplog.rs.stats 컬렉션의 collStas
- buildinfo
- getCmdLineOpts
- hostinfo

WiredTiger의 내부 작동 방식

읽기 작업

1. cursor 에 데이터 요청
    - cache 에 있는 경우 바로 읽고 종료
2. blockManager
    - 디스크에서 필요한 블록을 읽어서 cache 에 저장한다.
3. cache 의 데이터를 client 에 전달한다.

쓰기 작업

1. transaction 이 발생하면 cursor 를 이용하여 cache 에 내용을 기록한다.
2. 변경내용을 journal buffer 에 기록한다.
3. (optional) cache가 부족하면 eviction 을 통해 일부 페이지를 삭제하거나 디스크에 기록한다.
4. (optional) 조건 만족시 checkpoint 를 발생하여 더티 페이지를 디스크에 기록한다.
5. (optional) 더티 페이지는 원본+변경 분을 reconciliation 을 통해 병합한다.

journal

동작 방식

journal record 를 생성하여 변경된 collection 의 documents 와 관련 index 데이터를 기록한다. 모든 journal record 는 별도의 스레드에 의해 최대 128kb 크기로 buffering 되어 기록된다. journal record 가 buffer 에 기록되면 client 에 완료 ack 를 보낸다.

파일 기록 조건

다음 조건이 발생하는 경우 disk 에 바로 기록된다.

- checkpoint 발생시
- 지정된 interval (storage.journal.commitIntervalMs, default 100ms)
- journal 파일 최대 크기를 넘어 새로운 파일을 기록하는 경우
- replicaset 에서 oplog 에 대한 요청이 생기는 경우
- write concern 의 j 옵션을 true 로 하는 경우

journal 파일 관리

- journal 파일은 미리 생성할 수 있으며 로테이션 하지 않고 새로운 파일을 기록한다.
- checkpoint 시 불필요한 old 파일을 삭제하거나 archive 할 수 있다.

> mongodb 4.0 이상부터는 wiredtiger 엔진사용시 journal 을 비활성화 할 수 없다. (데이터 일관성 문제 for replicaset)

> mongodb 3.2 부터 client 의 쓰기 작업 마다 journal record 를 생성하도록 변경됨

공유 캐시

캐시 이빅션(Cache Eviction)

동작 방식

- 백그라운드의 별도의 스레드로 작동
  - 바쁘면 사용자 스레드도 eviction 을 같이 수행한다.
- cache 가 부족할때 최근에 가정 적게 사용된 페이지를 cache 로 부터 제거한다.
- 더티페이지 -> reconciliation 을 통해 디스크에 반영
- 일반페이지는 그냥 삭제
- checkpoint 와의 관계 ??

체크포인트(Checkpoint)

checkpoint 발생주기

- MongoDB driven
  - 서버 시작시 아래와 같은 옵션으로 wiredtiger 엔진에 옵션을 부여한다.
  - 60 sec interval
  - journal data 2gb 이상 기록되는 경우

checkpint 방식

- sharp checkpoint
  - 모아서 더티 페이지를 기록하는 패턴
  - interval 로 수행되는 fuzzy checkpoint 와는 상대적
  - 일지적인 disk i/o 증가로 jitter 의 원인이 될 수 있음

특이점

- 새로운 페이지 및 변경 페이지에 대해 처리 방식
  - 기존의 tree 를 수정하지 않음

    - 새로운 페이지를 할당받아서 기록
    - 기존 tree 의 노드를 새로운 tree 에 연결
    - 기존의 tree 를 대체 및 삭제
- 이로인해 일시적으로 최대 2배의 디스크 공간을 사용할 수 있음

MVCC(Multi Version Concurrency Control)

isolation level

wiredtiger 는 다음과 같은 격리수준을 제공한다.

- READ-COMMITTED
- READ-UNCOMMITTED
- SNAPSHOT(REAPATABLE-READ)

SNAPSHOT 을 기본으로 하며 설정에 따라 READ-UNCOMMITTED 로 할 수 있지만 READ-COMMITTED 는 설정할 수 없다.

데이터 블록(페이지)

특징

- 페이지의 크기는 가변적이다.
- 페이지의 최대 크기가 있다
    - internal 4k, leaf 32k
- collection 마다 지정할 수 있다.
- 데이터 페이지와 인덱스 페이지의 크기를 달리 할 수 있다.

예시

```
wiredTiger:
    engineConfig:
        cacheSizeGB: 20
    collectionConfig:
        configString: "internal_page_max=4k, leaf_page_max=64k"
    indexConfig:
        configString: "internal_page_max=4k, leaf_page_max=16k"
```

주의점

- insert 이후 대량의 조회가 있는 겨우
    - 페이지를 크게 설정
- 소규모의 데이터를 빈번하게 쓰고 읽는 경우
    - 페이지를 작게
- 공유 cache 의 효율성
    - b-tree 의 특성상 sequential 하지 않은 데이터 입력시
    - 페이지의 크기가 크다면 많은 디스크 i/o 가 발생하여 비효율적일 수 있다.

운영체제 캐시(페이지 캐시)

wiredTiger 엔진은 direct io, cached io 모두 지원하지만 mongodb 에서는 cached io 만 지원한다. direct io 를 사용하게되면 페이지 크기의 가변 정책에 맞지 않기 때문이다.

direct io

direct io 는 app 의 디스크 i/o 동작이 운영체제의 캐시를 거치지 않고 물리적인 디스크에 바로 동작하는 것을 말한다.

- 운영체제의 캐시 정책에 영향을 받지 않기 때문에 일정한 throughput 을 낼 수 있다.
- 가부화 상태에서 확연하게 차이가 발생한다.
- 일반적인 상태에서는 cached io 보다 많이 느리다.

cached io

cached io 는 app 의 디스크 i/o 동작이 운영체제의 캐시를 통해 동작하는 것을 말한다.

- 디스크에 대한 io 를 운영체제에 일임하기 때문에 일반적으로 direct io 보다 성능이 빠르다.
- 운영체제의 절대적인 영향을 받기 때문에 app 에서 이를 제어할 수 없다.

압축

장점

wiredTiger 엔지는 RDBMS 대비 다음과 같은 장점이 있으며 압축측면에서도 장점이 있다.

- 가변 사이즈의 페이지 사용
  - 압축 대상의 크기를 고려하지 않아도 된다.
- 데이터 입출력 레이어에서의 압축 지원
  - blockManager 에서 압축기능을 지원하기 때문에 wiredTiger 엔진의 코드가 가볍다
- 다양한 압축 알고리즘 지원
  - 데이터 블록(journal 포함)과 인덱스 블록에 대해 다양한 압축 알고리즘을 제공한다.

wiredtiger 에서 제공되는 압축

- 블록 압축(Block Compression)
  - zstd, snappy 알고리즘
- 인덱스 프리픽스 압축(Key Prefix Compression)
  - prefix 알고리즘
  - cache 안에서도 압축형태 유지
  - 재조립 과정으로 다소 성능 저하가 있기도하다.
  - index 를 역순으로 탐색시
- 사전 압축(Dictionary Compression, mongoDB 지원x)
- 허프만 인코딩(Huffman Encoding, mongoDB 지원x)

wiredTiger 압축 설정

```
wiredTiger:
    engineConfig:
        cacheSizeGB: 20
        journalCompressor: snappy
        collectionConfig:
            blockCompressor: snappy
        indexConfig:
            prefixCompression: true
```

운영체제 파일 캐시

wiredTiger 는 cached io 를 사용하게 되는데 index 의 경우 prefix 알고리즘을 사용하면 운영체제의 파일 캐시에도 prefix 가 적용된 상태로 io 가 발생하게 된다. 그래서 파일 캐시의 효율도 올라가는 효과가 있기도 하다.

암호화

- RDBMS 의 TDE(Transparent Data Encryption) 과 유사한 방식
- 엔터프라이즈 버전에서만 사용 가능
- 오픈소스 코드에 인터페이스가 있기 때문에 플러그인을 제작하여 사용자가 암호화를 할 수 있다.

그외의 스토리지

Reference

- https://docs.mongodb.com/manual/core/journaling/
- https://docs.mongodb.com/manual/reference/write-concern/#writeconcern.j
- https://docs.mongodb.com/manual/core/read-isolation-consistency-recency/
- https://docs.mongodb.com/manual/core/wiredtiger/
- https://docs.mongodb.com/manual/reference/glossary/#term-prefix-compression