KV 数据库性能测评

测试平台说明：

CPU: Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz
CPU cache size: 8192 KB
Memory: 16206236 kB
Python 版本：2.7

测评数据库说明：

Cassandra 版本：3.11.2
Cassandra-driver 版本：3.14.0
cql 版本：1.4.0
Redis 版本：4.0.9
Redis(python)版本：2.10.6
LevelDB 版本：0.194
BerkeleyDB（bsddb3）版本：6.2.5

目录

1 数据库的安装

注：以 MacOS 系统安装为主，不同系统可能操作略有不同。以 python 语言为主要编程语言，其他语言可能需要更多安装。

准备工作

安装 Homebrew

```
ruby -e "$(curl -fsSL
https://raw.github.com/Homebrew/homebrew/go/install)"
```

安装 Java

http://www.oracle.com/technetwork/java/javase/downloads/index.html

安装 python

```
brew install python
```

Cassandra

参考网址：https://gist.github.com/mars/a303a2616f27b46d72da
https://www.cnblogs.com/piaolingzxh/p/4197833.html
http://blog.sina.com.cn/s/blog_83dc494d0102vkfr.html

在 Terminal 中运行 Cassandra

1. **安装 cassandra**

```
brew install cassandra
```

2. **安装 cql**

```
pip install cql
```

3. **启动 cassandra**

   a. Terminal 1: $ Cassandra -f

      当显示如图时，打开另一个 tab

```
INFO  [main] 2018-06-12 11:19:24,148 CassandraDaemon.java:529 - Not starting RPC
 server as requested. Use JMX (StorageService->startRPCServer()) or nodetool (en
ablethrift) to start it
```

   b. Terminal 2: $ cqlsh

      Connected to Test Cluster at 127.0.0.1:9042.

      [cqlsh 5.0.1 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]

      Use HELP for help.

      cqlsh>

      退出：cqlsh> quit

4. **停止 Cassandra**

```
ps auwx | grep Cassandra
kill -9 <pid>
```

5. **Terminal 中测试：**

```
cqlsh> CREATE KEYSPACE demo WITH REPLICATION =
{ 'class' : 'SimpleStrategy', 'replication_factor' : 1
}; //创建 keyspace
cqlsh> DESCRIBE keyspaces //查看所有 keyspace
cqlsh> USE demo;  //进入 demo keyspace
cqlsh> CREATE TABLE users (id int, name text, primary
key(id));    //创建表格 users，包含两列（id 和 name），
primary key 为 id。
cqlsh> INSERT INTO users (id, name) VALUES (1002,
‘Test’);   //插入值
cqlsh> SELECT * FROM users    //查询值
```

```
 id   | name
1002| Test
```

在 **python** 中运行 **cassandra**

1. 安装 **python driver**：

   ```
   pip install Cassandra-driver
   ```

2. **python** 中测试：

```
 $   python
 >>> from cassandra.cluster import Cluster
 >>> session = cluster.connect()
 >>> session.execute("create KEYSPACE test_cassandra
WITH replication = {'class':
'SimpleStrategy', 'replication_factor': 1};")
 >>> session.execute("use test_cassandra")
 >>> session.execute("create table users(id int, name
text, primary key(id));")
 >>> session.execute("insert into users(id, name)
values(1003, 'Test');")
 >>> session = cluster.connect("test_cassandra")
 >>> rows = session.execute("select * from users;")
 >>> rows[0].id
 1003
 >>> cluster.shutdown()
```

CentOS7 安装请参考：

https://www.howtoforge.com/tutorial/how-to-install-apache-cassandra-on-centos-7/

Redis

参考网址：https://www.cnblogs.com/feijl/p/6879929.html

在 **terminal** 中运行 **Redis**：

1. 下载 **redis-4.0.10.tar.gz**：

   https://redis.io/

2. 安装：

```
tar -zxvf redis-4.0.10.tar.gz
mv redis-4.0.10.tar.gz /usr/local
cd /usr/local/redis-4.0.10
sudo make test
sudo make install
```

3. 配置：

```
sudo mkdir  /usr/local/redis-4.0.10/bin
sudo mkdir  /usr/local/redis-4.0.10/etc
sudo mkdir  /usr/local/redis-4.0.10/db
cp /usr/local/redis-4.0.10/src/mkreleasehdr.sh .
cp /usr/local/redis-4.0.10/src/redis-benchmark .
cp /usr/local/redis-4.0.10/src/redis-check-rdb .
cp /usr/local/redis-4.0.10/src/redis-cli .
cp /usr/local/redis-4.0.10/src/redis-server .
cp /usr/local/redis-4.0.10/redis.conf /usr/local/redis-
4.0.10/etc
```

4. 启动：

   a. Tab1: redis-server
   b. Tab2: redis-cli
      127.0.0.1:6379>
   c. 退出：127.0.0.1:6379> shutdown

5. 测试：

```
$ redis-cli
127.0.0.1:6379> keys *
(empty list or set)
127.0.0.1:6379> set foo bar
OK
127.0.0.1:6379> keys *
1) “foo”
127.0.0.1:6379> get foo
“bar”
```

在 **python** 中运行 **redis**

1. 安装 **redis**

   ```
   pip install redis
   ```

2. 测试

   ```
   $ python
   >>> import redis
   >>> r = redis.StrictRedis(host = ‘localhost’, port = 6379)
   >>> r.set(‘1002’, ‘test’)
   >>> r.get(‘1002’)
   ‘test’
   ```

3. 报错

   如果出现 redis.exceptions.ResponseError: MISCONF Redis is configured to save RDB snapshot 错误，则进入 redis-cli 设置：

   ```
   $ redis-cli
   127.0.0.1:6379> CONFIG SET dbfilename temp.rdb
   127.0.0.1:6379> CONFIG SET stop-writes-on-bgsave-error no
   ```

LevelDB

参考网址：https://blog.csdn.net/kwsy2008/article/details/52326204

安装 python 版本的 levelDB:

```
pip install leveldb
```

测试：

```
$   python
>>> import leveldb
>>> db = leveldb.LevelDB('./data') //新建数据库，没有则新建，有则打开。
>>> db.Put('foo', 'bar')  //存数据
>>> db.Get('foo')   //读数据
'bar'
```

Berkeley DB

参考网址：https://blog.csdn.net/wushanyun1989/article/details/75336389

1. **安装 Berkeley DB**

   http://www.oracle.com/technetwork/database/database-technologies/berkeleydb/downloads/index.html
   下载最新版本

```
tar -zxcf db-6.2.32.tar.gz
cd db-6.2.32/build_unix
../dist/configure --prefix=/opt/Berkeley
make
make install
```

如果需要重新创建：

```
make clear
make
make install
```

删除：

```
make uninstall
```

2. **python 安装 bsddb3**

```
BERKELEYDB_DIR=/opt/Berkeley/
YES_I_HAVE_THE_RIGHT_TO_USE_THIS_BERKELEY_DB_VERSION=yes
pip install bsddb3
```

3. **python 中测试**

```
$   python
>>> import bsddb3
>>> db = bsddb3.btopen('spam.db', 'c')
>>> for i in range(10): db2['%d'%i] = '%d'% (i*i)
>>> db['1']
1
>>> db.keys()
['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
```

4. **报错**

   a. 如果遇到<python.h>: no such file or directory, 则 brew install python-dev

2 数据库性能对比

准备工作：

1. 数据下载：
   http://101.236.63.184:12345/weibo/
2. 数据预处理：

```
cut -f5,6 weibo_freshdata.2018-05-17 > userid+nickname.txt (测试 1、5)
cut -f5,9 weibo_freshdata.2018-05-17 > userid+weiboid.txt （测试 3）
cat userid+weiboid.txt | awk 'BEGIN{last=0}, {if($1!=last) print $0,
last = $1}' > userid+uniq_weiboid.txt （测试 2）
Cut -f9,10 weibo_freshdata.2018-05-17 > weiboid+content.txt （测试 4）
```

测试：

1. string key，string value

用微博的 userid 和屏幕名（昵称）来入库，并随机选 10 万条依次查询得到结果，对比哪个数据库快。

数据库	时间（s）
Cassandra	201.789
Redis	9.79055190086
LevelDB	2.639
BerkeleyDB	3.661

由上表可见，LevelDB 在查找 string key 时用时最短。

2. int key，int value

用微博的 userid 和 unique 的微博 id 来入库，并随机选 10 万条依次查询得到结果，对比哪个数据库快。

数据库	时间（s）/报错
Cassandra	208.853
Redis	8.882
LevelDB	argument 1 must be string or buffer, not int
BerkeleyDB	Integer keys not allowed

由上表可见，LevelDB 和 BerkeleyDB(BTree)无法存入 int key。Redis 在查找 int key 时用时最短。

3. multiple-int key，int value；

用微博的 userid 和微博 id，一个 userid 可能对应多个 weiboid，对比哪个数据库支持一个 key 存多个 value

数据库	是否支持
Cassandra	Not support, overwrite
Redis	Not support, overwrite
LevelDB	Not support, overwrite
Berkeley DB	C 语言中支持。参考 https://web.stanford.edu/class/cs276a/projects/docs/berkeleydb/ref/am_conf/dup.html

由上表可见，四个数据库 python 中都不支持一个 key 对应多个 value。当一个 key 需要存多个 value 时，数据库中存在的 value 为最后一个存入的 value。

4. String key，长 value

用 weiboid 和微博内容入库，并随机选 10 万条依次查询得到结果，对比哪个数据库快。

数据库	时间（s）
Cassandra	424.539
Redis	29.134
LevelDB	18.726
BerkeleyDB	15.433

由上表可见，在读取长 value 时，BerkeleyDB 的表现最好。

5. 多线程并发测试

并发 10 个线程做测试 1，每个线程执行 1 万条查询任务。

数据库	时间（s）	提速（%）
Cassandra	149.06	35.37%
Redis	9.681	1.1316%
LevelDB	1.21	118.926%
BerkeleyDB	1.65	121.879%

由上表可见，在用多线程读取数据的时候，LevelDB 速度最快。除 Redis，其余数据库与单线程相比时间都明显减少，其中 BerkeleyDB 提速最明显。

6. 扫 key

从 kv 库中把全部 key 扫出来。

数据库	用法
Cassandra	select <key_name> from <table_name>
Redis	keys *
LevelDB	db.RangeIter(include_value = False)
BerkeleyDB	db.keys()

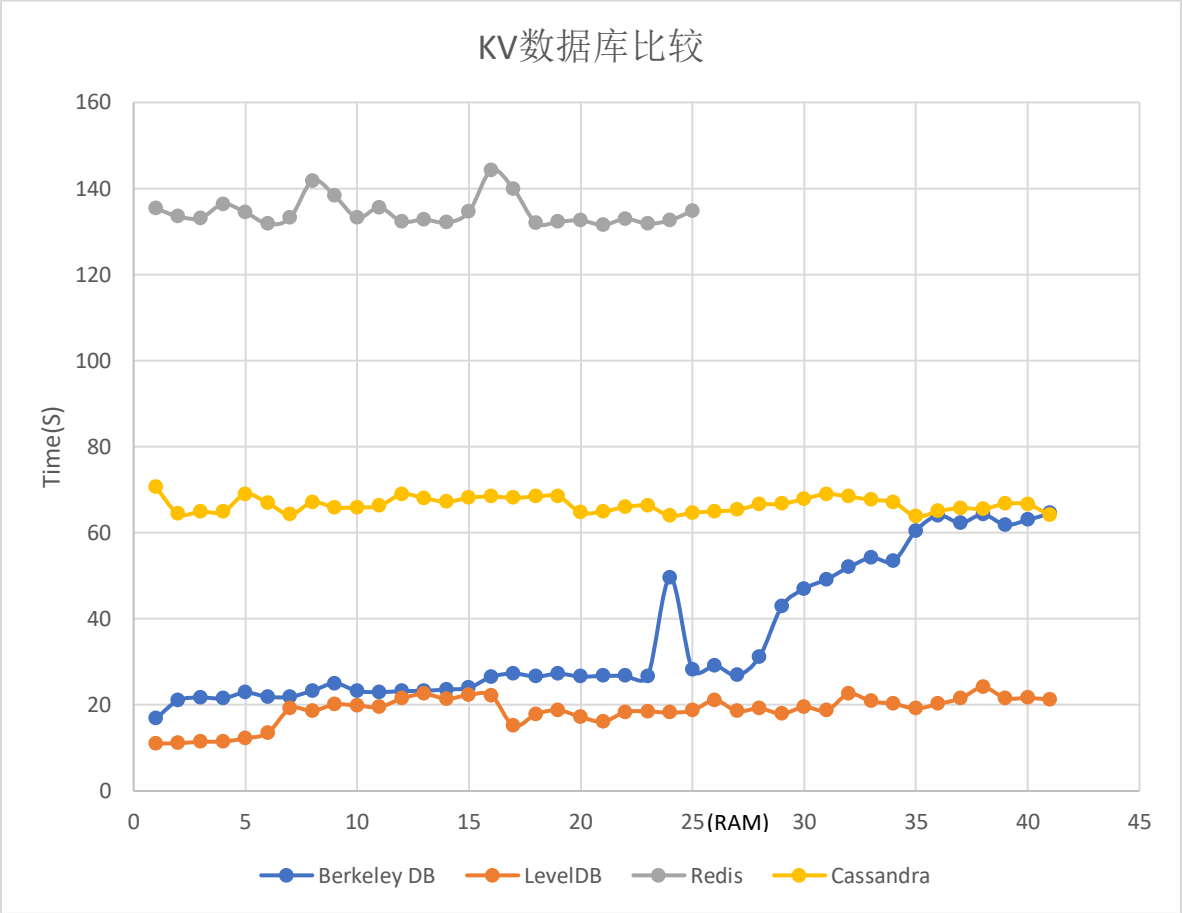
四个数据库都支持扫 key 的功能。

7．range-query

从 kv 库中把 key 在一个区间范围内的结果都找到。

数据库	用法（不支持则为 not support）
Cassandra	select * from <table_name> where <column_name> >/</= <number> ALLOW FILTERING
Redis	Not Support
LevelDB	Not Support
BerkeleyDB	Not SUpport

8．存储超过电脑内存数量的数据，对比各个数据库的性能。



由上图可见，当存储的数据量超过电脑内存时，Cassandra 和 LevelDB 并未有明显的变慢；BerkeleyDB 在超过内存后时间明显变长；Redis 则会在超过内存后自

动停止存储数据（报错：Can't save in background: fork: Cannot allocate memory）。

3 源代码

注：所有的文件不可以数据库的名字命名（如 cassandra.py 等）

1. **Cass.py**

```
import random
import time
import threading
from cassandra.cluster import Cluster

#连接 cassandra 服务器
cluster = Cluster(['127.0.0.1'])
session = cluster.connect()
session.execute("use userid_nickname")

#存数据（内存实验,循环多次）
cqlsh: user> COPY new_userid_nickname (id, name) from 'filename.txt' WITH
HEADER = FALSE AND DELIMITER = '\t';

#随机读取 100000 条数据
userfile = open('userids.txt', 'r')
u = []
for lines in userfile:
      u.append(int(lines))
ran = random.sample(u, 100000)

since = time.time()
for lines in ran:
            row=session.execute("SELECT * FROM users where id
= %s", (str(lines),))
            #print row[0].name
print time.time() - since

#多线程
userfile = open('userids.txt', 'r')
u = []
for lines in userfile:
      u.append(int(lines))
ran = random.sample(u, 10000)
print time.time()

class myThread (threading.Thread):
      def __init__(self, threadID, name):
            threading.Thread.__init__(self)
            self.threadID = threadID
```

```
            self.name = name
      def run(self):
            print "Starting " + self.name
            get_name(self.name, ran)
            print "Exiting " + self.name
            print time.time()

# f = open ('userid+nickname.txt', 'r')
def get_name(threadName,rand):
      for lines in rand:
            row=session.execute("SELECT * FROM users where id
= %s", (str(lines),))

thread1 = myThread(1, "Thread-1")
thread2 = myThread(2, "Thread-2")
thread3 = myThread(3, "Thread-3")
thread4 = myThread(4, "Thread-4")
thread5 = myThread(5, "Thread-5")
thread6 = myThread(6, "Thread-6")
thread7 = myThread(7, "Thread-7")
thread8 = myThread(8, "Thread-8")
thread9 = myThread(9, "Thread-9")
thread10 = myThread(10, "Thread-10")
thread1.start()
thread2.start()
thread3.start()
thread4.start()
thread5.start()
thread6.start()
thread7.start()
thread8.start()
thread9.start()
thread10.start()
```

2. **Red.py**

```
import redis
import random
import time
import threading
r = redis.StrictRedis(host='localhost', port=6379)

#存数据（内存实验）
f = open ('userid+nickname.csv', 'r')
users = []
names = []
for lines in f:
        u, n = lines.split("\t")
        users.append(u)
        names.append(n)
count = 0
while (count < 60):
        since = time.time()
        for i in range(len(users)):
                newuser = int(users[i]) * 100 + count
                r.set(str(newuser), names[i])
        count += 1
        print str(count) + ": " + str(time.time() - since)

#随机读取 100000 条数据
userfile = open('userids.txt', 'r')
u = []
for lines in userfile:
     u.append(int(lines))
ran = random.sample(u, 100000)
since = time.time()
for lines in ran:
          r.get(int(lines))
print time.time() - since

#多线程
userfile = open('userids.txt', 'r')
u = []
for lines in userfile:
     u.append(int(lines))
ran = random.sample(u, 10000)
since = time.time()

class myThread (threading.Thread):
     def __init__(self, threadID, name):
```

```
            threading.Thread.__init__(self)
            self.threadID = threadID
            self.name = name
      def run(self):
            print "Starting " + self.name
            get_name(self.name, ran)
            print "Exiting " + self.name
            print time.time() - since

def get_name(threadName, rand):
      since = time.time()
      for lines in rand:
            r.get(int(lines))

thread1 = myThread(1, "Thread-1")
thread2 = myThread(2, "Thread-2")
thread3 = myThread(3, "Thread-3")
thread4 = myThread(4, "Thread-4")
thread5 = myThread(5, "Thread-5")
thread6 = myThread(6, "Thread-6")
thread7 = myThread(7, "Thread-7")
thread8 = myThread(8, "Thread-8")
thread9 = myThread(9, "Thread-9")
thread10 = myThread(10, "Thread-10")

thread1.start()
thread2.start()
thread3.start()
thread4.start()
thread5.start()
thread6.start()
thread7.start()
thread8.start()
thread9.start()
thread10.start()
```

3. **Ldb.py**

```
import leveldb
import time
import threading
import random
db = leveldb.LevelDB('./data')

#存数据（内存实验）
f = open ('userid+nickname.csv', 'r')
users = []
names = []
for lines in f:
        u, n = lines.split("\t")
        users.append(u)
        names.append(n)
count = 0
while (count < 60):
        since = time.time()
        for i in range(len(users)):
                newuser = int(users[i]) * 100 + count
                db.Put(str(newuser), names[i])
        count += 1
        print str(count) + ": " + str(time.time() - since)

#随机读取 100000 条数据
userfile = open('userids.txt', 'r')
u = []
for lines in userfile:
     u.append(int(lines))
ran = random.sample(u, 100000)

since = time.time()
for lines in ran:
          db.Get(str(int(lines)))
print time.time() - since

#多线程
userfile = open('userids.txt', 'r')
u = []
for lines in userfile:
     u.append(int(lines))
ran = random.sample(u, 10000)

print time.time()
```

```
class myThread (threading.Thread):
      def __init__(self, threadID, name):
            threading.Thread.__init__(self)
            self.threadID = threadID
            self.name = name
      def run(self):
            print "Starting " + self.name
            get_name(self.name, ran)
            print "Exiting " + self.name
            print time.time()

def get_name(threadName, rand):
      since = time.time()
      for lines in rand:
            db.Get(str(int(lines)))
      time_elapsed = time.time() - since

thread1 = myThread(1, "Thread-1")
thread2 = myThread(2, "Thread-2")
thread3 = myThread(3, "Thread-3")
thread4 = myThread(4, "Thread-4")
thread5 = myThread(5, "Thread-5")
thread6 = myThread(6, "Thread-6")
thread7 = myThread(7, "Thread-7")
thread8 = myThread(8, "Thread-8")
thread9 = myThread(9, "Thread-9")
thread10 = myThread(10, "Thread-10")

thread1.start()
thread2.start()
thread3.start()
thread4.start()
thread5.start()
thread6.start()
thread7.start()
thread8.start()
thread9.start()
thread10.start()
```

4. **Bdb.py**

```
import bsddb3 as db
import time
import random
import threading
database = db.rnopen('recno.db', 'c')

#存数据（内存实验）
f = open ('userid+nickname.csv', 'r')
users = []
names = []
for lines in f:
        u, n = lines.split("\t")
        users.append(u)
        names.append(n)

count = 0
while (count < 60):
        since = time.time()
        for i in range(len(users)):
                newuser = int(users[i]) * 100 + count
                database[str(newuser)] = names[i]
        count += 1
        print str(count) + ": " + str(time.time() - since)

#随机读取 100000 条数据
userfile = open('userids.txt', 'r')
u = []
for lines in userfile:
     u.append(int(lines))
ran = random.sample(u, 100000)
since = time.time()
for lines in ran:
     print(database[str(int(lines))])
time_elapsed = time.time() - since
print time_elapsed

#多线程
userfile = open('userids.txt', 'r')
u = []
for lines in userfile:
     u.append(int(lines))
ran = random.sample(u, 10000)

print time.time()
```

```
class myThread (threading.Thread):
      def __init__(self, threadID, name):
            threading.Thread.__init__(self)
            self.threadID = threadID
            self.name = name
      def run(self):
            print "Starting " + self.name
            get_name(self.name, ran)
            print "Exiting " + self.name
            print time.time()

def get_name(threadName, rand):
      since = time.time()
      #file = open ('head_userid+nickname.txt', 'r')
      for lines in rand:
            result=database[str(int(lines))]
      time_elapsed = time.time() - since

thread1 = myThread(1, "Thread-1")
thread2 = myThread(2, "Thread-2")
thread3 = myThread(3, "Thread-3")
thread4 = myThread(4, "Thread-4")
thread5 = myThread(5, "Thread-5")
thread6 = myThread(6, "Thread-6")
thread7 = myThread(7, "Thread-7")
thread8 = myThread(8, "Thread-8")
thread9 = myThread(9, "Thread-9")
thread10 = myThread(10, "Thread-10")

thread1.start()
thread2.start()
thread3.start()
thread4.start()
thread5.start()
thread6.start()
thread7.start()
thread8.start()
thread9.start()
thread10.start()
```