通过 bilibili 数据集探究数据库管理系统的性能优势

Iydon Liang, 11711217

2020 年 3 月 29 日

目录

1 介绍

本实验的目的是通过 bilibili 数据集探究自己实现的数据库部分功能与 DBMS 之间的性能差异，同时比较不同 DBMS 之间的区别，最后利用 DBMS 提高数据的检索能力，加速数据分析。

2 实验设计

2.1 实验数据和环境

原先数据集爬取 bilibili 知名或有争议的 UP 主（具体见附录 A）全部视频下的全部评论，用以做水军等检测，因此数据库课程的 Project 可以拿来直接使用。但是个人觉得数据集不够完整，因此在数据库建模部分增加用户及视频表格，并通过外键将三张表关联起来。前期下载数据量有些大，用户数量约五百万，如果将全部用户信息爬下来时间较长，因此放弃。同时期间有被永久封禁的用户，现在已经找不到其个人信息，所以数据库建模时用户信息省略，仅保存其 uid。当然为了以后研究方便，并未取消用户表格，所以数据库中共有三张表格：`User`、`Video`、`Comment`。数据库建模可视化如图 1，同时利用数据库对象关系映射[1] 进行数据库建模，但是为进行跨文件系统的查询，因此有不符合范式的地方，数据库采用 PostgreSQL、MySQL 与 SQLite，操作系统采用 Linux，具体代码见附录 B。

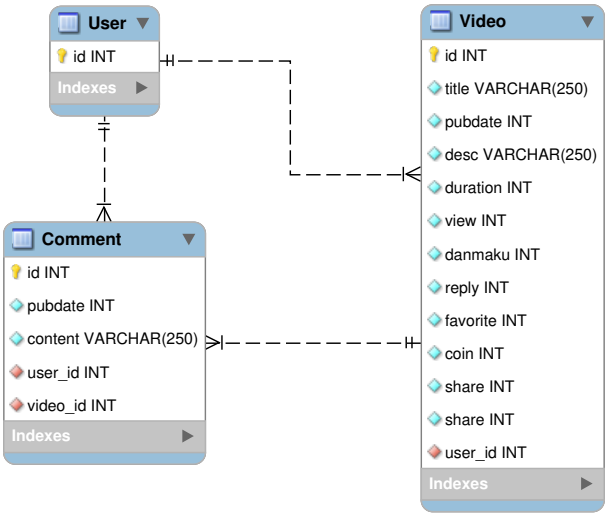

图 1: Workbench 数据库模型

所有数据分为三个文件夹，分别存储用户信息、视频信息以及评论信息，用户信息文件夹仅保存 UP 主的信息及出现在评论区全部用户的 uid，例如：

users/546195.json

```
1 {"name": " 老番茄", "sex": " 男", "face":
  ↪ "http://i2.hdslb.com/bfs/face/bc5ca101313d4db223c395d64779e76eb3482d60.jpg",
  ↪ "sign": " 新浪微博：_ 老番茄 _", "level": 6, "birthday": "08-13",
  ↪ "archive_view": 909097528, "article_view": 0, "likes": 37261387,
  ↪ "following": 1, "follower": 9548766}
```

视频信息文件夹保存全部视频的信息，例如：

videos/av84887919.json


```
{"pic":
"http://i2.hdslb.com/bfs/archive/202bc40ecf4991d21df91f52a2d112eddb977de1.jpg",
"title": " 最强自夸王!!!!! ", "pubdate": 1579924812, "desc": " 歌
曲:《自夸小队》\n作词/演唱:某幻君,老番茄,中国 boy 超级大猩猩,花少
北\nHOOK:茶理理理子\n编曲:Lglywww \n混音:Blue coat\n摄影:藤井旋风,大
饼\n后期:藤井旋风", "duration": 231, "owner": 1577804, "view":
11436611, "danmaku": 224768, "reply": 26880, "favorite": 530197,
"coin": 989930, "share": 130342, "like": 956804}
```


评论信息文件夹按照视频通过嵌套字典保存保存全部评论信息，第一层的键为用户的 uid，第二册的键为时间戳，第二层的值为评论内容，后期更新代码可以保存评论的全部信息，但是本次 Project 仅考虑用户及时间戳，例如：

comments/av3051327.json


```
{"585481": {"1444662778": " 作为天天刷太平洋的刷子团,今天在运送车队这个准备
任务上载跟头了,来看看妹子的抢劫放松一下"}, "10917828": {"1444665830": "
这期有点恶意卖萌 不过还好 反正我看了"}, "8872386": {"1444671846": "
么么哒 (` · ·´)"}, "342211686": {"1581531133": " 考古 [吡牙][吡牙]"},
"29407696": {"1530278286": " 考古"}, "3999038": {"1444689968": " 来顶
(づ ) づ"}, "7266479": {"1444661483": "44444444 先投币点赞在说"},
"495569": {"1444661357": " 第三 (` · ·´)"}, "5713034": {"1444659325":
" 第二 ^_^"}, "2812699": {"1444658015": " 第一 ( )"}, "6306456":
{"1497997152": " ( - - ) "}, "15514305": {"1493999905": " 考古 (= · ·
=) 感觉每天就指紫雨视频活了"}, "10954013": {"1547426274": " 🄵 🄵
"}, "74234761": {"1535518362": " 考古"}}
```


数据库中的用户表格共有 5890919 行数据，视频表格共有 24742 行数据表格，评论共有 34097426 行数据，纯文本数据共有 3.7 G。全部的程序见 GitHub，至于 bilibili 工具类例如兼容 bv 索引的爬虫见 GitHub，预计 2020 年 4 月 1 日开源。代码使用 Python 语言，通过 pipenv 进行虚拟环境管理，具体见附录 A。

2.2 实验细节

时间记录均为多次实验（5 次），后对时间取平均值，减少误差。

首先进行比较 DBMS 与 RAM，根据 Python 中 Hashmap 的设计[2] 以及实际的计算，发现 `Video` 表格可以导入内存，在硬盘中因为由许多小文件，所以大小为 98M。Python 中因为涉及到指针，使用 `sys.getsizeof` 计算对象大小会造成误差，因此通过递归方式对原对象解构，得出内存中数据大小为 22M，同时以 `Video.id` 为数据库索引。实验将 IO 读取与 RAM 索引进行分别计时，但是由于爬取的数据为 Json 格式，解析耗时较长，因此预处理为 CSV 格式，同时实现序列化 CSV 格式文件的功能（堵塞方式读取），提升效果显著。其中 SQL 语句为 `select * from Video as v where v.title like "%substr%"`，其中 `substr` 可以为任意关键词，本次实验选取 bilibili 标题中较容易火的关键词：“震惊”、“内幕”、“锤”、“抑郁”、“盗”、“赞”。最后结果如图 2。

接下来进行不同 DBMS 之间的比较，包括 PostgreSQL、MySQL 与 SQLite。因为采用 ORM 进行数据库建模，因此在不同数据库之间转换较为容易，可以通过程序进行比较而非手动运行并记录数据。本次实验选取了五个方面：检索小数据库（`Video`）、检索大数据库（`Comment`）、使用关联对象、聚合函数以及分组。最后结果如图 3。

最后的实验则测试实现相同功能的 Python 程序与 DBMS 之间的区别，具体可以见第 4 节，因为前些天电脑出问题重装，原先程序丢失，所以并无对比，但是直观可以感受到 DBMS 的效率要远高于相同功能的程序，且不需要进行复杂的逻辑判断，只需要构造相应的 SQL 语句即可。

2.3 实验结果

第一次实验虽然 DBMS 性能略差于直接使用 RAM 读取数据，但是它们之间进行比较是不公平的，因为 DBMS 每次得到请求会通过硬盘索引数据，所以 RAM 读取数据的结果应加上 IO 消耗的时间，同时考虑到自己的程序只对单个 CSV 格式文件进行解析序列化，如果增加复杂的关系等，拓展性是不如 DBMS，因此 DBMS 更为出色。

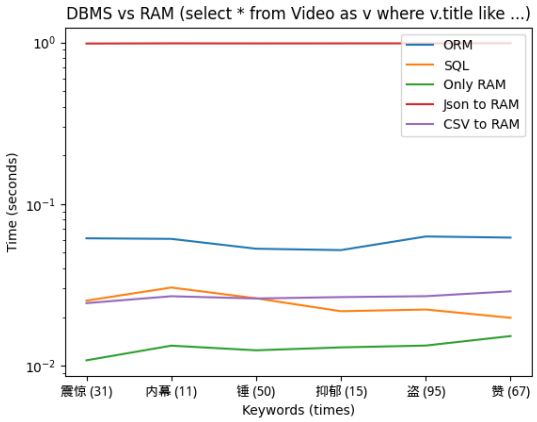

图 2: DBMS 与 RAM 比较

第二次实验在 98M 与 3.4G 的数据量上可以看出大概的趋势，即 PostgreSQL 较优于 SQLite 较优于 MySQL。这个结果有些不符合直觉，因为 SQLite 将所有数据保存为一个文件，同时本次实验只进行索引，可能这个数据量小于 MySQL 的优势区间。但是 SQLite 的劣势显而易见：缺乏用户管理与性能优化的灵活性。

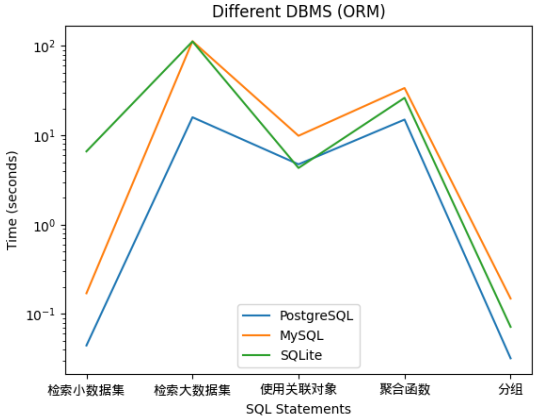

图 3: 不同 DBMS 比较

3 结论

使用 DBMS 对持久化数据进行管理有非常大的优势，不仅其管理数据速度快，而且其响应时间较小。同时 DBMS（除报告中使用的 SQLite）可以通过事务管理对用户数据进行保护，避免低权用户操作敏感数据。同时参考相应的文献可以得到 DBMS 的其他优势，例如并发能力、降低数据冗余、事务管理（Transaction Management）等。

4 附加内容

4.1 提及到的 Bonus

1. 用户权限管理的思路是通过监控服务器某一端口（例如 42926），通过异步 IO 与高并发的方式，将不同用户的请求分为不同的协程，验证由安全散列算法生成的密码散列确定其权限，由服务器判断是否将数据返回给用户；

2. 本实验的数据集约为 3.7G，其中 `Comment` 约为 3.4G，前面的实验通过检索大数据集比较不同 DBMS 性能差异时使用了不能完全载入内存的数据。

4.2 提高检索能力，加速数据分析

由于报告作者最近兼职在 bilibili 做 UP 主，因此经常对其进行相应的数据分析，通过 DBMS 可以快速获取不同视频的点赞、投币、收藏的数据，并对其进行分析，如图 4 与图 5。初步得出以下结论：

1. 大概 2017 年之前点赞等更“值钱”一些（这里的”值钱“指的是当时大家看重推荐视频的权利，或者是有些数据丢失了造成了我的错觉，前者可能的解释是当时部分 UP 主是课余兴趣才投稿的，随着用户基数增大变成盈利手段，号召一键三连，这句话没有负面意思毕竟人是要吃饭的）；

2. 现在的部分知名 UP 主也是从早期粉丝少通过不断更新积累粉丝（发现知名 UP 主部分远古稿件的点赞等数据不好看，而且大部分人为考古，比如老番茄、中国 BOY 等，与现在有些

“UP” 短期粉丝快速增长活跃度低形成对比，这句话也没有负面意思毕竟有名气了到哪里粉丝都会关注）；

3. 不适当的评论或多来自于三观容易受影响的未成年人（简单看了公开资料，部分“不适当”的评论或来自于未成年人，当然可能也包括心智未成熟的部分成年人，部分 UP 主可能也没有注意到自己的视频中或多或少出现具有误导性、带“情感色彩”词汇，然后由观看者进行进一步加工，理解成为自己想要的意思，进而产生不适当言论，毕竟刷视频多了，分配到一个视频的思考时间与能力会相应减少，而且视频相较文本信息更容易接受）。

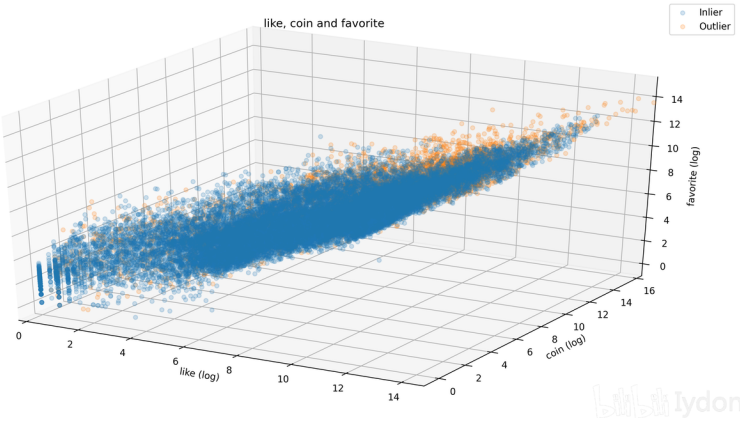

图 4: 点赞 + 投币 + 收藏直观展示图及离群值

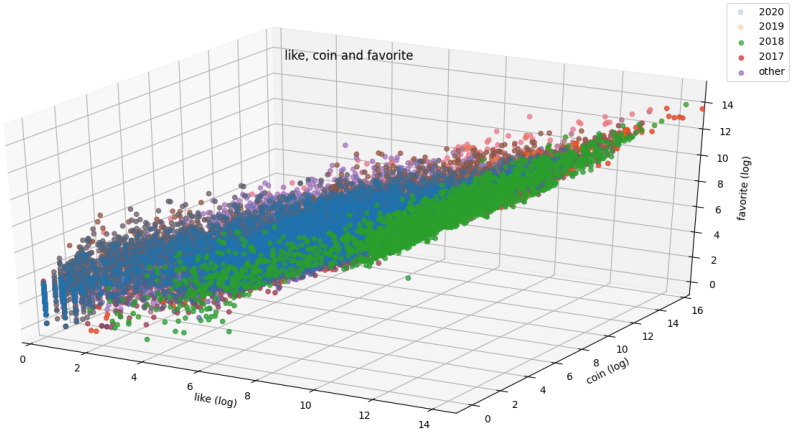

图 5: 不同年份点赞 + 投币 + 收藏直观展示图

同时最近也尝试做更进一步的研究，敬请期待，详情请参考 GitHub。

5 参考文献

[1] AUTHORS S, Contributors. The Python SQL Toolkit and Object Relational Mapper[Z]. https://www.sqlalchemy.org/. Accessed March 11, 2020. 2020.

[2] REES G. Python's underlying hash data structure for dictionaries[Z]. https://stackoverflow.com/questions/4279358/pythons-underlying-hash-data-structure-for-dictionaries. Accessed November 25, 2010. 2010.

[3] WATT A. Chapter 3 Characteristics and Benefits of a Database[Z]. https://opentextbc.ca/dbdesign01/chapter/chapter-3-characteristics-and-benefits-of-a-database/. Accessed August 24, 2014. 2014.

[4] CASTRO K. Advantages of Database Management System[Z]. https://www.tutorialspoint.com/Advantages-of-Database-Management-System. Accessed July 25, 2018. 2018.

[5] Vianzhang. BTree 和 B+Tree 详解[Z]. https://www.cnblogs.com/vianzhang/p/7922426.html. Accessed November 29, 2017. 2017.

[6] CENTER O H. Transaction Management[Z]. https://docs.oracle.com/cd/B19306_01/server.102/b14220/transact.htm. Accessed March 29, 2020. 2020.

A 配置文件

Pipfile

```
[[source]]
name = "pypi"
url = "https://pypi.mirrors.ustc.edu.cn/simple"
verify_ssl = true

[dev-packages]

[packages]
sqlalchemy = "*"
ipython = "*"
requests = "*"
psycopg2 = "*"
faker = "*"
fire = "*"
tqdm = "*"
retrying = "*"
pymysql = "*"
matplotlib = "*"
cryptography = "*"

[requires]
python_version = "3.8"
```

config.py

```
#!/usr/bin/python3
import getpass


_prompt = 'Which database (1: PostgreSQL, 2: MySQL, 3: SQLite) >> '
choice = {
    '1': ('PostgreSQL', 'postgresql', 'iydon', '5432'),
    '2': ('MySQL', 'mysql+pymysql', 'root', '3306'),
}.get(input(_prompt), None)
if choice:
    database_password = getpass.getpass('({} Password) >>> '.format(choice[0]))
    database_path = f'{{}}://{{}}:{database_password}@localhost:{{}}/bilibili' \
```

```
        .format(*choice[1:])
else:
    database_path = 'sqlite:////data/bilibili.sqlite'
string_max_len = 250

user_ids = 25876945, 121225592, 10909041, 9824766, 36825256, 2375158,
    ↪ 164139557, 437316738, 436000615, 472747194, 17819768, 1654470,
    ↪ 10330740, 32786875, 258150656, 496085430, 378885845, 350905839,
    ↪ 19577966, 32820037, 1791101, 486287787, 546195, 278761367, 96081167,
    ↪ 37663924, 80304, 2381918, 17873487, 163637592, 279583114, 1577804,
    ↪ 562197, 2206456, 318223, 481393564, 63231, 254726274, 124735327,
    ↪ 14110780, 8047632, 2374194, 265059948, 8366990, 116683, 375375,
    ↪ 297242063, 40433405, 333849444, 7792521, 67141499, 1754707, 888465,
    ↪ 22500342, 125526, 360739161, 5294454, 37199377, 178135921, 5687194,
    ↪ 39298350, 21448599, 89595, 52982448, 246370149, 438880209, 442184180,
    ↪ 478904588, 51896064, 18690024, 27880221, 168064909, 203581440,
    ↪ 1918296, 250681504, 585267, 290738125, 9064879, 433351, 7349, 7150454,
    ↪ 223958603, 1532165, 467942, 30625977, 26463792, 123938419, 28266043,
    ↪ 8261480, 16529853, 11984956, 10243458, 4766804, 225558766, 346353480,
    ↪ 434376696, 406595597, 7788379, 52250, 513811800, 83228795, 4898834,
    ↪ 168064909, 471300508, 510856133, 3530725, 320491072, 389988163,
    ↪ 291939565, 883968, 483935679, 168552156, 119801456, 54992199,
    ↪ 14583962, 479409514, 312177468

comment_path = '/data/bilibili/comments'
user_path = '/data/bilibili/users'
video_path = '/data/bilibili/videos'
```

B 数据库建模

database.py

```
#!/usr/bin/python3
__all__ = ('session', 'User', 'Video', 'Comment')


from sqlalchemy import Column, String, Integer, ForeignKey
from sqlalchemy import create_engine
from sqlalchemy.orm import relationship, sessionmaker
from sqlalchemy.ext.declarative import declarative_base
```

```
9
10  from config import database_path, string_max_len
11
12
13  Base = declarative_base()
14
15
16  String = String(string_max_len)
17
18
19  class User(Base):
20      __tablename__ = 'user'
21
22      id = Column(Integer, primary_key=True)
23      videos = relationship('Video', back_populates='user')
24      comments = relationship('Comment', back_populates='user')
25
26      # 前期下载数据量有些大，所以这里暂时注释掉
27      # name = Column(String, nullable=True)
28      # sex = Column(String, nullable=True)
29      # sign = Column(String, nullable=True)
30      # level = Column(Integer, nullable=True)
31      # archive_view = Column(Integer, nullable=True)
32      # article_view = Column(Integer, nullable=True)
33      # likes = Column(Integer, nullable=True)
34      # following = Column(Integer, nullable=True)
35      # follower = Column(Integer, nullable=True)
36
37
38  class Video(Base):
39      __tablename__ = 'video'
40
41      id = Column(Integer, primary_key=True)
42      user_id = Column(Integer, ForeignKey('user.id'))
43      user = relationship('User', back_populates='videos')
44      comments = relationship('Comment', back_populates='video')
45
46      title = Column(String, nullable=False)
47      pubdate = Column(Integer, nullable=False)
48      desc = Column(String, nullable=False)
49      duration = Column(Integer, nullable=False)
```

```
50      view = Column(Integer, nullable=False)
51      danmaku = Column(Integer, nullable=False)
52      reply = Column(Integer, nullable=False)
53      favorite = Column(Integer, nullable=False)
54      coin = Column(Integer, nullable=False)
55      share = Column(Integer, nullable=False)
56      like = Column(Integer, nullable=False)
57
58
59  class Comment(Base):
60      __tablename__ = 'comment'
61
62      id = Column(Integer, primary_key=True)
63      user_id = Column(Integer, ForeignKey('user.id'))
64      user = relationship('User', back_populates='comments')
65      video_id = Column(Integer, ForeignKey('video.id'))
66      video = relationship('Video', back_populates='comments')
67
68      pubdate = Column(Integer, nullable=False)
69      content = Column(String, nullable=False)
70
71
72  engine = create_engine(database_path)
73  Base.metadata.create_all(engine)
74  DBSession = sessionmaker(bind=engine, autoflush=False)
75  session = DBSession()
76
77
78  if __name__ == '__main__':
79      '''
80      session.add(...)
81      session.commit()
82      session.close()
83
84      print(session.query(...).all())
85      print(session.query(...).filter(...).first())
86      '''
87      u = User(
88          id=8888, name='XXXX', sex='保密', sign='hello world',
89          level=5, archive_view=100, article_view=100,
90          likes=100, following=100, follower=100,
```

```
91      )
92      v = Video(
93          id=1024, user_id=8888, title='测试标题', pubdate=1496979918,
94          desc='懒', duration=233, view=100, danmaku=100, reply=100,
95          favorite=100, coin=100, share=100, like=100,
96      )
97      c = Comment(
98          user_id=8888, video_id=1024, pubdate=1496979888, content='前排',
99      )
```