УДК 004.657

Е. Е. Доминов
магистрант кафедры вычислительных систем и сетей
А. В. Аксенов – старший преподаватель – научный руководитель

НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ БИБЛИОТЕКИ ОБЪЕКТНО-РЕЛЯЦИОННОГО ОТОБРАЖЕНИЯ МОДИФИЦИРОВАННЫМ МЕТОДОМ TPC-C

При написании приложений на Python, для работы с базами данных часто используются объектно-реляционные мапперы (ORM). Примерами ORM для Python являются SQLALchemy, PonyORM и объектно-реляционный маппер, входящий в состав Django. При выборе ORM довольно важную роль играет её производительность.

В интернете можно найти большое количество тестов производительности. В основном данные тесты оценивают скорость выполнения различных типов запросов за определенный промежуток времени. В качестве примера таких тестов можно привести бенчмарк от Tortoise ORM [1]. Данный бенчмарк анализирует скорость работы шести ORM для одиннадцати различных видов SQL-запросов.В целом бенчмарк от tortoise хорошо позволяет оценить скорость выполнения запросов при использовании разных ORM, но у такого подхода к тестированию существует одна проблема. ORM зачастую используют в веб приложениях, где одновременно несколько пользователей могут посылать различные запросы, но не было найдено ни одного бенчмарка, оценивающего работу ORM при таких условиях. Вследствие этого было решено написать свой тест, который бы оценивал работу ORM при таком сценарие.

Компания TPC с 1988 года разрабатывает тесты, направленные на обработку данных. Они давно стали индустриальным стандартом и используются почти всеми вендорами оборудования на различных образцах аппаратного и программного обеспечения. Главная особенность этих тестов состоит в том, что они нацелены на тестирование при огромной нагрузке в условиях, максимально приближенных к реальным.

TPC-C симулирует работу сети складов. Он включает в себя комбинацию из одновременно выполняемых транзакций пяти различных типов и сложности. База данных состоит из девяти таблиц с большим количеством записей. Производительность в тесте TPC-C измеряется в транзакциях в минуту.

Описание теста

В написанном мной тесте сначала создается и наполняется база данных, которая представляет из себя базу сети складов. Схема БД представлена на рис. 1

База данных состоит из восьми отношений:

1. Warehouse – склад
2. District – участок склада
3. Order – заказ
4. OrderLine – строка заказа (позиция заказа)
5. Stock – количество определенного товара на определенном складе
6. Item – товар
7. Customer – клиент
8. History – История платежей клиента.

В ходе теста, обрабатываются транзакции, посылаемых одновременно от лица нескольких виртуальных пользователей. Каждая транзакция состоит из нескольких запросов. Всего в данном тесте существует пять видов транзакций, которые подаются на обработку с разной процентной вероятностью:

1. new_order (создание нового заказа) 45 %
2. payment (оплата клиентом заказа) 43 %
3. order_status (возвращает состояние последнего заказа клиента) 4 %

4. delivery (доставка заказов) 4 %
5. stock_level (возвращает остаток на складе заказанных предметов) 4 %

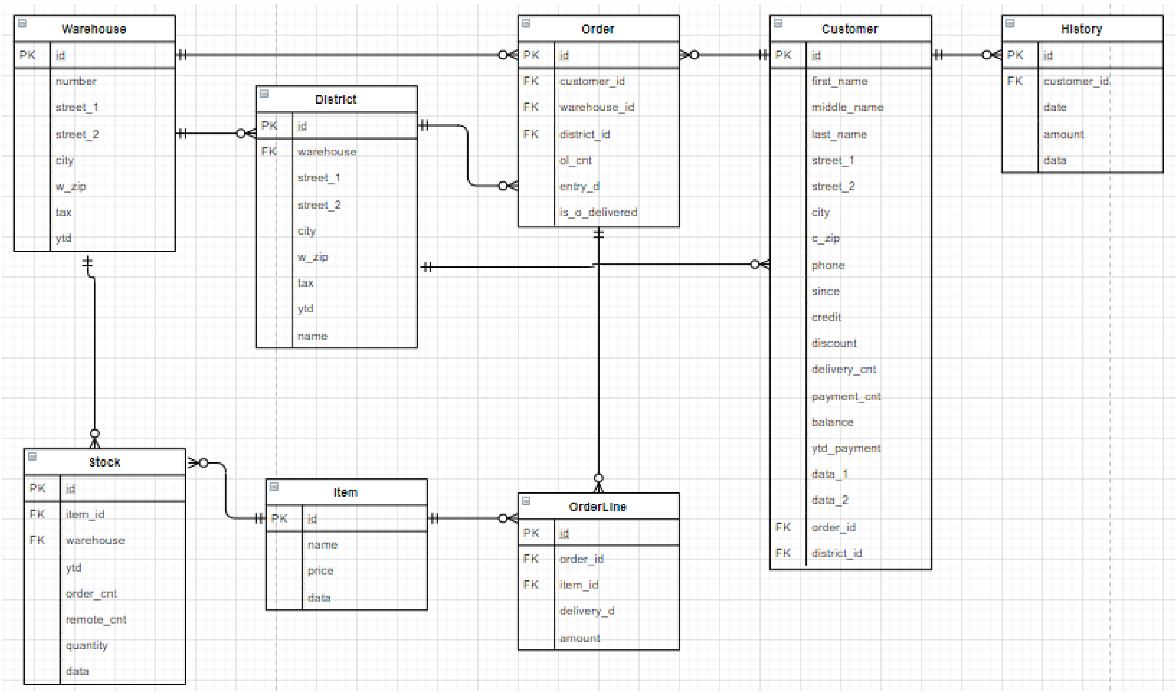

Рис. 1. Схема тестовой базы данных

Вероятность появления транзакций такая же, как и в оригинальном тесте TPC-C.

По сравнению с оригинальным тестом TPC-C данный тест несколько упрощен, в связи с техническими ограничениями и тем, что стоит цель проверить производительность ORM, а не устойчивость железа к нагрузке. Оригинальный тест проводится на вычислительных машинах, имеющих не менее 64 гигабайт оперативной памяти.

Основные различия:

1. Тест запускается с меньшим количеством виртуальных пользователей, чем в оригинальном тесте. В данный момент оборудование позволяет запустить от 2 до 5 виртуальных пользователей

2. Меньше количество записей в таблицах. Например: количество записей в отношении Stock в оригинальном тесте рассчитывается по формуле 100 000 * W, где W – это количество складов, а в предлагаемом тесте: 100 * W
3. В оригинальном тесте некоторые из 5 транзакций имеют несколько вариантов запроса данных из базы. Например в транзакции Payment с одной вероятностью клиент будет запрашиваться из базы по ID, с другой по фамилии и имени. На данный момент в предлагаемом тесте в подобных ситуациях вызов производится только по ID, в дальнейшем планируется реализовать и второй вариант.
4. В схеме данного теста отсутствует одна таблица, которая есть в оригинальном тесте. В оригинальном тесте, когда создается заказ, то он добавляется и в таблицу Order, и в таблицу NewOrder. После доставки заказа, он удаляется из таблицы NewOrder. Это ускорит работу при огромном количестве транзакций в минуту, но так как у в данном тесте меньше потоков и меньше единовременных транзакций, обращающихся к базе, то это излишне. Вместо этого в таблице Order был добавлен boolean атрибут “is_o_delivered”, который будет равен False, до тех пор, пока заказ не доставят.

Далее будет кратко описано, что делает каждая транзакция.

Транзакции

New Order

Краткое описание алгоритма:

1. Транзакции подается два аргумента: id клиента и id склада
2. Из базы запрашиваются склад и клиент по переданным id
3. Случайным образом берется один из участков склада
4. Генерируется случайное число строк(позиций) заказа
5. Создается объект заказа
6. В цикле создаются объекты для позиций данного заказа. На каждой итерации цикла из базы данных случайным образом берется товар из таблицы Item
7. Для каждого товара в заказе, в базе данных изменяется его доступное количество на складе

Payment

Краткое описание алгоритма:

1. Транзакции подается два аргумента: id клиента и id склада
2. Из базы данных запрашиваются склад и клиент по переданным id
3. Случайным образом берется один из участков склада и сумма оплаты
4. Баланс склада и отдельного участка увеличивается на сумму оплаты
5. Баланс клиента уменьшается на сумму оплаты
6. Счетчик количества оплат у клиента увеличивается на 1
7. Суммарная сумма, полученного от данного клиента, увеличивается на сумму оплаты
8. Создается объект истории платежей.

Order Status

Краткое описание алгоритма:

1. Транзакции подается id клиента
2. Из базы данных берутся клиент и его последний заказ
3. Из заказа берутся его статус (доставлен он или нет) и позиции заказа

Delivery

Краткое описание алгоритма:

1. Транзакции подается id склада
2. Из базы запрашиваются склад по id и все его участки
3. Для каждого участка берется самый старый из не доставленных заказов. В каждом из них статус доставки меняется на True
4. Из базы данных берутся пользователи, чьи заказы были доставлены в ходе данной транзакции, и у каждого из них увеличивается счетчик доставок

Stock Level

Краткое описание алгоритма:

1. Транзакции подается id склада
2. Из базы данных запрашивается склад по id
3. Из базы данных запрашиваются последние 20 заказов этого склада
4. Для каждой позиции этих заказов из базы данных запрашиваются кол-во остатка товара на складе

Результаты тестирования

В тестировании участвуют два ORM:

1. SQLAlchemy. На графиках изображен синей линией
2. PonyORM. На графиках изображен желтой линией

Ниже приведены результаты запуска теста на 10 минут с 2 параллельными процессами, обращающимися к базе. Процессы реализованы с помощью модуля multiprocessing. В качестве СУБД используется PostgreSQL. На рисунках ниже изображены в виде графиков результаты запуска теста на каждой из пяти типов тестовых транзакций и для смеси пяти транзакций. На графиках ниже ось X обозначает время в минутах, ось Y – количество транзакций выполненных за минуту.

Запуск со всеми транзакциям

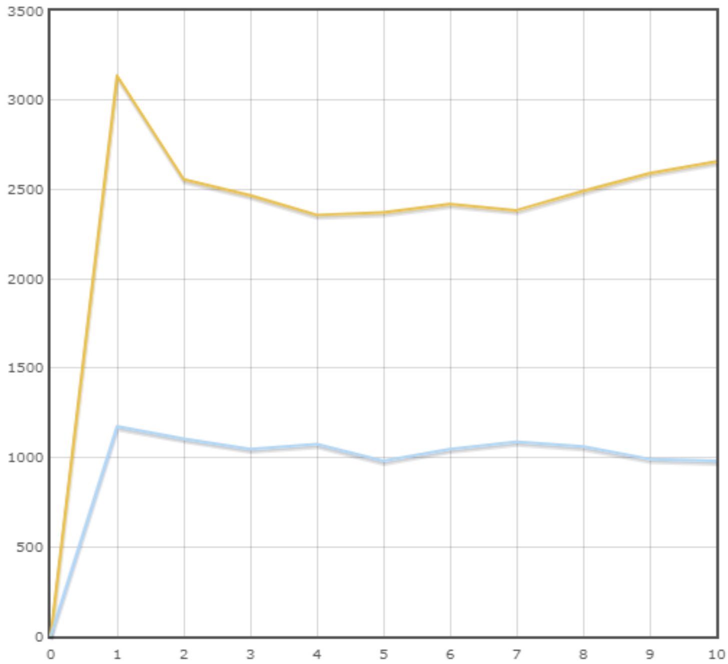

Рис. 2. Запуск со всеми транзакции

Средняя скорость:
1. PonyORM – 2543 тран/мин
2. SQLAlchemy – 1055.8 тран/мин

Транзакция «New Order»

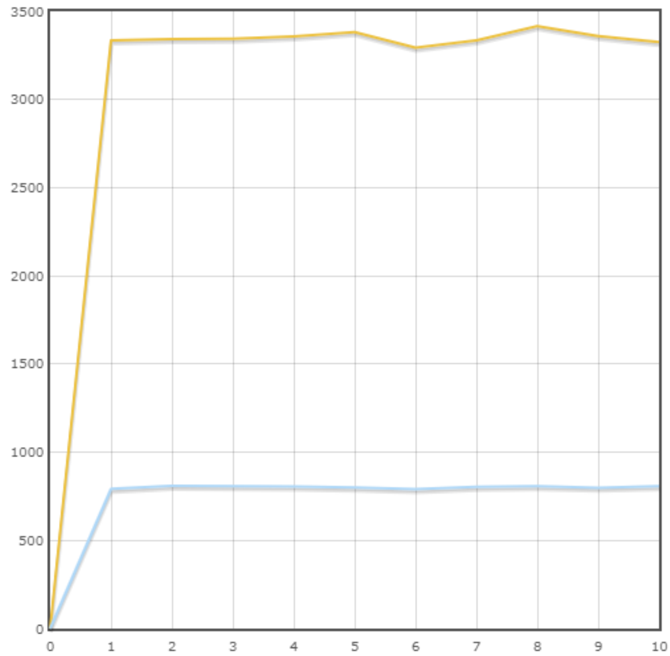

Рис. 3. Транзакция «New Order»

Средняя скорость:
1. PonyORM – 3349.2 тран/мин
2. SQLAlchemy – 802.1 тран/мин

Транзакция «Payment»

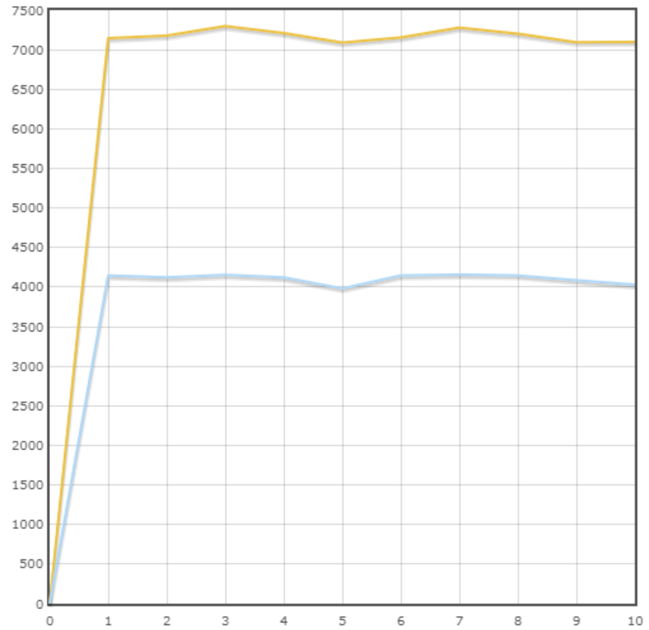

Рис. 4. Транзакция “Payment”

Средняя скорость:

1. PonyORM – 7175.3 тран/мин
2. SQLAlchemy – 4110.6 тран/мин

Транзакция «Order Status»

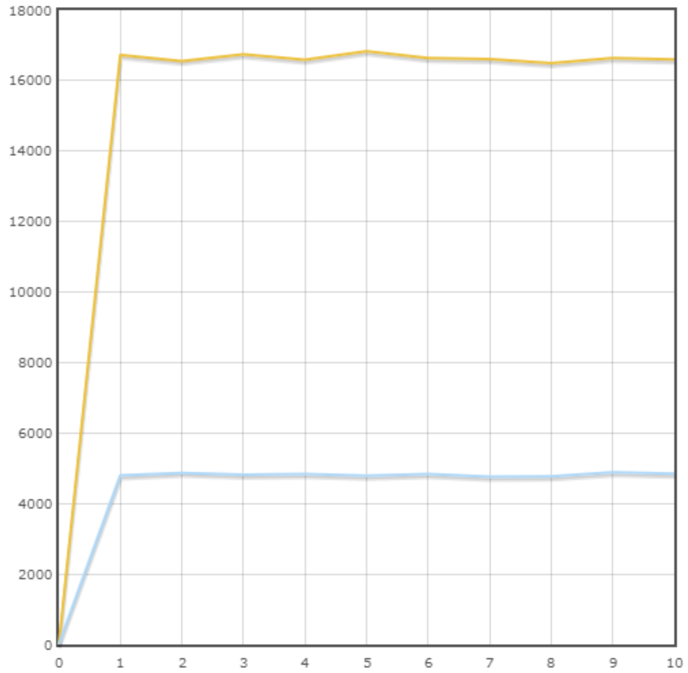

Рис. 5. Транзакция «Order Status»

Средняя скорость:

1. Pony – 16645.6 тран/мин
2. SQLAlchemy – 4820.8 тран/мин

Транзакция «Delivery»

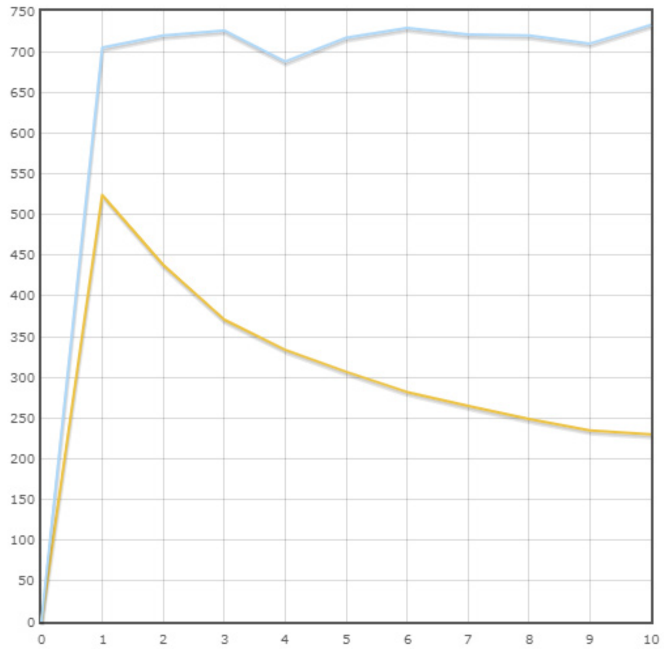

Рис. 6. Транзакция «Delivery»

Средняя скорость:

1. SQLAlchemy – 716.9 тран/мин
2. PonyORM – 323.5 тран/мин

Транзакция «Stock Level»

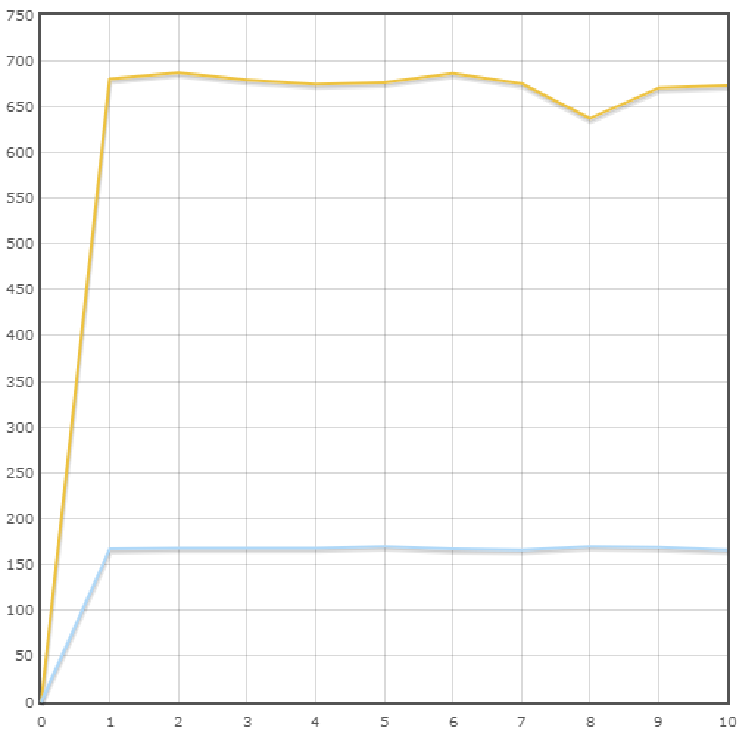

Рис. 7. Транзакция «Stock Level»

Средняя скорость:

1. PonyORM – 677.3 тран/мин
2. SQLAlchemy – 167.9 тран/мин

Анализ результатов тестиров:

Далее были проанализированы результаты тестирования и получены следующие выводы:

1. В 4 из 5 транзакций PonyORM оказалась быстрее. После анализа различий в работе между данными ORM-библиотеками выяснилось, что при генерации SQL кода PonyORM запоминает результат трансляции выражений Python в язык SQL, и не выполняет трансляцию заново при повторном выполнении запроса, в то время как SQLALchemy вынуждена генерировать текст SQL при каждом выполнении запроса. Можно предположить, что именно по этой причине PonyORM оказывается быстрее в большинстве случаев.
2. По результатам теста SQLAlchemy выполняет транзакции типа Delivery быстрее. После анализа генерируемых SQL запросов, выяснилось, что SQLAlchemy умеет объединять несколько операций UPDATE, применяемых к разным объектам, в единую команду. PonyORM в таких случаях генерирует отдельные SQL запросы для каждого отдельно взятой операции UPDATE.Вот пример такого запроса, как он записан в логах SQLAlchemy:

```
INFO:sqlalchemy.engine.base.Engine:UPDATE order_line SET delivery_d= %(delivery_d)s WHERE
order_line.id =  %(order_line_id)s
INFO:sqlalchemy.engine.base.Engine:(
{'delivery_d': datetime.datetime(2020, 4, 6, 14, 33, 6, 922281), 'order_line_id': 316},
{'delivery_d': datetime.datetime(2020, 4, 6, 14, 33, 6, 922272), 'order_line_id': 317},
{'delivery_d': datetime.datetime(2020, 4, 6, 14, 33, 6, 922261))
```

В данном примере видно, что SQLAlchemy генерирует один SQL запрос, в котором производится UPDATE трех строк отношения OrderLine. PonyORM в такой же ситуации сгенерировал бы три запроса.

Вывод

Данное тестирование показало, что Pony работает быстрее в 2 – 4 раза при выборке из базы данных, а SQLAlchemy в некоторых случаях может с заметно большей скоростью производить запросы типа Update.

В дальнейшем таким методом можно проанализировать и другие ORM-библиотеки для Python, что позволит сравнить их друг с другом, увидеть их преимущества и недостатки, и выбирать максимально подходящий ORM, в зависимости от нужд проекта.

Данный тест справился со своей задачей, смог проанализировать работу двух ORM библиотек для Python и выявить, в каких ситуациях лучше использовать ту или иную библиотеку.

Репозиторий теста

URL: https://github.com/DominovTut/Python_ORM_Benchmark

Библиографический список

1. Tortoise ORM Benchmark. URL: https://github.com/tortoise/orm-benchmarks (дата обращения: 20.03.2020).
2. Документация SQLAlchemy. URL: https://docs.sqlalchemy.org/en/13/ (дата обращения: 20.03.2020).
3. Документация PonyORM. URL: https://docs.ponyorm.org/ (дата обращения: 20.03.2020).
4. Спецификация теста TPC-C. URL: http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-c_v5.11.0.pdf (дата обращения: 20.03.2020).
5. Волков А. А. Тесты TPC. URL: https://www.osp.ru/news/articles/1995/0402/13031418 (дата обращения: 20.03.2020).