Метод тестирования производительности и стресс-тестирования центральных сервисов идентификации облачных систем на примере Openstack Keystone[1]

[1]И.В. Богомолов <igor95n@gmail.com>
[1]А.В. Алексиянц <aleksiyantsa@gmail.com>
[1]А. В. Шер <sher-ars@ispras.ru>
[1]О.Д. Борисенко <al@somestuff.ru >
[1,2,3]А.И. Аветисян <arut@ispras.ru>
[1] Институт системного программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, дом 25
[2] Московский государственный университет имени М.В. Ломоносова,
119991, Россия, Москва, Ленинские горы, д. 1.
[3]Московский физико-технический институт (государственный университет),
141700, Россия, Московская область, г. Долгопрудный, Институтский пер., 9

Аннотация. На сегодняшний день платформа OpenStack является одной из лидирующих технологий в области построения облачных сервисов. Keystone - это один из основных сервисом платформыOpenStack. Данный компонент отвечает за аутентификацию и авторизацию пользователей в системе. Keystone является сильнонагруженным сервисом в инфраструктуре OpenStack, поскольку любое взаимодействие между сервисами происходит через Keystone. Таким образом, увеличение числа пользователей облачным сервисом значительно усиливает нагрузку на Keystone. В этой статье мы обращаем внимание на проблему деградации данного сервиса при постоянной нагрузке. Чтобы найти причину такого поведения, мы протестировали сервис Keystone с различными СУБД (MariaDB, PostgreSQL), HTTP-серверами (Apache2, nginx) и физическими устройствами хранения данных (HDD, SSD и tmpfs в оперативной памяти). Использование различных конфигураций для запуска Keystone было необходимо, чтобы сузить поиск причины возникающей деградации. Все тесты запускались через тестирующую систему OpenStack Rally. В качестве результата мы обнаружили, что система начинает довольно быстро деградировать при достаточно слабых нагрузках. Так же было отмечено, что подобная деградация возникает во всех возможных конфигурациях. С целью подтвердить, что подобное поведение Keystone является неудовлетворительным, мы реализовали макетный сервис, который выполняет простейшие функции Keystone. Подобный сервис отказался значительно быстрее самого Keystone. Мы предполагаем, что проблема с Keystone может возникать в результате реализации внутренней логики приложения, либо неэффективного взаимодействия с другими компонентами. Поиск причин является следующим этапом нашего исследования.

Ключевые слова: OpenStack, Keystone, Rally

DOI: 10.15514/ISPRAS-2015-27(5)-4

Для цитирования: Богомолов И.В., Алексиянц А.В., Шер А.В., Борисенко О.Д., Аветисян А.И. Метод тестирования производительности и стресс-тестирования центральных сервисов идентификации облачных систем на примере Openstack Keystone. Труды ИСП РАН, том 27, вып. 5, 2015 г., стр. 49-58. DOI: 10.15514/ISPRAS-2015-27(5)-4.

[1] РФФИ **15-29-07111 офи_м** Исследование методов обеспечения масштабируемости систем в облачных средах и разработка высокопроизводительного отказоустойчивого центрального сервиса идентификации

1. Введение

На текущий момент облачные технологии становятся все более востребованными. Наиболее популярным решением в области открытых облачных сред является платформа OpenStack [1][2][3]. К важнейшим компонентам OpenStack относится сервис Keystone [4], который служит для авторизации и аутентификации пользователей. В его обязанности также входит каталогизация доступных в облаке сервисов. Поскольку любое взаимодействие с облаком, требующее аутентификации и/или авторизации проходит через Keystone, его производительность имеет значительное влияние на качество работы облака в целом.

В ходе эксплуатации OpenStack нами были замечены серьезные проблемы в производительности Keystone. На их существование также указывают, например, следующие источники [5][6].

Таким образом, актуальной видится задача сбора статистики производительности Keystone на различных конфигурациях, которую далее можно будет использовать для поиска узких мест в работе сервиса. Данная работа посвящена достижению этой цели.

В следующем разделе описываются конфигурации тестов и общая методика тестирования. Далее рассматриваются выбранные метрики. В четвертом разделе анализируются результаты.

2. Методика тестирования

С точки зрения последовательности обработки запросов архитектуру Keystone можно разделить на три уровня: верхний, принимающий запросы (веб-сервер), центральный, реализующий основную логику обработки, и нижний, отвечающий за ввод-вывод (базы данных). Проблемы с производительностью могут происходить на любом из них или между ними. В качестве веб-сервера тестировались Apache2 с mod_wsgi и nginx с uwsgi. В качестве слоя баз

данных использовались MariaDB и PostgreSQL. Отметим, что при этом настройки Keystone и используемых компонентов оставались по умолчанию и не оптимизировались под каждую тестовую конфигурацию. Это решение обосновывается тем, что цель данной работы заключается не в оптимизации Keystone под конкретные конфигурации, а в выявлении общих проблем производительности Keystone. Разнообразие выбранных компонентов позволит нам в случае деградации скорости работы на всех выбранных конфигурациях утверждать, что источник проблемы находится не в этих компонентах, а в самом Keystone.

Место для хранения базы данных выбиралось из тех же соображений – тестировались разные конфигурации, чтобы с большей точностью определить узкие места Keystone. При этом были использованы обычные жесткие диски (HDD), твердотельные накопители (SSD) и, наконец, файловая система в оперативной памяти tmpfs.

Таким образом, получается набор конфигураций, показанный на табл. 1.

Табл. 1 Набор тестовых конфигураций

№	Хранение БД	Веб-сервер	БД
1	HDD	mod_wsgi + Apache2	MariaDB
2	HDD	mod_wsgi + Apache2	PostgreSQL
3	HDD	uwsgi + nginx	MariaDB
4	HDD	uwsgi + nginx	PostgreSQL
5	SSD	mod_wsgi + Apache2	MariaDB
6	SSD	mod_wsgi + Apache2	PostgreSQL
7	SSD	uwsgi + nginx	MariaDB
8	SSD	uwsgi + nginx	PostgreSQL
9	tmpfs	mod_wsgi + Apache2	MariaDB
10	tmpfs	mod_wsgi + Apache2	PostgreSQL
11	tmpfs	uwsgi + nginx	MariaDB
12	tmpfs	uwsgi + nginx	PostgreSQL

Для проведения тестов был использован OpenStack Rally [7]. Rally создан специально для нагрузочного тестирования OpenStack облаков. Весь процесс тестирования автоматизирован с помощью инструмента оркестрации Ansible[8].

2.1 Конфигурация тестового стенда

Keystone был развернут на одном вычислительном узле с конфигурацией, указанной на табл. 2.

Табл. 2. Описание тестового стенда

Компонент	**Описание**
Процессор	Intel Core i7-4790 CPU 3.60GHz
Оперативная память	16 GB DDR3, 1600MHz

HDD	Seagate ST2000DM001-1ER164 7200 rpm
SSD	3 диска Kingston V300 450MB/s, все диски объединены в RAID 0 под управлением аппаратного контроллера
Операционная система	Ubuntu Server 14.04.3 LTS

3. Обзор метрики

Мы считаем, что система деградирует при определенной нагрузке, если растет время отклика на запросы, или если происходят ошибки транспортного или HTTP уровня. Таким образом, основной метрикой в нашем тестировании является наличие или отсутствие деградации производительности системы, что свидетельствует о её работоспособности. Работоспособность системы зависит от нагрузки на систему и компонент, с которыми она взаимодействует. Набор используемых компонент определяются описанными выше конфигурациями, а нагрузка числом запросов в секунду к Keystone (Requests per second, далее RPS).

Rally может работать как консольное приложение, так и в качестве демона. Шаблоны тестов называются сценариями и задаются в виде json или yaml файла. Сценарии группируются по темам, каждая тема в коде Rally соответствует классу в Python (например, KeystoneBasic), унаследованному от base.Scenario. Методы этого класса и являются сценариями, а его аргументы - аргументами сценария; таким образом, разработка собственных сценариев достаточно проста.

Кроме того, у описания всех сценариев есть еще три раздела, задающие общее для всех тестов поведение: runner, context и sla. Раздел runner определяет, какого типа будет нагрузка и ее интенсивность. С помощью раздела context указывается контекст нагрузки, например, количество пользователей и тенантов, создающих ее. В опциональном разделе sla (service-level agreement) можно задать условия, при которых тестирование заканчивается. Для тестирования Keystone мы выбрали самый простой существующий сценарий — выдача токена.

В первую очередь нас интересуют результаты работы Keystone при горячем старте. Для этого каждый очередной тест будет запускаться таким образом, чтобы система работала не менее 6 минут. В качестве результатов тестирования рассматриваем поведение системы в течение 5 минут после завершения разогревочного этапа, равного одной минуте.

Будем считать, что N – максимальное значение RPS, при котором не наблюдается деградации в течение 5 минут. Считалось, что система деградирует, если за время выполнения теста был хотя бы один ответ с ошибкой Keystone, или если время отклика начало возрастать. Способ обнаружения спада производительности будет описан ниже.

Для нахождения значения N для каждой комбинации «конфигурация+сценарий» необходимо решить три задачи:

Первая — это непосредственный поиск N. Нахождение N выполняется бинарным поиском, поскольку в выбранной конфигурации факт наличия или отсутствия деградации монотонно зависит от RPS. В качестве верхней границы было выбрано значение 200 RPS, при котором система гарантированно имела деградацию во всех наших конфигурациях.

Следующей задачей является построение сценария для очередного запуска теста. Как было сказано выше, каждый тест должен выполняться не менее 6 минут. В таком случае для каждого теста будет вычисляться значение числа итераций, которое бы нагрузило систему в течение не менее 6 минут при выбранном значении RPS. Для этого необходимо выбранную частоту умножить на 360 секунд. Например, для 200 RPS это равно 360 сек. * 200 RPS = 72000 запросов.

Последней проблемой поиска N является обнаружение факта деградации системы или её отсутствия. Для этого строится модель линейной регрессии по выполненным запросам вида y = ax + b. Будем считать, что деградация отсутствует, если a близок к 0 (для нашего тестирования мы выбрали $a < 0.01$). В случае возникновения хотя бы одной ошибки при ответе считалось, что система деградирует.

После получения N проводятся эксперименты для N-1 и для N+1, чтобы на них визуально убедиться, что полученное N верно. После этого выполняются тесты при N+1 и собираются данные о выполнении тестов.

Результатом тестирования будут являться выходные файлы Rally, которые могут быть двух видов: JSON и HTML. JSON удобно использовать для обработки, а HTML, помимо результатов, содержит код на javascript, предназначенный для их наглядного отображения.

Написанное нами тестирующее приложение для каждого выполненного запроса проверяет отсутствие ошибок в разделе errors и считывает значения duration и timestamps для их последующей обработки.

4. Анализ данных

На графиках представлены результаты Keystone для двух конфигураций: SSD, MariaDB, uWSGI и HDD, PostgreSQL, Apache2 для N и N+1 запросов в секунду.

Для первой представленной конфигурации N=37, для второй N=99. Как видно из графиков, при переходе через определенное значение RPS система показывает постепенную деградацию производительности. Даже изменение RPS на единицу сильно меняет среднее время ответа на запрос в рассматриваемом нами диапазоне. Аналогичная картина наблюдается во всех остальных конфигурациях.

Для сравнения мы реализовали простой сервис, написанный на языке Python с использованием микрофреймворка для веб-приложений Flask[9], который

принимает запрос на выдачу токена, генерирует токен и запоминает информацию о выданных токенах. Этот простой сервис отработал значительно лучше Keystone. Для такого микросервера N оказалось равным 711. Так как все сгенерированные токены за время работы хранились в памяти программы, то можно сравнивать эти результаты с любой конфигурацией Keystone с использованием tmpfs в качестве устройства хранения.

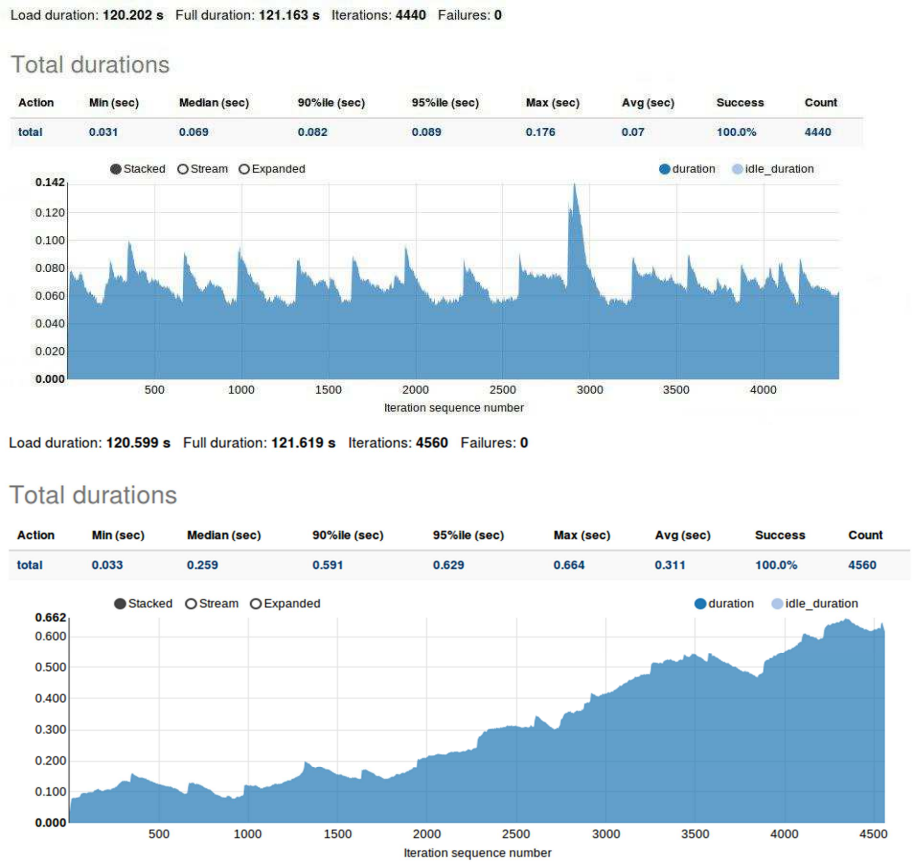

Рис. 1 tmpfs, MariaDB, uWSGI. На оси ординат обозначено время отклика в секундах.

Для нашего тестового сервиса на первом графике N=711, а на втором N=713. Результаты такого сервиса оказались значительно лучше результатов работы Keystone в аналогичной конфигурации. Также стоит отметить, что все тестируемые сценарии проводились с параметром "resource_management_workers": 30, который отвечает за число рабочих

процессов, используемых Rally. При тестировании Keystone для Rally всегда хватало ресурсов на всех экспериментах, поэтому этот параметр не был критичным. В то время как при тестировании нашего сервера Rally не хватало производительности процессора при высоких значениях RPS, что ухудшило результаты, поскольку Rally и тестируемое приложение располагались на одной машине.

Рис. 2 HDD, PostgreSQL, Apache2. На оси ординат обозначено время отклика в секундах.

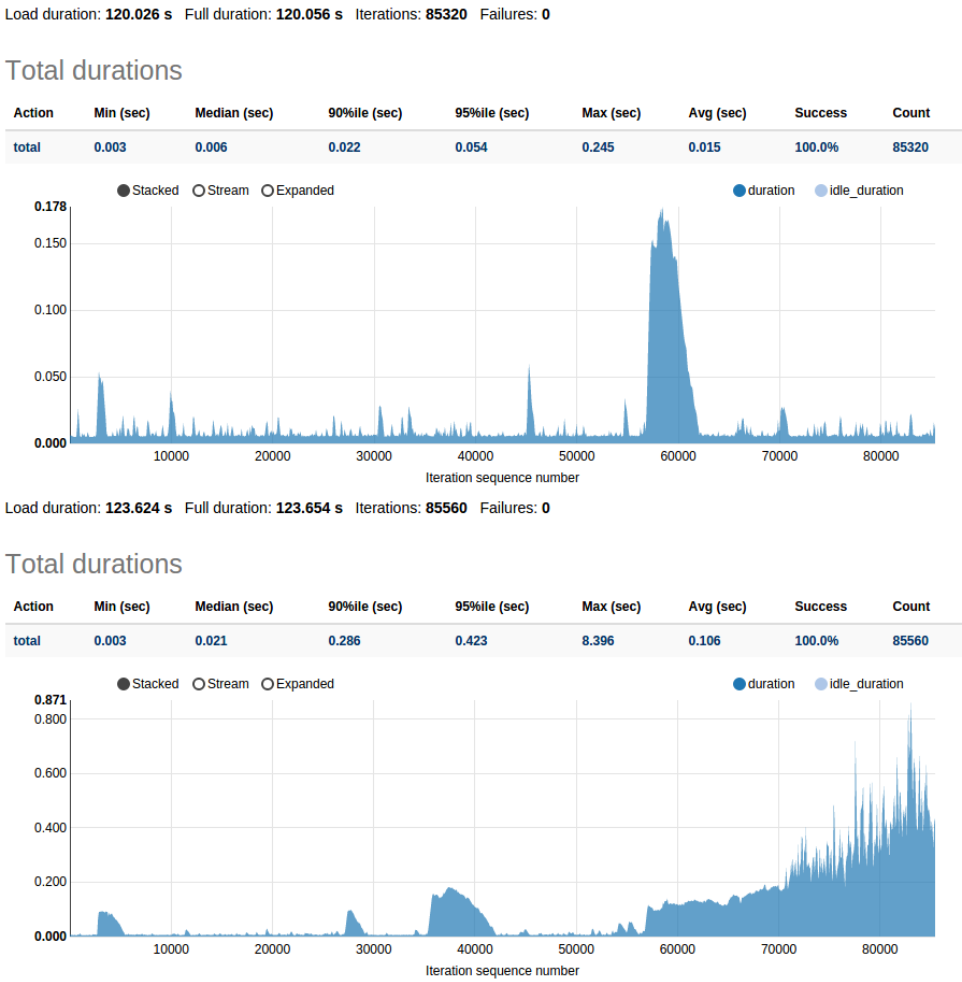

Рис. 3 Результаты тестирования нашего сервиса при N=711 и N=713. На оси ординат обозначено время отклика в секундах.

5. Вывод

Абсолютно на всех перечисленных конфигурациях найденное значение N для Keystone оказывалось значительно меньше, чем на реализованном нами простом сервере. Такое низкое значение N может быть следствием проблем в бизнес-логике Keystone или же неправильной работы с другими компонентами. Выявление причины такого поведения системы является темой для дальнейших исследований. В будущем мы также намерены улучшить модель тестирования таким образом, чтобы производительность Rally не

влияла на работу тестируемого сервера. Для этого планируется подобрать более справедливую конфигурацию для Rally, а также разнести Rally и сервер на разные физические машины.

Список литературы

[1]. Официальная страница проекта OpenStack— https://www.openstack.org/
[2]. M. Bist, M. Wariya, A. Agarwal. Comparing Delta, Open Stack and Xen Cloud Platforms: A Survey on Open Source IaaS. 3rd IEEE International Advance Computing Conference (IACC) , 2013
[3]. T. Rosado, J. Bernardino. An overview of openstack architecture. Proceedings of the 18th International Database Engineering & Applications Symposium. ACM, 2014.
[4]. Keystone project web page — http://docs.openstack.org/developer/keystone/
[5]. Alexander Maretskiy. Finding a Keystone bug while benchmarking 20 node HA cloud performance at creating 400 VMs, December 15 2015. (https://rally.readthedocs.org/en/0.1.1/stories/nova/boot_server.html)
[6]. Neependra Khare. 4x performance increase in Keystone inside Apache using the token creation benchmark, December 15 2015. (https://rally.readthedocs.org/en/0.1.1/stories/keystone/authenticate.html)
[7]. Официальная страница проекта Rally — https://wiki.openstack.org/wiki/Rally
[8]. Официальная страница проекта Ansible — http://www.ansible.com/
[9]. Официальная страница проекта Flask — http://flask.pocoo.org/

A Performance Testing and Stress Testing of Cloud Platform Central Identity: Openstack Keystone Case Study[2]

[1]I. V. Bogomolov <igor95n@gmail.com>
[1]A. Aleksiyants <aleksiyantsa@gmail.com>
[1]A. Sher <sher-ars@ispras.ru>
[1]O. Borisenko <al@somestuff.ru>
[1,2,3]A. Avetisyan <arut@ispras.ru>
[1]Institute for System Programming of the RAS,
25, Alexander Solzhenitsyn Str., Moscow, 109004, Russia.
[2] Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia.
[3] Moscow Institute of Physics and Technology (State University)
9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia

[2] RFBR 15-29-07111 ofi_m

Abstract. Nowadays OpenStack platform is a leading solution in cloud computing field. Keystone, the OpenStack Identity Service, is one of its major components. This service is responsible for authentication and authorization of users and services of the system. Keystone is a high-load service since all interactions between services happen through it. This leads us to the following problem: increasing the number of cloud service users results to significant load growth. In this paper we demonstrate the problem of Keystone performance degradation during constant load. In order to find source of the problem we have tested Keystone with different backends (PostgreSQL, MariaDB), frontends (Apache2, ngnix) and keeping the database on different hardware (HDD, SSD and tmpfs on RAM). Using different configuration sets is necessary because it helps us to narrow the amount of possible reasons causing degradation. Tests were conducted with Rally. As a result, in all test cases we have seen inadequate quick degradation under relatively light load. We have also implemented a mock service which represents the simplest Keystone tasks. Our service turned out to be much faster than Keystone. The problem with Keystone might be related to either its internal logic implementation or incorrect interaction with other components; it is the subject of further research.

Keywords: OpenStack, Keystone, Rally

DOI: 10.15514/ISPRAS-2015-27(5)-4

For citation: Bogomolov I.V., Aleksiyants A., Sher A., Borisenko O., Avetisyan A. A Performance Testing and Stress Testing of Cloud Platform Central Identity: Openstack Keystone Case Study. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 5, 2015, pp. 49-58 (in Russian). DOI: 10.15514/ISPRAS-2015-27(5)-4.

References

[1]. OpenStack project web page — https://www.openstack.org/
[2]. M. Bist, M. Wariya, A. Agarwal. Comparing Delta, Open Stack and Xen Cloud Platforms: A Survey on Open Source IaaS. 3rd IEEE International Advance Computing Conference (IACC) , 2013
[3]. T. Rosado, J. Bernardino. An overview of openstack architecture. Proceedings of the 18th International Database Engineering & Applications Symposium. ACM, 2014.
[4]. Keystone project web page — http://docs.openstack.org/developer/keystone/
[5]. Alexander Maretskiy. Finding a Keystone bug while benchmarking 20 node HA cloud performance at creating 400 VMs, December 15 2015. (https://rally.readthedocs.org/en/0.1.1/stories/nova/boot_server.html)
[6]. Neependra Khare. 4x performance increase in Keystone inside Apache using the token creation benchmark, December 15 2015. (https://rally.readthedocs.org/en/0.1.1/stories/keystone/authenticate.html)
[7]. Rally project web page — https://wiki.openstack.org/wiki/Rally
[8]. Ansible project web page — http://www.ansible.com/
[9]. Flask project web page — http://flask.pocoo.org/