

Проблемы масштабируемости Openstack Keystone

и методы их решения

Борисенко О.Д.,

Облачные среды

Облачная среда — инфраструктура, предназначенная для предоставления вычислительных мощностей по запросу в виде услуги.

Основные модели предоставления услуг:

- Инфраструктура как услуга (IaaS)
- Платформа как услуга (PaaS)
- Программное обеспечение как услуга (SaaS)
- Бессерверные вычисления (serverless computing)

Некоторые возможности облачных сред

- Выделение виртуальных машин с заданными характеристиками по запросу
- Построение виртуальных сетевых инфраструктур
- Виртуализация блочного хранения
- Предоставление объектного хранения с поддержкой метаданных
- Встроенные системы оркестрации ресурсов
- Предоставление заранее настроенного ПО в качестве услуги по запросу пользователя

Разумеется, возможностей гораздо больше.

Все операции с облаком предоставляются, как правило, при помощи API, основанных на HTTP-протоколе (обычно REST API)

Amazon

Компания Amazon предоставляет самый полный набор решений на основе облачных технологий. Остальные облачные среды пытаются догнать лидера.

Compute

-  **EC2**
Virtual Servers in the Cloud
-  **EC2 Container Service**
Run and Manage Docker Containers
-  **Elastic Beanstalk**
Run and Manage Web Apps
-  **Lambda**
Run Code in Response to Events

Storage & Content Delivery

-  **S3**
Scalable Storage in the Cloud
-  **CloudFront**
Global Content Delivery Network
-  **Elastic File System** PREVIEW
Fully Managed File System for EC2
-  **Glacier**
Archive Storage in the Cloud
-  **Import/Export Snowball**
Large Scale Data Transport
-  **Storage Gateway**
Hybrid Storage Integration

Database

-  **RDS**
Managed Relational Database Service
-  **DynamoDB**
Managed NoSQL Database
-  **ElastiCache**
In-Memory Cache
-  **Redshift**
Fast, Simple, Cost-Effective Data Warehousing
-  **DMS**
Managed Database Migration Service

Networking

-  **VPC**
Isolated Cloud Resources
-  **Direct Connect**
Dedicated Network Connection to AWS
-  **Route 53**
Scalable DNS and Domain Name Registration

Developer Tools

-  **CodeCommit**
Store Code in Private Git Repositories
-  **CodeDeploy**
Automate Code Deployments
-  **CodePipeline**
Release Software using Continuous Delivery

Management Tools

-  **CloudWatch**
Monitor Resources and Applications
-  **CloudFormation**
Create and Manage Resources with Templates
-  **CloudTrail**
Track User Activity and API Usage
-  **Config**
Track Resource Inventory and Changes
-  **OpsWorks**
Automate Operations with Chef
-  **Service Catalog**
Create and Use Standardized Products
-  **Trusted Advisor**
Optimize Performance and Security

Security & Identity

-  **Identity & Access Management**
Manage User Access and Encryption Keys
-  **Directory Service**
Host and Manage Active Directory
-  **Inspector** PREVIEW
Analyze Application Security
-  **WAF**
Filter Malicious Web Traffic
-  **Certificate Manager**
Provision, Manage, and Deploy SSL/TLS Certificates

Analytics

-  **EMR**
Managed Hadoop Framework
-  **Data Pipeline**
Orchestration for Data-Driven Workflows
-  **Elasticsearch Service**
Run and Scale Elasticsearch Clusters
-  **Kinesis**
Work with Real-Time Streaming Data
-  **Machine Learning**
Build Smart Applications Quickly and Easily

Internet of Things

-  **AWS IoT**
Connect Devices to the Cloud

Game Development

-  **GameLift**
Deploy and Scale Session-based Multiplayer Games

Mobile Services

-  **Mobile Hub**
Build, Test, and Monitor Mobile Apps
-  **Cognito**
User Identity and App Data Synchronization
-  **Device Farm**
Test Android, FireOS, and iOS Apps on Real Devices in the Cloud
-  **Mobile Analytics**
Collect, View and Export App Analytics
-  **SNS**
Push Notification Service

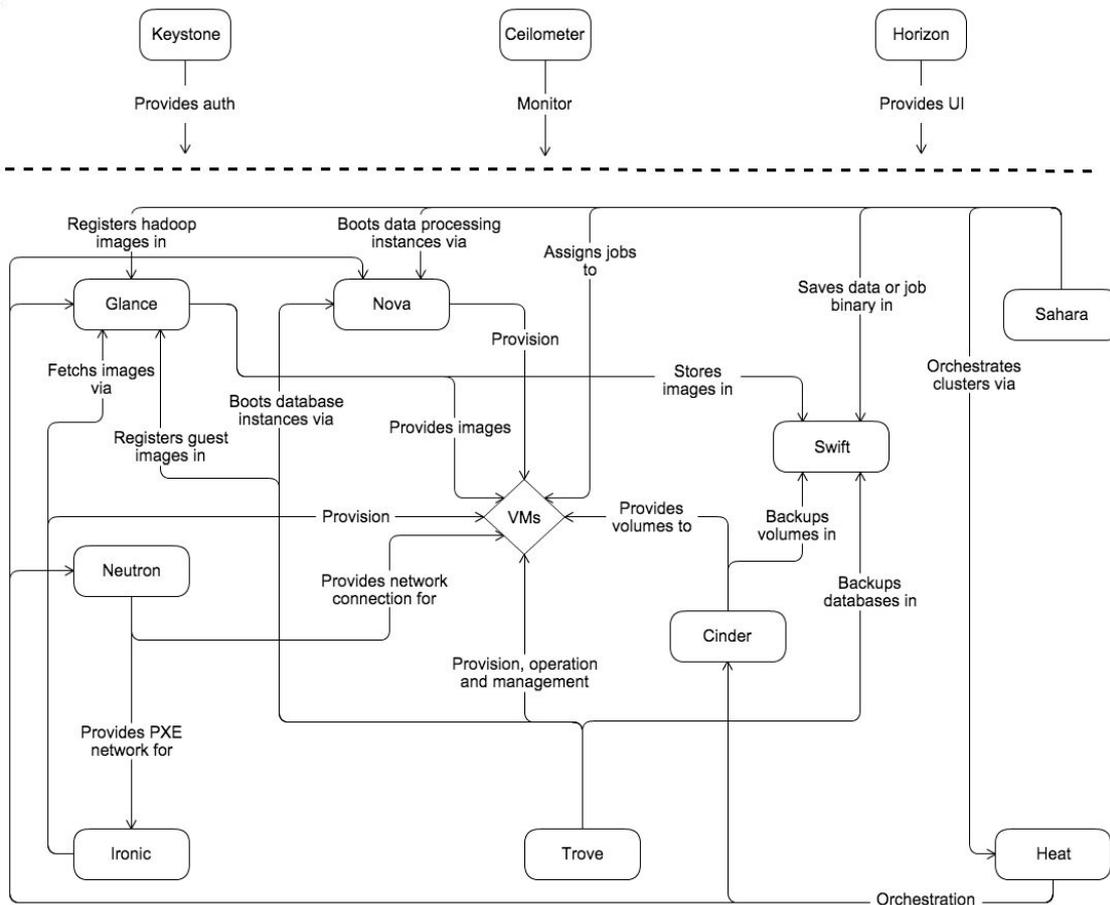
Application Services

-  **API Gateway**
Build, Deploy and Manage APIs
-  **AppStream**
Low Latency Application Streaming
-  **CloudSearch**
Managed Search Service
-  **Elastic Transcoder**
Easy-to-Use Scalable Media Transcoding
-  **SES**
Email Sending and Receiving Service
-  **SQS**
Message Queue Service
-  **SWF**
Workflow Service for Coordinating Application Components

Enterprise Applications

-  **WorkSpaces**
Desktops in the Cloud
-  **WorkDocs**
Secure Enterprise Storage and Sharing Service
-  **WorkMail**
Secure Email and Calendaring Service

Базовые компоненты Openstack



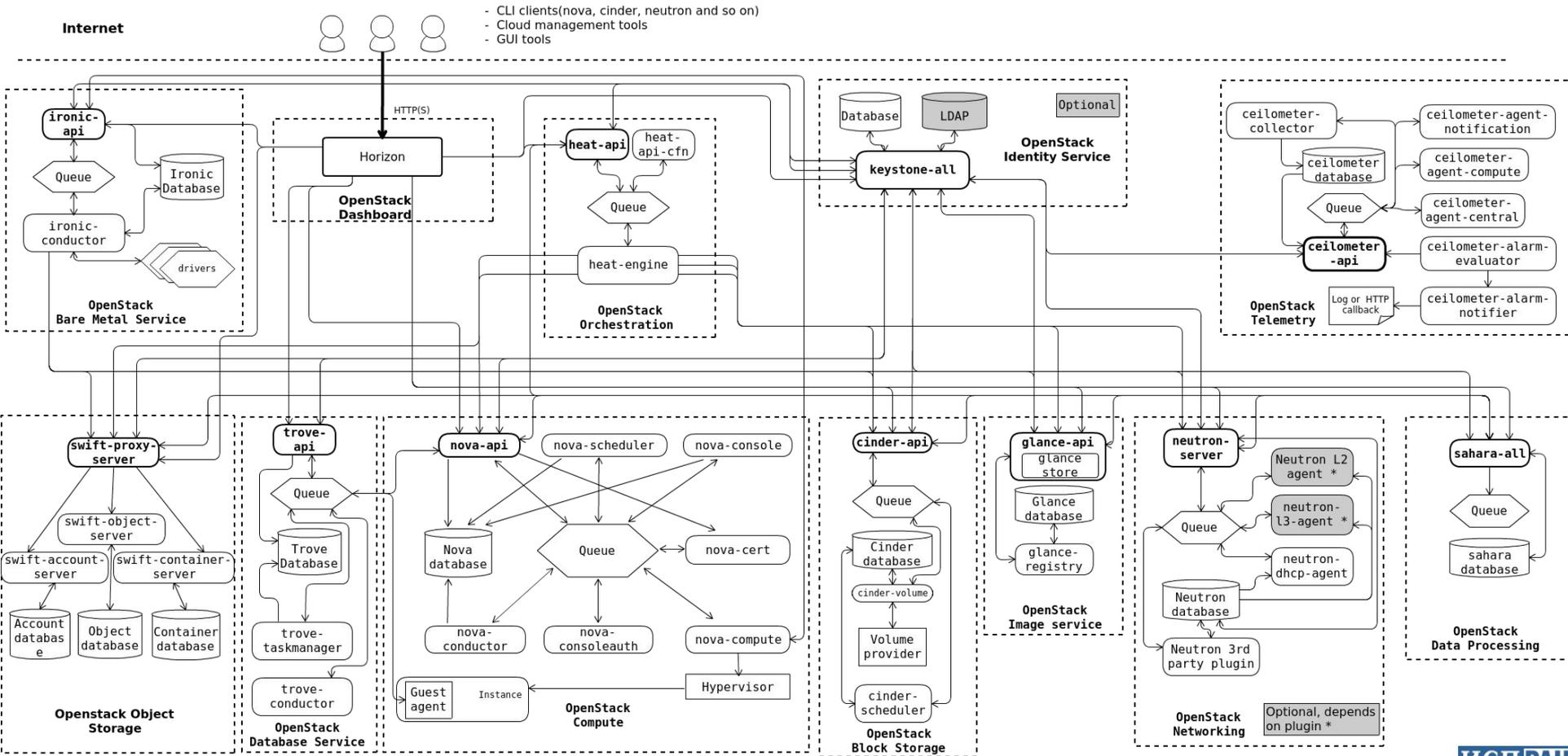
Openstack состоит из т.н. “проектов”.

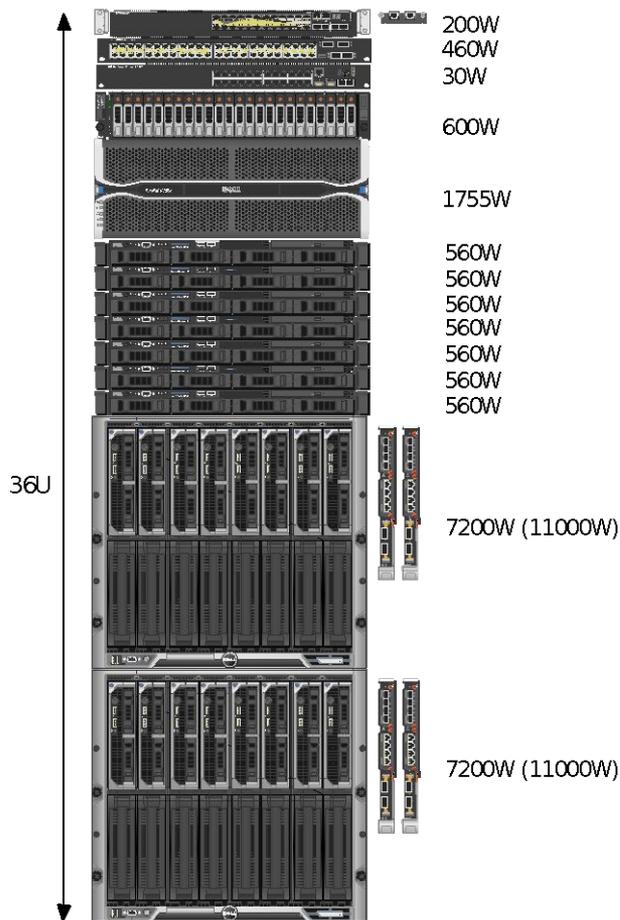
Основные из них:

- Keystone
- Nova
- Horizon
- Neutron
- Cinder
- Glance
- Swift
- Heat
- Sahara
- Ceilometer

Минимальный необходимый набор —
Keystone, Glance и Nova.

Openstack, полная архитектура





Характеристики (без контроллеров):

- Openstack Liberty из исходных кодов
- 3968GB RAM без учета overcommit
- 256 ядер (512 с учетом HT)
- 28.5ТВ локальных для виртуальных машин дисков
- >130ТВ нелокального хранения с пропускной способностью 10Гбит/с
- Полная связность 20Гбит/с
- Канал на вход/выход 1Гбит/с, но будет 10Гбит/с к декабрю
- 240 IPv4 внешних адресов
- 1000 внутренних адресов в сети ИСП

Масштабируемость - обман!

Openstack страдает от проблем масштабируемости, как по количеству физических узлов системы, так и по количеству активных пользователей.

Это касается именно открытой части кодовой базы: коммерческие компании, утверждающие, что предоставляют доступ к облачным услугам на основе Openstack, в реальности используют собственные закрытые решения с тем же набором API

Уязвимые сервисы в Openstack

Мы достоверно знаем о трех:

- Keystone — сервис центральной идентификации и аутентификации пользователей и сервисов
- Nova-conductor — внутренний сервис обмена данными в сервисе виртуализации (Nova)
- Oslo.messaging — набор библиотек, обеспечивающих гарантированную доставку сообщений в системе

Проблема сервисов аутентификации и авторизации

На момент начала 2016 года не существовало ни одного известного облака более чем на **200** физических узлов* на открытой кодовой базе Openstack

*по информации разработчиков основной ветки Openstack

Все подобные сервисы построены по общему принципу:

- Слой внутренней маршрутизации запросов
 - Отвечает за работу сервиса через протокол HTTP.
- Слой промежуточного ПО (middleware)
 - На этом слое происходит вся логическая обработка запросов.
- Слой персистентного хранения
 - Отвечает за связь с СУБД и передачу запросов к базе данных.

- Проверка учетных данных пользователей системы
- Осуществление мультиарендности для разных групп пользователей системы
- Проверка прав пользователей
- Проверка принадлежности других сервисов облачной среды к данному облаку (защита от подмены)
- Выдача токенов авторизации
- Каталог сервисов системы

Openstack Keystone реализован на языке Python в соответствии со стандартом WSGI и с использованием ORM SQLAlchemy для коммуникаций с СУБД.

Несмотря на универсальность стандарта WSGI и поддержку множества СУБД данной ORM, количество поддерживаемых вариантов развертывания системы ограничено.

Варианты развертывания Keystone

- Маршрутизация запросов:
 - Apache2 + mod_wsgi
 - uWSGI + Nginx
- Промежуточное ПО:
 - CPython
- СУБД
 - MariaDB/MySQL
 - PostgreSQL
- Слой кэширования
 - Dogpile
 - memcached

Выявление деградации

Нами был реализован набор утилит:

- автоматизирующих развертывание Keystone во всех возможных конфигурациях,
- автоматизирующих процесс тестирования (при помощи Openstack Rally)
- собирающих информацию о проведенных тестах и выявляющих замедление системы при возрастании нагрузки

Также тестировался бэкенд системы хранения:

данные, хранящиеся в СУБД, попадали на обычный SATA HDD, на RAID-0 из 3 SSD или в tmpfs в зависимости от сценария

Метод тестирования

Цели тестирования:

1. Нахождение такого числа запросов в секунду (RPS) к Keystone, при котором время задержки (latency) ответа на запрос начинает линейно или быстрее возрастать в течение фиксированного временного окна.
2. Поиск причин роста времени задержки ответов.

Тестирование проводилось как в режиме единственного узла Keystone, так и в High Availability варианте развертывания, рекомендованном разработчиками.

В качестве сценария тестирования выбраны наиболее естественные операции: выдача токена и валидация токена.

Разница между вариантами развертывания

Были протестированы все комбинации вариантов развертывания каждого из слоев системы, включая бэкенд хранения (HDD/SSD/tmpfs).

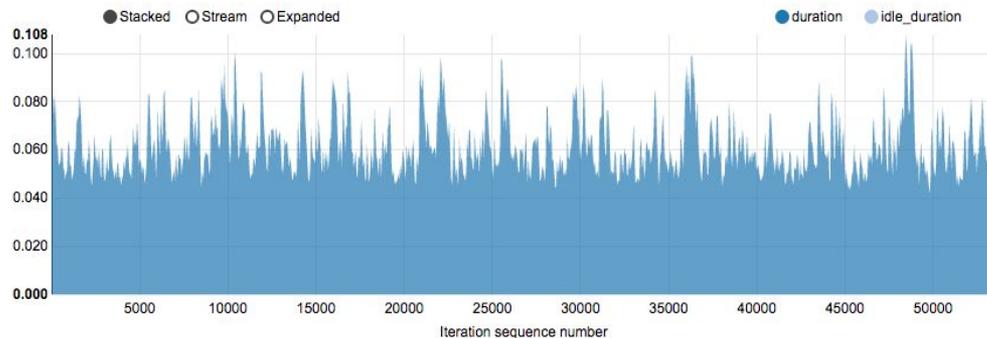
Разница между результатами в разных вариантах не существенна в пределах одного бэкенда хранения.

Тем не менее, приведем некоторые из них.

Примеры нормальной работы системы

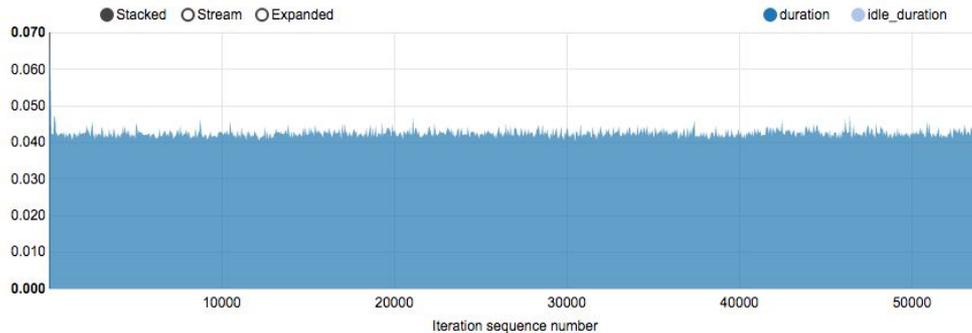
Total durations

Action	Min (sec)	Median (sec)	90%ile (sec)	95%ile (sec)	Max (sec)	Avg (sec)	Success	Count
total	0.031	0.059	0.079	0.086	0.16	0.061	100.0%	53640



Total durations

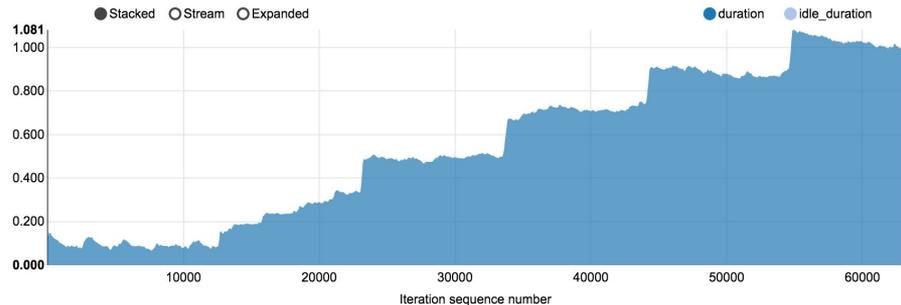
Action	Min (sec)	Median (sec)	90%ile (sec)	95%ile (sec)	Max (sec)	Avg (sec)	Success	Count
total	0.031	0.042	0.045	0.046	0.181	0.043	100.0%	54000



Примеры деградации производительности

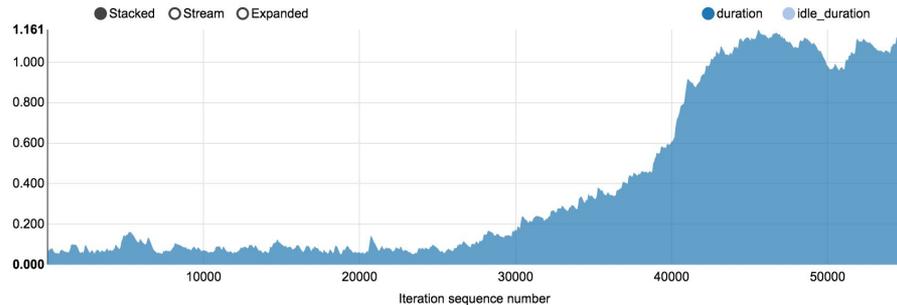
Total durations

Action	Min (sec)	Median (sec)	90%ile (sec)	95%ile (sec)	Max (sec)	Avg (sec)	Success	Count
total	0.036	0.503	1.01	1.032	1.23	0.546	100.0%	63000



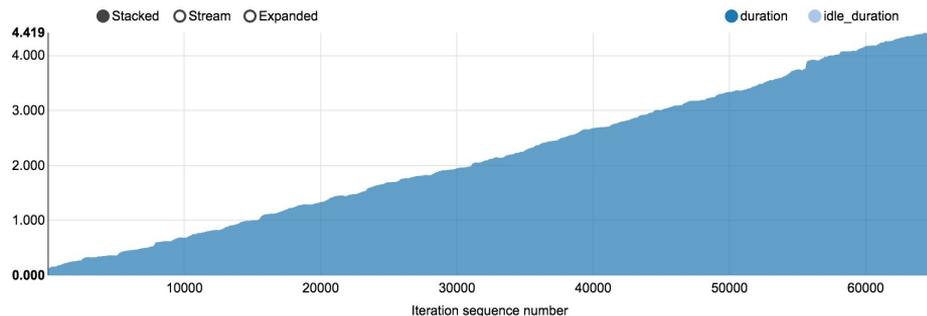
Total durations

Action	Min (sec)	Median (sec)	90%ile (sec)	95%ile (sec)	Max (sec)	Avg (sec)	Success	Count
total	0.033	0.138	1.092	1.115	1.262	0.39	100.0%	54720

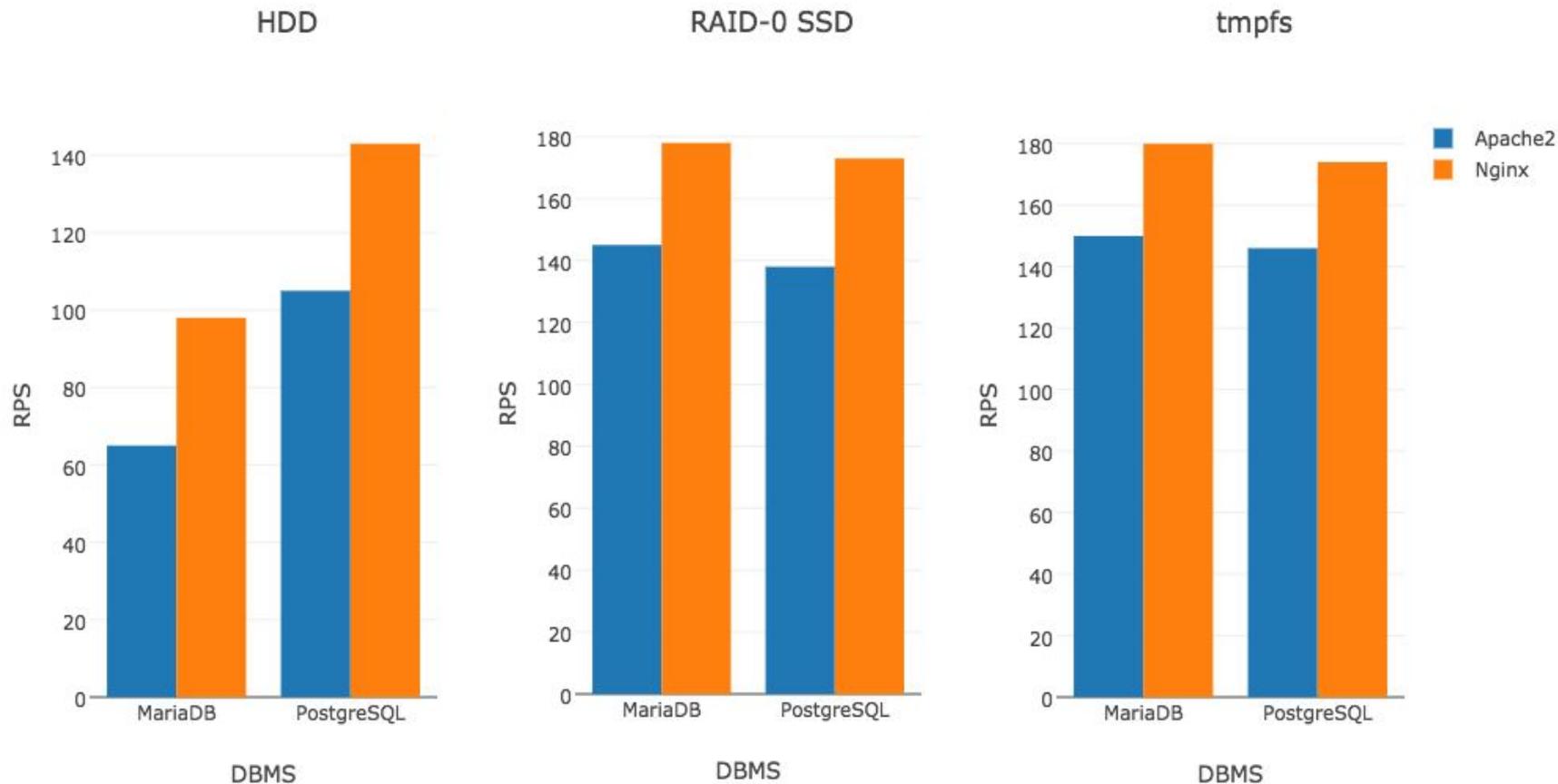


Total durations

Action	Min (sec)	Median (sec)	90%ile (sec)	95%ile (sec)	Max (sec)	Avg (sec)	Success	Count
total	0.046	2.116	4.076	4.262	4.456	2.194	100.0%	64800



Результаты для одного узла Keystone



Граф вызовов и профилирование

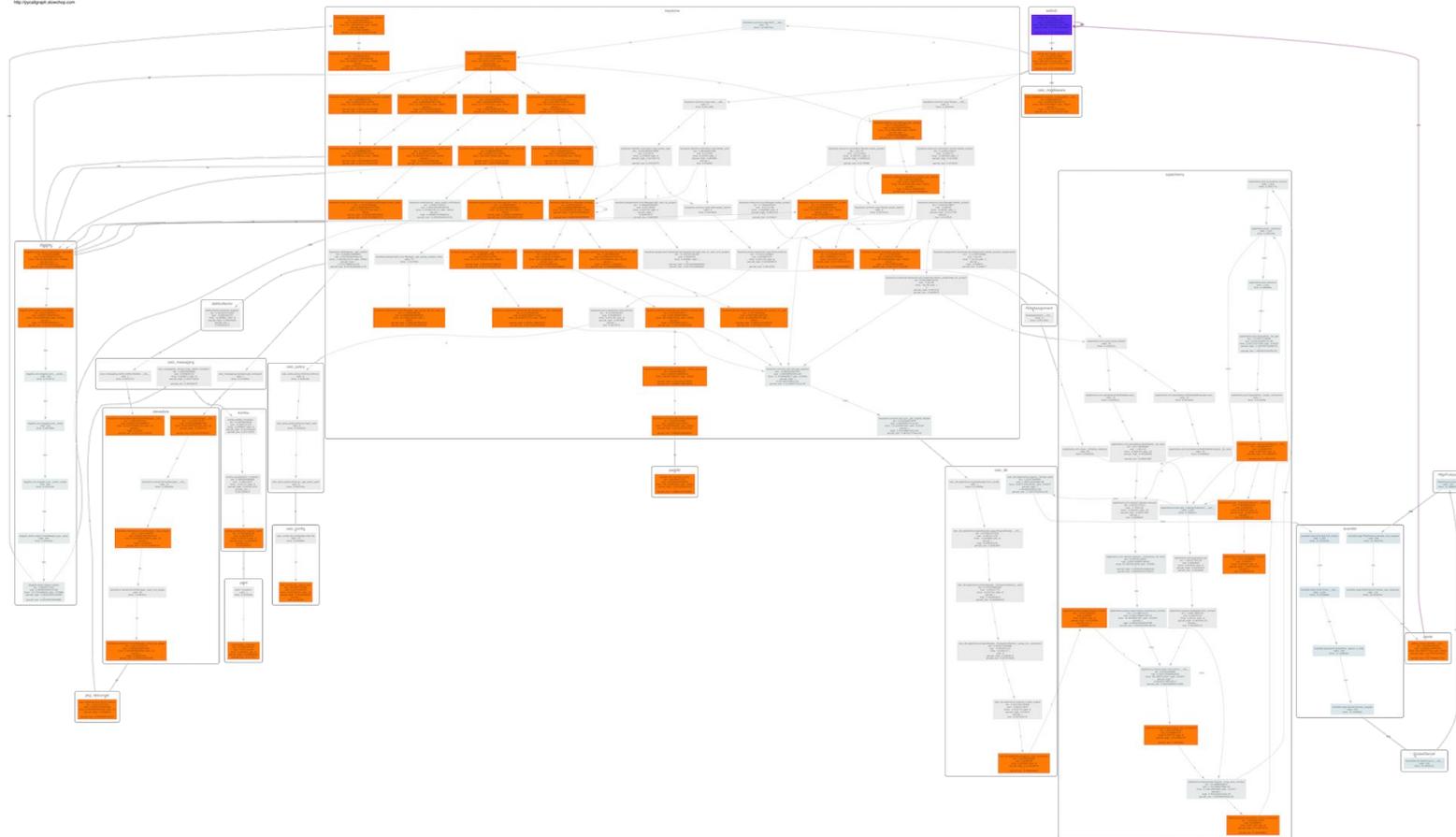
Перед тестированием НА-варианта, мы решили проанализировать поведение программного кода.

Мы реализовали утилиту, которая накладывает результаты профилирования системы при разных RPS на граф вызовов и при помощи специальной эвристики фильтрует узлы этого графа по набору параметров.

Изначальное число узлов в графе вызовов для основного сценария тестирования — 1172

Уменьшенный и отфильтрованный граф вызовов

Generated by Public Call Graph v1.1
http://www.kaspersky.com/learning/



Шифрование!

```
keystone.common.utils.check_password
div: 4.84281005885
sub: 0.00367101419642
time: 146.583596863 calls: 39930
percall_high: 0.0142341832958
percall_low: 0.00865258068087
```

73

passlib

```
passlib.utils.handlers.verify
div: 4.84340477321
sub: 0.00366416458469
time: 146.310091867 calls: 39930
percall_high: 0.014203367418
percall_low: 0.00863247439801
```

После изучения всех “интересных” узлов графа, стало ясно, что в варианте развертывания Keystone на одном узле главная проблема — проверка пароля.

Пароль хэшируется при помощи SHA-512 с использованием 10000 раундов.

Центральная SQL база

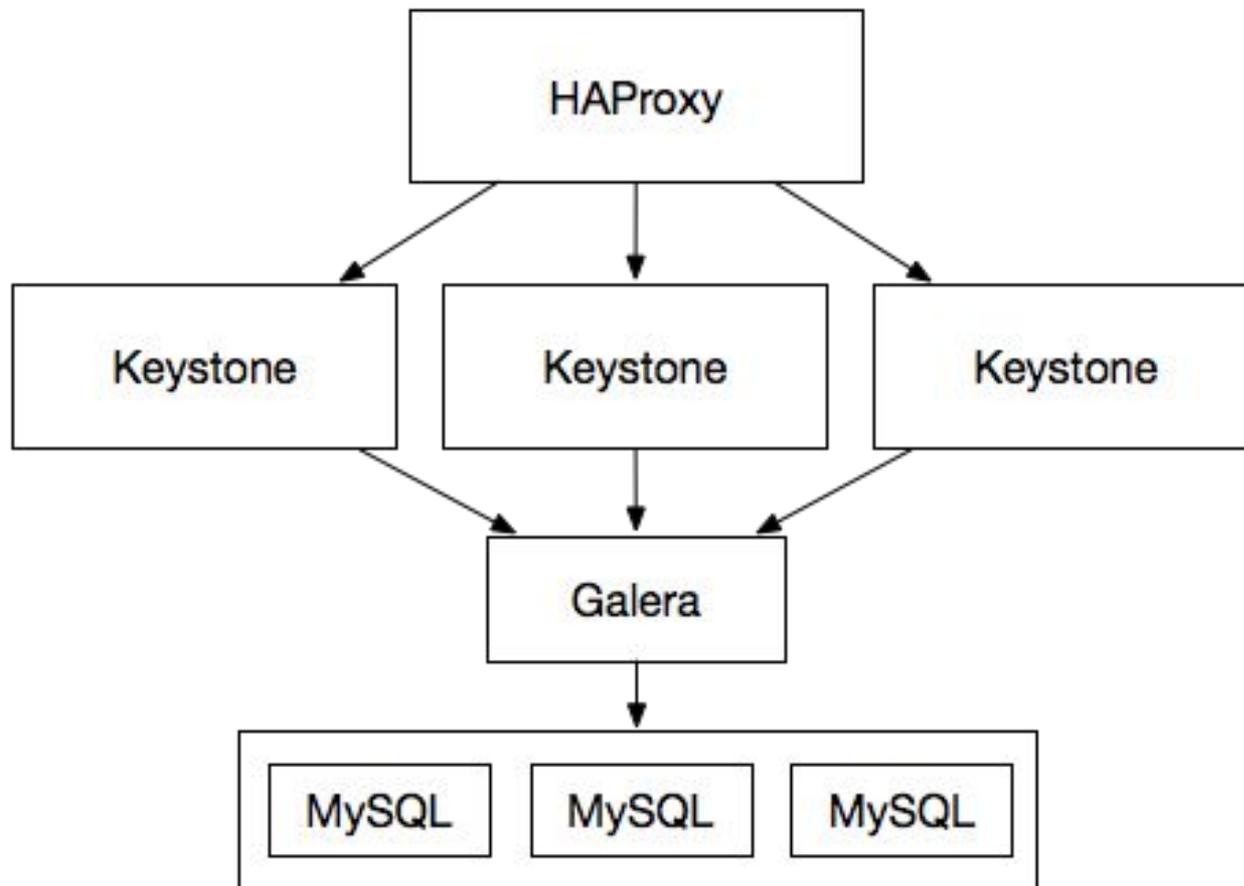
В то же самое время, при проведении любой операции в Keystone, проверяется принадлежность пользователя к мультиарендной группе и сверяется его SHA-512 хэш пароля с тем, который приходит в систему для получения токена.

Фактически любая операция затрагивает реляционную СУБД

Единственный метод, предлагаемый сообществом Openstack для обеспечения HA:

- Использовать строго MySQL/MariaDB
- Сохранность данных обеспечивается за счет нескольких реплик базы данных
- Координация MariaDB при помощи Galera
- Балансировка запросов к Keystone-узлам при помощи HAProxy

Схема масштабирования от разработчиков Openstack



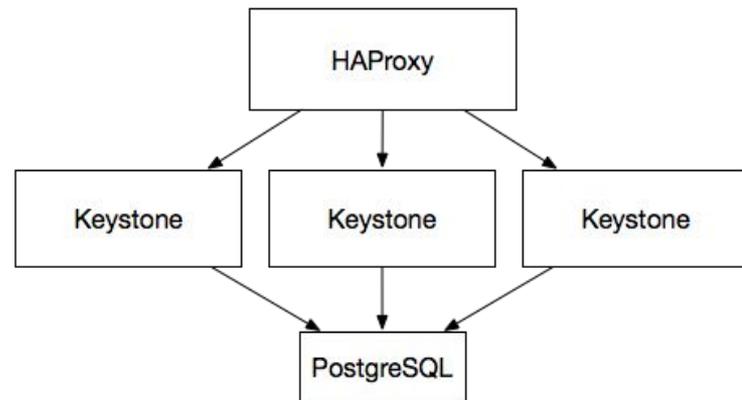
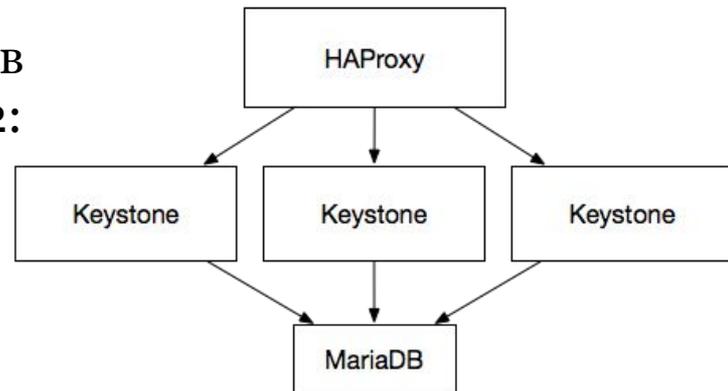
Слабости подхода

- Galera является внешним способом синхронизации СУБД и обеспечивает работу системы с производительностью самого медленного узла + накладные расходы
- Увеличение числа узлов с Keystone увеличивает количество соединений с БД
- Увеличение числа рабочих процессов Keystone увеличивает число соединений с БД

Тестирование на множестве узлов

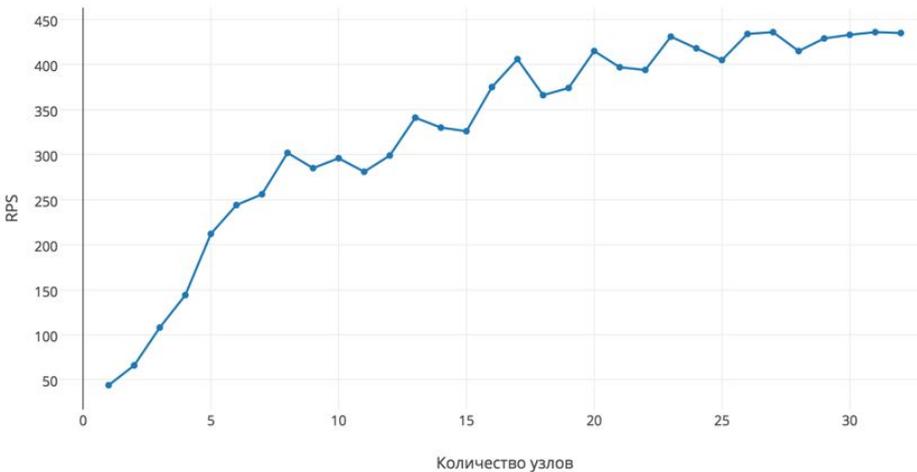
Мы проводили тестирование HA-развертывания в двух вариантах с числом узлов Keystone от 1 до 32:

- Пропускная способность сети 4Гбит/с (из-за инкапсуляции в VXLAN)
- Rally (тестирующий фреймворк):
 - 8 VCPU, 32GB RAM
- HAProxy:
 - 8 VCPU, 32GB RAM
- Keystone узлы:
 - 2 VCPU, 8GB RAM каждый
- СУБД-узел:
 - 8 VCPU, 32GB RAM

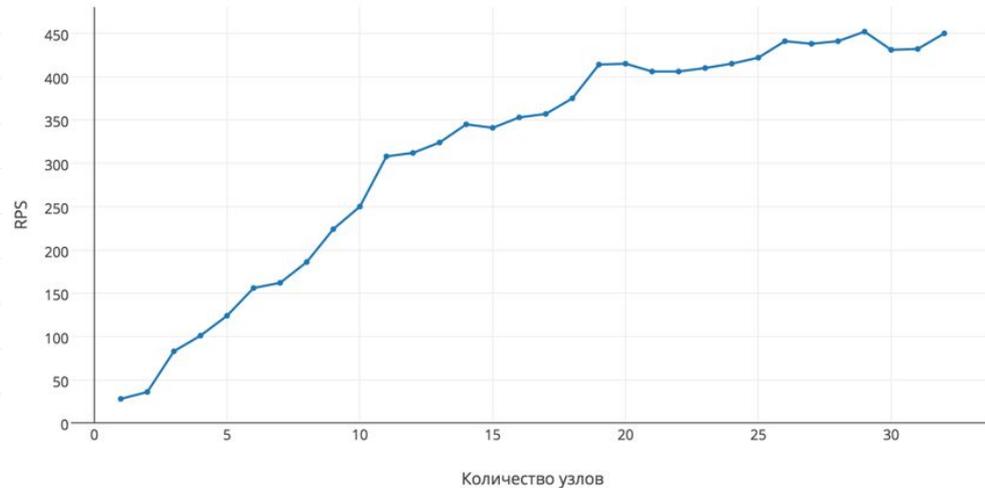


Эксперименты

High Availability Keystone/MariaDB/tmpfs



High Availability Keystone/PostgreSQL/tmpfs



Ограничения по масштабируемости

Найденные значения RPS характеризуют максимальное количество одновременных пользователей системы.

Выявленные ограничения:

- 45RPS на одно вычислительное ядро с частотой 3ГГц
- С ростом числа узлов порождается все больше соединений с СУБД, и это проблема

Так почему это проблема?

1. Каждый запрос аутентификации съедает 1 RPS и при этом обращается к базе данных
2. Каждая операция с любым сервисом облака также порождает запрос к Keystone. В процессе создания виртуальной машины обычно порождается 12 внутренних запросов.
3. Каждый пользователь и каждый сервис Openstack обязан переполучить токен раз в час

И все еще, почему это проблема?

Представим себе эталонную конфигурацию, предлагаемую разработчиками: 3 узла с Keystone (остальное за скобками).

Самый мощный серверный процессор из доступных сейчас — **Intel® Xeon® Processor E7-8890 v4 (24 ядра, 2.6ГГц, до 8 ядер на одной материнской плате)**. Т.е 3 сервера в пределе 576 ядер, которые выдержат ~ 23000 RPS в идеальном случае. Суммарная стоимость только процессоров ~172000\$

И все это ради сервиса аутентификации и авторизации на 23000 RPS?

Безумие

Тем не менее, 23000 RPS ограничивают число пользователей 23000 одновременными пользователями единовременно, при условии, что они ничего особенно не делают и в системе есть только Keystone. При этом мощности делятся с узлами системы.

Новый подход: noSQL distributed/IMDG

- От сложности хэширования избавиться нельзя
- Тем не менее, можно избавиться от узкого места в виде соединений с СУБД
- Для этого необходимо отказаться от реляционных СУБД и перейти к решениям, которые масштабируются линейно от числа узлов и работают в оперативной памяти:
In-Memory Data Grid

In-Memory Data Grid

Мы проводили исследования на эту тему, и выяснили, что существует как минимум два решения, который обеспечивают устойчивость при уровне изоляции Repeatable Read с линейным ростом производительности:

1. Tarantool: 55 тысяч транзакций в секунду на узел
2. Apache Ignite/Gridgain 23 тысячи транзакций в секунду на узел

Сохранность данных гарантируется синхронной репликацией с заданным числом копий объектов между узлами системы.

Более того, транзакционный режим нужен не для всего.

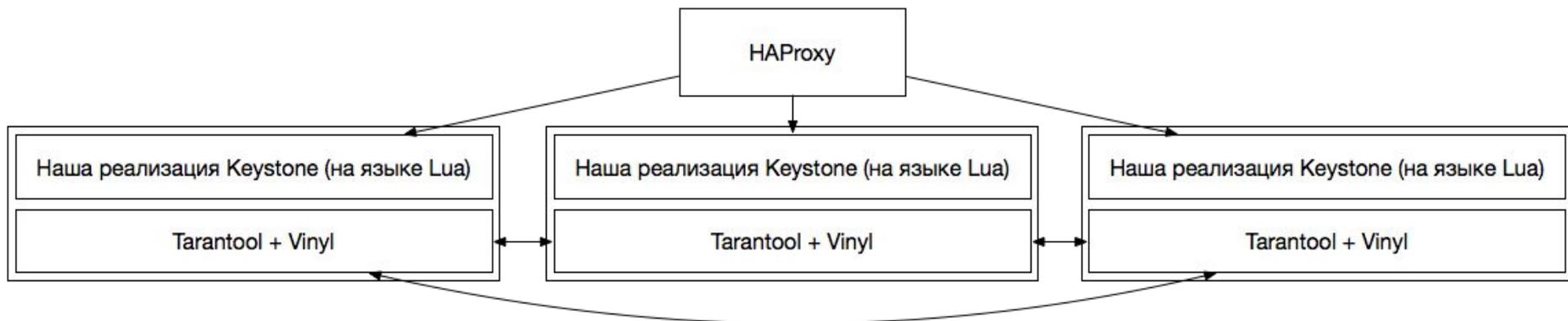
Предлагаемое построение системы

- Каждый узел сервиса центральной идентификации и авторизации обеспечивает работу всех слоев системы сразу
- Масштабируемость достигается за счет близости данных к узлу и репликации этих данных между узлами системы
- Быстродействие достигается работой с данными исключительно в памяти и без ограничений реляционных СУБД

Устройство прототипа

Он еще не готов, но мы надеемся на 30000+ RPS на узел с минимально возможной конфигурацией в три узла (т.е 90000+RPS). Это позволит строить облачные среды для датацентров.

(дальнейший план — kong + tarantool + lua)



- Системы аутентификации и авторизации в облачных системах опираются на методы шифрования/хэширования и реляционные СУБД
- Хэширование — узкое место при выдаче токенов с точки зрения одного узла
- Использование вариантов реляционных СУБД ограничено использованием ORM для всех систем
- Отсутствие полноценного master-master режима в распространенных РСУБД становится проблемой
- Выход: перенос решений на noSQL СУБД с полноценным режимом master-master.

Спасибо за внимание