Московский физико-технический институт

(Государственный университет)

Факультет управления и прикладной математики

Кафедра теоретической и прикладной информатики

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

«Динамическое распределение вычислительной нагрузки в виртуальных средах на примере Jelastic Cloud»

Выполнила:

студентка 4 курса 073 группы

Ехлакова Елена Андреевна

Научный руководитель:

Гребенщиков Владимир Борисович

Москва, 2014

Содержание

1 Введение

Развитие виртуализации серверов, сетей и устройств хранения позволяет сегодня создавать пулы серверных ресурсов, где множество приложений могут совместно использовать любой сервер в пуле. Главные стимулы для перехода на такие технологии — это увеличение гибкости при изменении вычислительных потребностей, возможность быстро перенацелить серверную мощность под текущие требования бизнеса, и снижение общей стоимости владения.

К сожалению, сложность облачной инфраструктуры создает дополнительные проблемы при администрировании. Имеется много нагрузок, на каждый сервер можно назначить некое конечное их число, и каждая нагрузка имеет требования по мощности, которые могут часто меняться в зависимости от потребностей бизнеса. И пока существующие способы администрирования мощности работают только для отдельных видов приложений.

В данной работе предложен процесс управления вычислительной мощностью, позволяющий автоматизировать эффективное использование таких пулов при хостинге большого количества сервисов.

2 Задача балансировки нагрузки

2.1 Постановка задачи

Имеется набор виртуальных сред (серверов), на каждом из которых развернуты приложения. Приложения создают нагрузку на сервера. Разную нагрузку в разные периоды времени. В общем случае, нагрузка характеризуется несколькими параметрами: CPU, RAM, др. В облачной платфоме Jelastic единица измерения ресурсов - клаудлет (cloudlet). Один клаудлет равен примерно 200МГц тактовой частоты процессора или 128МБ оперативной памяти.

По каждому приложению известна создаваемая им нарузка в течение планового периода. Это позволяет определить нагрузку на каждый сервер в каждый момент времени по каждому параметру. Если нагрузка не превосходит заданных порогов, то сервер находится в рабочем режиме. В противном случае сервер работает с перегрузкой. Чтобы избежать перегрузки, приложения можно перемещать с одного сервера на другой.

Задача состоит в том, чтобы достичь минимальной суммарной перегрузки серверов на всем плановом периоде.

2.2 Математическая модель

S - множество серверов

A - множество приложений

T - плановый период

c_{at} - нагрузка приложения a в момент времени t

C_s - пороговая нагрузка для сервера s

$b_{as}^{i/o}$ - накладые расходы по вставке / изъятию приложения a с сервера s

$B_s^{i/o}$ - предельно допустимые накладные расходы на вставку / изъятие приложений с сервера s

x_{as}^0 - начальное распределение приложений по серверам

$$x_{as} = \begin{cases} 1 & \text{если приложение } a \text{ развернуто на сервере } s \\ 0 & \text{в противном случае} \end{cases}$$

Требуется найти перераспределение приложений по серверам, при котором суммарное превышение нагрузки над пороговым было бы минимальным, а накладные расходы на перемещение приложений укладывались в предельно допустимые.

Таким образом, требуется найти минимум целевой функции

$$\sum_{s \in S} \sum_{t \in T} max\{0, \sum_{a \in A} c_{at} x_{as} - C_s\} \tag{1}$$

при ограничениях:

$$\sum_{s \in S} x_{as} = 1, \forall a \in A \tag{2}$$

$$\sum_{a \in A} b_{as}^i max\{0, x_{as} - x_{as}^0\} \leq B_s^i, \forall s \in S \tag{3}$$

$$\sum_{a \in A} b_{as}^o max\{0, x_{as}^0 - x_{as}\} \leq B_s^o, \forall s \in S \tag{4}$$

$$x_{as} \in \{0, 1\}, \forall a \in A, \forall s \in S \tag{5}$$

В ограничении (2) учитывается, что одно приложение развернуто только на одном сервере. Согласно ограничениям (3) и (4) перемещение приложений

с одного сервера на другой требует затрат, которые не могут превышать пороговые значения. Под решением задачи понимается распределение приложений по серверам x_{as} .

Решение является допустимым, если выполняются условия (2) - (5). Необходимо найти допустимое решение с минимальной целевой функцией (1).

Данная задача дискретной оптимизации является NP-трудной. Для ее решения предлагается метод локального поиска, основанный на идее рандомизации окрестностей и списках запретов, и метод поиска с чередующимися окрестностями.

2.3 Поиск с чередующимися окрестностями

2.3.1 Описание алгоритма

Для решения задач дискретной оптимизации методы локального поиска являются наиболее естественными и наглядными. Однако простой локальный спуск не позволяет находить глобальный оптимум задачи [1].

Стандартный алгоритм локального спуска начинает работу с некоторого начального решения x_0,выбранного случайно или с помощью какого-либо вспомогательного алгоритма. На каждом шаге локального спуска происходит переход от текущего решения к соседнему решению с меньним значением целевой функции до тех пор, пока не будет достигнут локальный оптимум.

Алгоритм локального спуска

1. Выбрать начальное решение x_0
2. Цикл $k = 1, ..., k_{max}$
   (a) Найти такое $x' \in N_k(x)$, что $f(x') = max\{f(y)|y \in N_k(x)\}$
   (b) Если $f(x') > f(x_0)$, то $x_0 = x'$, иначе достигнут локальный максимум

$N_k(x)$ - функция окрестности. На каждом шаге локального спуска она задает множество возможных направлений движения. Правило выбора направлений может оказать существенное влияние на временную сложность алгоритма и результат его работы. Таким образом, при разработке алгоритмов важно не только правильно определить окрестность, но и верно задать правило выбора направления спуска. Интуитивно кажется, что в окрестности надо брать элемент с наименьшим значением целевой функции. Однако, иногда несколько шагов с ухудшением впоследствии могут привести к лучшему локальному оптимуму.

При выборе окрестности хочется иметь множество $N_k(x)$ как можно меньшей мощности, чтобы сократить трудоемкость одного шага. С другой стороны, более широкая окрестность может привести к лучшему локальному оптимуму.

Алгоритм VNS

В рамках алгоритма допускаются следующие операции:

1. $Move(x_{as})$ перемещение приложения на другой сервер
2. $Swap(x_{as})$ меняем одно приложение на другое
3. $Assign(x_{as})$ изъять с каждого сервера по приложению и распределить их оптимально между серверами

Окрестность $N_k(x_{as})$ - двигаем k приложений на другие серверы (или меняем местами)

1. Выбрать начальное решение $x_{as} = x_{as}^0$
2. Цикл $t = 1, ..., t_{max}$
   (a) Цикл $k = 1, ..., k_{max}$
      i. Выбрать случайным образом решение из окрестности $x'_{as} \in N_k(x_{as})$
      ii. Применить к x'_{as} метод локального улучшения, получить локальный минимум x''_{as} по окрестности $N_k(x_{as})$
      iii. Если новый локальный минимум лучше, чем x_{as}, то $x_{as} = x''_{as}$
   (b) $Assign(x_{as}), x_{as} = x'''_{as}$
   (c) Если нет перегрузки, то STOP.
3. Предъявить наилучшее найденное решение.

2.3.2 Эксперименты и результаты

Количество виртуальных сред $|S| = 8$

Количество приложений $|A| = 20$

Накладные расходы $b_{as}^{i/o} = 1, B_s^{i/o} = 4$ для $\forall s \in S, \forall a \in A$

При проведении экспериментов на Jelastic, во время конфигурирования серверов использовались только фиксированные клаудлеты (fixed cloudlets), поэтому предполагается, что все сервера в пуле имели постоянную вычислительную мощность.

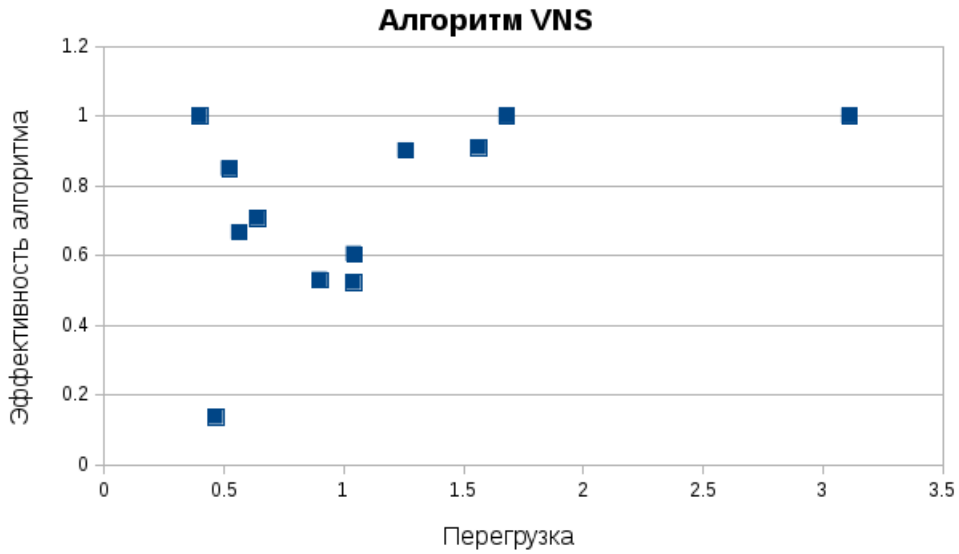
В следующей таблице 1 единица нагрузки примерно равна 40МГц тактовой частоты процессора

Пороговая нагрузка на пул ресурсов	450	450	400
Общая нагрузка на пул ресурсов	288	469	672
Перегрузка до балансировки	116	163	415
Перегрузка после балансировки	34	76	272
Коэффициент перегруженности	0.64	1.04	1.68
Улучшение после балансировки, %	70	53	34
Эффективность балансировки, %	71	60	100

Эффективность балансировки оценена как отношение максимальной перегрузки, которую мог бы снять некоторый идеальный алгоритм балансировки, к перегрузке, которую удалось снять рассматриваемому алгоритму.

В каждом эксперименте алгоритм показал улучшение, относительно начального распределения.

При высокой загруженности системы эффективность алгоритма более 90%, однако когда общая нагрузка на систему намного превосходит мощность имеющихся ресурсов, балансировка хоть и улучшает ситуацию, но в целом система так и остается перегруженной. И исправить это можно или общим снижением нагрузки, или добавлением в пул новых виртуальных сред, или



увеличением мощности уже имеющихся.

Однако при средней загруженности, когда балансировка нагрузки может быть наиболее эффективна, результат, показанный алгоритмом сложно назвать оптимальным. В среднем 64%.

2.4 Стохастический локальный поиск

2.4.1 Описание алгоритма

Обобщение классического метода локального улучшения. Алгоритм не останавливается в локальном оптимуме, а "путешествует" от одного локального оптимума к другому, используя рандомизацию окрестности и список запретов.

В рамках алгоритма допускаются следующие операции:

1. $Move(x_{as})$ перемещение приложения на другой сервер
2. $Swap(x_{as})$ меняем одно приложение на другое
3. $Assign(x_{as})$ изъять с каждого сервера по приложению и распределить их оптимально между серверами

Алгоритм SLS

1. Задать параметры алгоритма t_{max}, l_{max}, p
2. Выбрать начальное решение $x_{as} = x^0_{as}$
3. Цикл $t = 1, ..., t_{max}$
   (a) Цикл $l = 1, ..., l_{max}$
      i. Найти наилучшее соседнее решение $x'_{as} \in Move_p(x_{as}) \cup Swap_p(x_{as})$
      ii. Положить $x_{as} = x'_{as}$
   (b) Вернуться в наилучшее из l_{max} полученных решений: $x_{as} = \overline{x_{as}}$
   (c) $Assign(x_{as}), x_{as} = \overline{\overline{x_{as}}}$
   (d) Если нет перегрузки, то STOP.
4. Предъявить наилучшее найденное решение.

2.4.2 Эксперименты и результаты

Количество виртуальных сред $|S| = 8$

Количество приложений $|A| = 20$

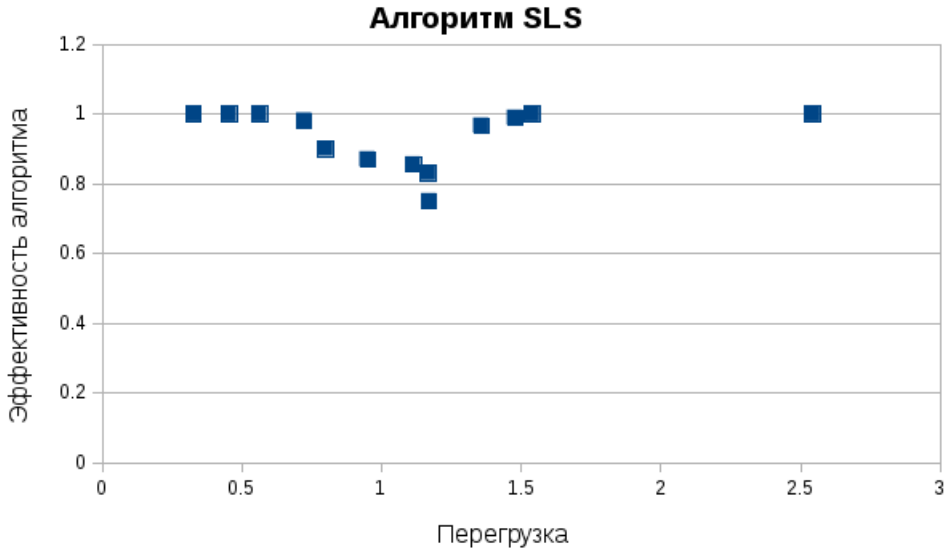
Накладные расходы $b_{as}^{i/o} = 1, B_s^{i/o} = 4$ для $\forall s \in S, \forall a \in A$

При проведении экспериментов на Jelastic, во время конфигурирования серверов использовались только фиксированные клаудлеты (fixed cloudlets), поэтому предполагается, что все сервера в пуле имели постоянную вычислительную мощность.

В следующей таблице 1 единица нагрузки примерно равна 40МГц тактовой частоты процессора

Пороговая нагрузка на пул ресурсов	365	405	365
Общая нагрузка на пул ресурсов	364	451	562
Перегрузка до балансировки	264	212	310
Перегрузка после балансировки	4	70	197
Коэффициент перегруженности	0.72	1.11	1.54
Улучшение после балансировки, %	98	67	36
Эффективность балансировки, %	98	86	100

Эффективность балансировки оценена как отношение максимальной перегрузки, которую мог бы снять некоторый идеальный алгоритм балансировки, к перегрузке, которую удалось снять рассматриваемому алгоритму.



В каждом эксперименте алгоритм показал улучшение, относительно начального распределения.

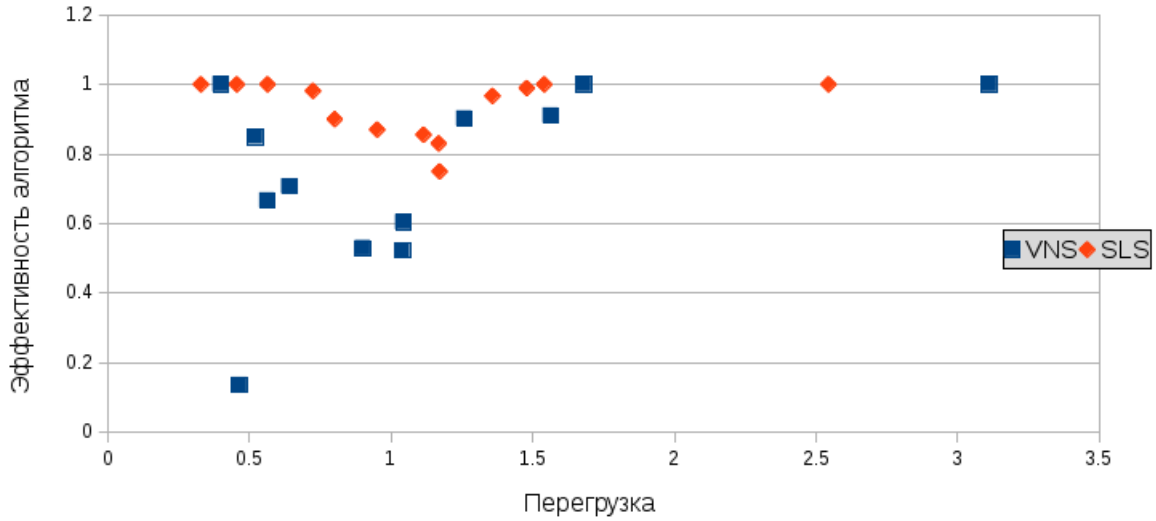
Алгоритм SLS показал хорошие результаты - более 80% эффективности при любом уровне загруженности серверов. Это означает, что предложенный алгоритм достаточно близок к идеальному и оптимально перераспределяет ресурсы.

2.5 Сравнение результатов работы алгоритмов

График и таблица ниже иллюстрирует, что при низкой и средней нагруженности виртуальных сред алгоритм стохастического локального поиска SLS работает эффективнее алгоритма поиска с чередующимися окрестностями VNS.

Средняя эффективность алгоритмов:

	Отсутствие перегрузки	Небольшая перегрузка	Большая перегрузка
VNS	67%	64%	97%
SLS	99%	88%	99%



3 APS - Application Packaging Standard

APS это стандарт, который говорит о том, как поставщики программного обеспечения предоставляют свои облачные сервисы провайдерам. APS позволяет различным облачным сервисам взаимодействовать друг с другом посредством RESTful API. Создание APS пакета (APS Package) приложения упрощает его интеграцию сервис-провайдером.

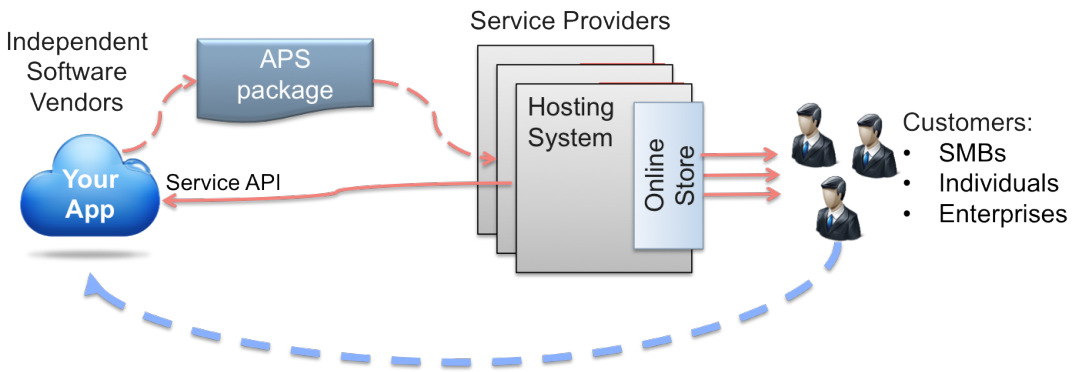
3.1 Процесс интеграции

1. Поставщик программного обеспечения создает APS-пакет, который раскрывает бизнес-логику приложения. Также пакет может содержать пользовательский интерфейс для удобста использования приложения конечными пользователями.

2. Хостинг-провайдер разворачивает APS-пакет в хостинг-системе. После этого хостинг-система сможет управлять сервисами приложения, взаимодействовать с ними посредством RESTful API.

3. Сервис-провайдеры продают сервисы приложения по отдельности или в наборе с другими ресурсами.

4. Клиенты потребляют ресурсы приложения и могут управлять ими с помощью интерфейса в контрольной панели.



3.2 Модель ресурса

Ресурс - ключевое понятие в APS. Каждое приложение предоставляет свои ресурсы APS контроллеру (APS-Controller) в JSON формате.

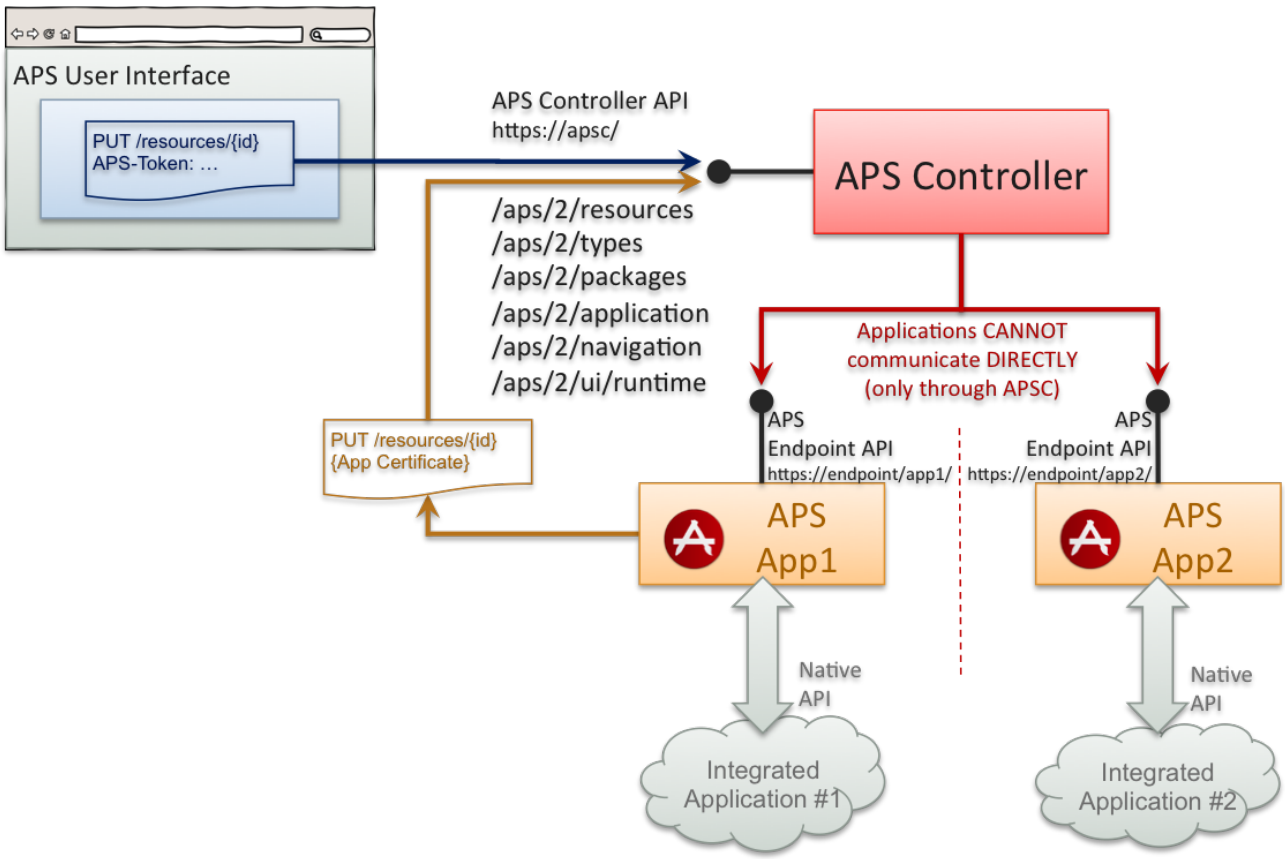
```
{
        "apsVersion": "2.0",
        "name": "JelasticEnv",
        "id": "http://domain/JelasticEnv",

        "properties": {
                "nCloudlets": {
                        "type": "integer",
                        "title": "nCloudlets"
                }
        },

        "structures": {},

        "relations": {
                "jelasticContext": {
                        "type": "http://domain/JelasticContext",
                        "required": true
                },
                "jelasticAccount": {
                        "type": "http://domain/JelasticAccount"
                },
                "applications": {
                        "type": "http://domain/Application",
                        "collection": true
                }
        }
}
```

APS ресурсы хранятся в базе данных в виде JSON объектов. Они доступны для пользователей через REST запросы к APS контроллеру. Для обращения приложения к конроллеру требуется наличие соответствующего сертификата, так как APS ресурсы защищены.

3.3 API

Application programming interface (API) - используется для вызова различных методов APS контроллера и APS приложений.

Инфраструктура APS:



APS контроллер - это центральная часть, ответственная за хранение всех ресурсов и управляющая взаимодействием между всеми компонентами инфраструктуры.

Каждое интегрированное приложение представлено своим *APS приложением*, которое предоставляет контроллеру свой адрес, например "https://endpoint/app1/ а также адрес каждого своего сервиса, например "https://enpoint/app1/service1/". *APS приложение* принимает все REST запросы от контроллера, адресованные экземляру приложения, его сервисам и ресурсам.

Основные REST операции:

Операция	**HTTP метод**	**Описание**
Create	POST	Создание ресурса - JSON объекта - в базе данных APS
Read	GET	Чтение списка ресурсов, конфигурации определенного ресурса в JSON формате
Update	PUT	Модификация ресурса путем отправления новой конфигурации в JSON формате
Delete	DELETE	Удаление ресурса из базы данных APS

4 Программная реализация

4.1 Балансировка нагрузки

Приложение, тестирующее нагрузку, реализовано на Java в виде APS-приложения с пятью сервисами: JelasticAccount, JelasticEnv, Application, JelasticContext, JelasticLoadBalancer

1. JelasticAccount - аккаунт в системе Jelastic.
   Параметры : login, password

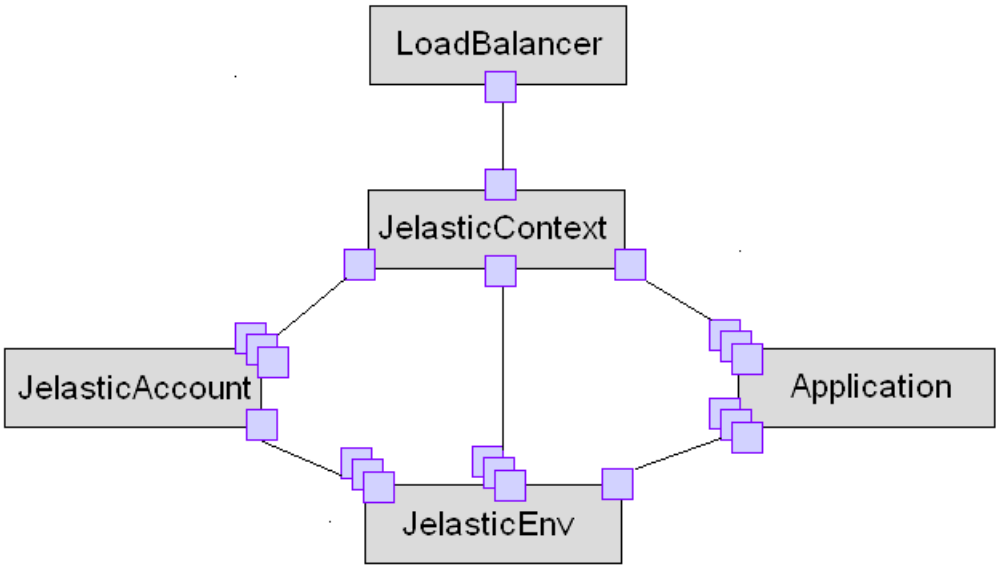
2. JelasticEnv - виртуальная среда
   Параметры : конфигурация среды (количество выделенных клаудлетов)

3. JelasticContext - набор виртуальных сред и приложений

4. Application - произвольное java-приложение
   Параметры : url (ссылка на скачивание приложения)

5. JelasticLoadBalancer - балансировщик нагрузки, который по заданному алгоритму распределяет нагрузку в JelasticContext
   Параметры : алгоритм балансировки

Балансировщик нагрузки распределяет приложения из контекста по виртуальным средам из того же контекста, может работать по двум ранее описанным алгоритмам VNS и SLS.



4.2 Эмуляция нагрузки

Для эмуляции нагрузки было реализовано небольшое java-приложение, выполняющее парсинг некоторого количества регулярных выражений в нескольких потоках. Количество выражений и потоков для каждого приложения было разным и определялось во время эксперимента. Таким образом, разные приложения создавали разную известную нагрузку на серевер в разное время.

5 Заключение

В работе была рассмотрена проблема распределения вычислительных нагрузок в виртуальных средах. Для решения задачи были выбраны два алгоритма балансировки : алгоритм поиска с чередующимися окрестностями и алгоритм стохастического локального поиска. Было разработано приложение, которое перераспределяет нагрузку в виртуальных средах Jelastic. Приложение может быть легко встроено в облачную инфраструктуру как сервис с помощью стандарта APS.

Получены результаты работы алгоритмов. Оба алгоритма балансировки улучшают начальное распределение вычислительной нагрузки, однако эффективность алгоритма стохастического поиска оказалась выше.

6 Список литературы

1. Ю.А. Кочетов "Вероятностные методы локального поиска для задач дискретной оптимизации"
   http://math.nsc.ru/LBRT/k5/Kochetov/locsearch.pdf

2. Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, Alfons Kemper "Capacity Management and Demand Prediction for Next Generation Data Centers"
   http://www.hpl.hp.com/personal/Lucy_Cherkasova/projects/papers/capacity-icws.pdf

3. Ю. А. Кочетов, Н. А. Кочетова "Задача балансировки нагрузки на серверы"
   http://www.nsu.ru/jspui/bitstream/nsu/

4. APS Standard Documentation
   http://doc.apsstandard.org/

5. Benjamin Heckmann, Ingo Stengel, Günter Turetschek, Andy Phippen "Utility Computing Simulation"

6. Jerry Rolia "Performance: Cloud computing"
   http://www.yorku.ca/mlitoiu/3rdIWCC/RoliaCascon-CC-Nov2009.pdf