Министерство образования и науки Российской Федерации

МОСКОВСКИЙ ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
(государственный университет)

ФАКУЛЬТЕТ УПРАВЛЕНИЯ И ПРИКЛАДНОЙ МАТЕМАТИКИ

КАФЕДРА ИНФОРМАТИКИ И ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ

Направление подготовки: 03.03.01 Прикладные математика и физика

Разработка DRS модуля в рамках OpenStack архитектуры

Выпускная квалификационная работа на степень бакалавра

студента 376 группы

Булаваса Владаса Висвальдовича

Научный руководитель:

Мелехова Анна Леонидовна, к.т.н.

Москва 2017

Содержание

1. Введение **3**

1.1 Облачные вычисления 3

1.2 Преимущества облачных вычислений 4

1.3. Балансировка ресурсов в облаке 5

2. Задача начального размещения **7**

2.1 Описание задачи 7

2.2 Постановка задачи 9

2.3 Обзор существующих алгоритмов 11

3. OpenStack **13**

4. Решение **15**

4.1 Идея алгоритма 15

4.2 Существующее решение 15

4.3 Внедрение решения в существующую архитектуру 17

5. Эксперимент **18**

5.1 Описание стенда 18

5.2 Выбор модели 19

5.3 Результаты 23

6. Выводы **29**

Список источников **30**

1. Введение

1.1 Облачные вычисления

Облачные вычисления (cloud computing) - это модель обеспечения повсеместного и удобного сетевого доступа по требованию к общему пулу вычислительных ресурсов (такие как сети передачи данных, серверы, хранилища данных, приложения и сервисы), которые могут быть быстро выделены и освобождены с минимальными усилиями или обращениями к провайдеру. В модели облачных вычислений выделяют 5 ключевых характеристик [1]:

- **Самообслуживание по требованию.** Потребитель может в одностороннем порядке обеспечивать изменение объема предоставляемых “облачных” ресурсов, без взаимодействия с представителем поставщика услуг.
- **Обеспечение сетевого доступа.** Ресурсы доступны по сети с использованием стандартных механизмов с любых устройств (мобильные телефоны, планшеты, ноутбуки или рабочии станции).
- **Объединение ресурсов в коллекции.** Вычислительные ресурсы провайдера объединяются для обслуживания большого числа потребителей, динамически перераспределяя физические и виртуальные ресурсы согласно потребностям потребителей. При этом клиент не имеет контроля над фактическим распределением ресурсов, имея возможность задавать только более высокий уровень абстракции, например географическое местоположение виртуальной машины

(страна, город, дата-центр). В качестве примера ресурсов можно привести размер хранилища данных, вычислительные мощности, память и канал сетевого доступа.

- **Эластичность.** Предоставляемые ресурсы могут быть получены или освобождены в соответствии с уровнем потребления пользователя, как правило, в автоматическом режиме.
- **Учет потребления.** Облачные системы автоматически контролируют и оптимизируют использованные ресурсы в различных уровнях абстракции (например, хранилища данных, вычислительные мощности, сетевой канал). Использование ресурсов может быть отслежено, обеспечивая прозрачность как для поставщика услуг, так и для клиента использующего облачные ресурсы. [1] Эта характеристика получила название “Платеж по мере использования” (Pay-As-You-Go).

1.2 Преимущества облачных вычислений

Технологии облачных вычислений имеют ряд преимуществ перед традиционными методами предоставления вычислительных ресурсов (utility computing).

Во-первых, облачные вычисления позволяют более эффективно использовать ресурсы как провайдеру, так и клиентам. Таким образом, клиент, использовав некоторое количество часов реального вычислительного времени, платит только за него и ему не приходится заботится о простое вычислительных ресурсов. С другой стороны провайдер может выделять ресурсы клиентам по требованию, позволяя более эффективно использовать имеющееся оборудование. .

Во-вторых, расположение виртуальных машин в облаке позволяет максимально быстро расширять объем предоставляемых ресурсов при пиковых нагрузках, позволяя сохранить работоспособность и доступность клиентских виртуальных машин. Кроме того, это расширение может происходить в автоматическом режиме, без участия провайдера. В то же время, в случае предоставления физического хостинга, расширение ресурсов занимает значительно большее время и необходимость явного участия в этом сотрудников на стороне провайдера.

1.3. Балансировка ресурсов в облаке

Из-за специфики облачных технологий связанных с перераспределением задач и предоставлением ресурсов по требованию. Возникает задача оптимального использования этих ресурсов. В действительности, достаточно сложно предсказать вычислительные ресурсы, которые будут затребованы заранее, могут возникать пиковые нагрузки, которые необходимо правильно обрабатывать, кроме того необходимость в динамической балансировке возникает при добавлении или выводе физических машин из облака.

Задача динамического распределения ресурсов в свою очередь делится на ряд подзадач [10]:

- Выбор недозагруженных серверов для их последующего заполнения
- Выбор перегруженных серверов для их последующей разгрузки
- Выбор виртуальных машин для миграции - определяются виртуальные машины, которые должны быть мигрированы
- Выбор физической машины для миграции на нее виртуальной машины

- Выбор времени миграции - определение оптимального времени и порядка миграции

Также стоит заметить, что разделяют две смежные задачи (DRS и DPM), которые вообще говоря по своему смыслу являются конфликтующими. Задача управления ресурсами - DRS (Dynamic resource scheduling) была рассмотрена выше, в то время как основной целью DPM (Dynamic power management) является оптимизация энергопотребления и как следствие уплотнения нагрузки на ограниченном числе включенных серверов. При решении этих задач важно соблюдать баланс в суммарном решении. [11]

Данная работа посвящена решению задачи изначального размещения виртуальных машин.

Cloud Computing Architecture

2. Задача начального размещения

2.1 Описание задачи

Среди задач облачных вычислений можно выделить ряд оптимизационных задач, решение которых позволит повысить полезную (клиентскую) нагрузкуч с использованием тех же ресурсов. Одной из наиболее важных таких задач является задача начального размещения (initial placement). Задача возникает при разворачивании новых виртуальных машин в облаке.

Жизненный цикл задачи можно разделить на 2 части: этап изначального размещения виртуальной машины и этап жизни (включая живую миграцию).

Живая миграция - это процесс клонирования виртуальной машины с одной физической машины на другую, при этом в процессе клонирования сохраняется работоспособность виртуальной машины. Технология живой миграцию полезна для балансировки нагрузки, консолидации ресурсов и обеспечения отказоустойчивости. [12]

В процессе живой миграции снимок виртуальной машины сначала копируется на другой сервер, а затем производится несколько итераций синхронизации двух инстансов. После этого оригинальная виртуальная машина на первом сервере подменяется склонированой версией на втором сервере.

Зная реальную статистику использования ресурсов, мы можем точнее предсказывать дальнейшее поведение виртуальных машин, что в свою очередь даст возможность для более эффективной оптимизации облака. С другой стороны, живая миграция сама по себе требует некоторое количество ресурсов. [12] В отличие от статической миграции, при живой миграции значительно повышается нагрузка сервера, которые являются источников и приемником. Так, в случае обычной миграции память виртуальной машины один раз копируется и сторонние затраты на это пропорциональны объему памяти виртуальной машины. При живой миграции происходит итерационное копирование образа, то есть образ сначала копируется, при следующей итерации копируются только изменившиеся за это время страницы памяти и так до тех пор пока не будет достигнуто некоторое условие окончания итерационного копирования, после этого первая виртуальная машина останавливается и происходит копирования оставшихся страниц памяти. [12] Здесь вводится такое понятие, как “стоимость миграции” (migration cost) - то есть затраты, которые мы несем при живой миграции. Очевидно, что стоимость миграции еще не размещенной виртуальной машины равна нулю.

Из сказанного выше следует, что правильное изначальное размещение позволяет сэкономить ресурсы. Но это не позволяет решать “динамических” задач, связанных с изменением средней нагрузки на виртуальные машины.

2.2 Постановка задачи

В данной работе в качестве основных ресурсов будут рассматриваться процессорное время (ЦПУ), оперативная память и сеть (точнее, пропускная способность сети). Таким образом, каждой виртуальной машине поставим в соответствие вектор из трех элементов - потребности этой машины в соответствующих ресурсах, и каждому физическому серверу также вектор из трех элементов - доступные ресурсы на этой машине. Стоит заметить, что это не все ресурсы машины могут быть использованы под виртуализацию, так как часть из них необходима для работы инфраструктуры.

Решением задачи изначального размещения виртуальной машины является алгоритм, на вход которому подаются список виртуальных машин, которые необходимо разместить (будем называть это множество M), а также список доступных физических серверов (множество K). Запрос на добавление виртуальной машины можно описать в виде вектора $v = \{(r_{1, min}, r_{1, max}), (r_{2, min}, r_{2, max}), ...\}$, где $r_{i, min}$ - минимальные требования i-го ресурса, а $r_{i, max}$ - максимальные. Физический сервер можно описать вектором $p = \{a_{1,} a_2, ...\}$, где a_i - количество доступного i-го ресурса. Таким образом, на вход алгоритму подается вектор $V = \{v_1, v_2, …\}$ - виртуальные машины для размещения и вектор $P = \{p_1, p_2, ...\}$ - физические сервера. [2]

Все ограничения на решение можно разделить на две части: жесткие и мягкие. К первым, например, относятся ограничения на минимальное количество физических ресурсов для каждой виртуальной машины. К мягким ограничениям можно отнести минимизацию активных серверов, увеличению

количества “запасных” ресурсов, на случай незапланированных нагрузок. Кроме того, тесно взаимодействующие виртуальные машины имеет смысл располагать на одном физическом сервере. Примером таких тесно взаимодействующих виртуальных машин могут стать системы с нагрузкой на нескольких узлах - так называемые системы с N-tier архитектурой.

Для построения алгоритма необходимо формализовать требования, которые оказываются противоречивыми. С одной стороны, если пытаться равномерно загрузить все ноды кластеры, мы будем терять энергоэффективность нашего облака: при малой нагрузке на облако имеет смысл консолидировать часть физических машин. С другой стороны, загружать один сервер, пока остальные простаивают может быть неэффективно, например, с точки зрения использования ЦПУ или производительности гостевых систем. Некоторые технологии позволяют управлять энергопотреблением, например, изменением вольтажа процессоров или отключением неиспользуемых машин. Так например, система управления ресурсами в VMware ESXi позволяет регулировать энергопотребление, таким образом в режиме низкого энергопотребления будет активнее использоваться консолидированные ресурсы, возможно в ущерб производительности. Обратно, в режиме высокой производительности предпочтительней будет оставлять дополнительные ресурсы на каждой ноде. [6]

Введем переменные, которые будут использоваться в некоторых алгоритмах:

Используемые ресурсы $U_i^d = \frac{c_i^d - a_i^d}{c_i^d}$ (resource usage), где c_i^d - общее количество соответствующего ресурса на i-ой ноде, a_i^d - доступное

количество соответствующего ресурса на i-ой ноде. Принимает значение от 0 до 1.

Сбалансированность ресурсов $B_i = \frac{min_d U_i^d}{<U_i^d>_d}$, где $< U_i^d >_d = \frac{\sum_d U_i^d}{D}$ (среднее использование ресурсов), D - количество ресурсов. Принимает значение от 0 до 1. Мы хотим максимизации этого значения.

2.3 Обзор существующих алгоритмов

Поставленная задача может быть сведена к классической задаче многомерного рюкзака. В данном случае одним измерением будет является некоторая метрика. К сожалению, эта задача в общем виде является NP-полной. То есть точное решение будет работать за экспоненциальное время, и если на малой инфраструктуре оно может отрабатывать за незначительное время и практически не потреблять ресурсов, то на больших промышленных датацентрах с десятками тысяч машин эта задача будет невыполнима за реальное время. Стоит учитывать, что в зависимости от реализации, мы можем иметь дело либо с одной задачей размещения N виртуальных машин по K серверам, так и с N задачами размещения одной виртуальной машины на одном из K серверов. И если подход ко второму случаю можно применить к первому, то обратное неверно. В частности вторая задача не является NP-полной (хотя ее решение и менее эффективно), поэтому в коммерческих решениях часто решают более простую задачу независимого размещения каждой виртуальной машины.

К решению общей задачи существуют различные подходы: можно использовать какие-то вероятностные или приближенные методы для

нахождения решения (так, например, RedHat в oVirt’е использует свой пакет OptaPlanner для нахождения приближенного решения [7]). Или же можно использовать некоторый “жадный” алгоритм. Ниже будет произведен обзор существующих алгоритмов [3]:

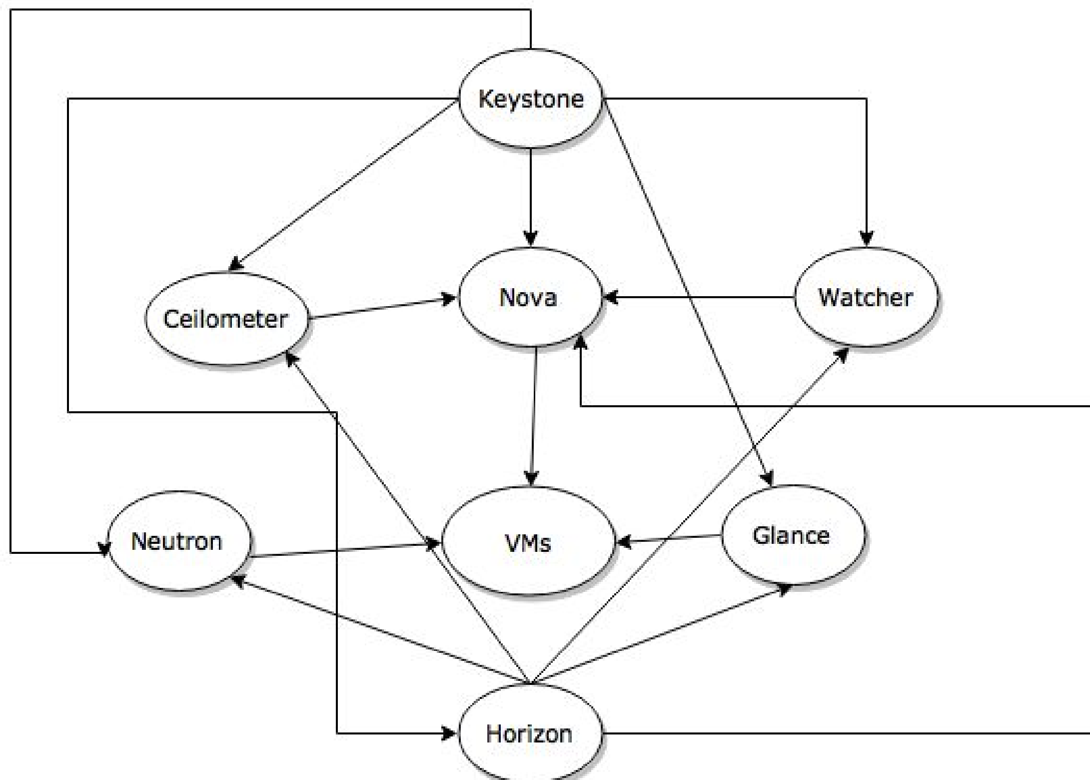
- **First-fit (round-robin) -** жадный алгоритм, идея которого заключается в том, чтобы разместить каждую виртуальную машину на первый сервер, который будет удовлетворять требованиям. В следствии детерминированного обхода серверов, они будут заполняться последовательно и как результат, мы будем иметь неравномерную нагрузку на физические машины. [13]
- **Random-fit -** жадный алгоритм, который в отличии от first-fit выбирает первый сервер не детерминистически, в результате более равномерная нагрузка, в отличие от first-fit. Этот алгоритм имеет обратную проблему: при низкой нагрузке на облако в целом, после такого размещения скорее всего понадобится консолидация ресурсов и как следствие необходимость живой миграции. [13]
- **Min-Min -** эвристический алгоритм, заключающийся в последовательном размещении виртуальных машин, отсортированных по требованиям к CPU на сервера, отсортированные по доступным вычислительным мощностям. Вычислительная мощность один из самых важных ресурсов, но не единственный. [4]
- **Max-Min -** аналогичный предыдущему эвристический алгоритм, с тем отличием, что сортирует список виртуальных машин в обратном порядке, таким образом размещая в первую очередь самые требовательные виртуальные машины. [4]

- **CPULoad -** алгоритм размещает каждую виртуальную машину на тот физический сервер, на котором в данный момент использование CPU минимально. Данный алгоритм используется в проекте OpenNebula. [5]

3. OpenStack

В качестве примера реализации концепции облачных вычислений будет рассмотрен комплекс проектов OpenStack.

OpenStack - это облачная платформа для организации и управления инфраструктурой для создания виртуальных машин по требованию. Эта система позволяет создавать виртуальные машины с необходимым функционалом и ресурсами. Сама платформа представляет из себя ряд независимых сервисов, каждый из которых выполняет некоторый функционал и взаимодействующие через REST API (REST API - это модель предоставления интерфейса через протокол HTTP [9]). Далее будут рассмотрены основные сервисы, входящие в состав архитектуры OpenStack.



- **Nova.** Управляет жизненным циклом виртуальных машин в окружении OpenStack. Отвечает за создание, распределение и отключение виртуальных машин по требованию. В частности выполняет задачи изначального размещения виртуальных машин. Сам по себе проект состоит из нескольких сервисов, взаимодействующих через очередь сообщений (AMQP). В них входит непосредственно демон compute, управляющий виртуальными машинами и scheduler, занимающийся изначальным размещением виртуальных машин.
- **Keystone.** Модуль, управляющий аутентификацие пользователей и сервисов. Содержит информацию о всех предоставляемых API и права на их вызов, а также все ограничения, такие как количество виртуальных машин и количество ресурсов.
- **Glance.** Хранилище образов виртуальных машин. Здесь хранятся все созданные пользователями образы и отсюда они загружаются с помощью Nova в гипервизор. Сами образы могут храниться как в

обычной файловой системе, так и в объектном хранилище, таком как Swift.

- **Neutron.** Модуль Neutron представляет из себя сервис виртуализации сети для виртуальных машин и других сервисов. При помощи Neutron можно конфигурировать виртуальные сети со сложной топологией.
- **Ceilometer.** Модуль для обеспечения мониторинга за компонентами OpenStack и виртуальными машинами. Занимается сбором метрик и предоставлении доступа к ним другим сервисам.
- **Horizon.** Графический пользовательский веб интерфейс для управление системой. Позволяет добавлять виртуальные машины, образы, виртуальные сети, а также дает пользователям доступ к статистике и состоянию виртуальных машин. [8]

4. Решение

4.1 Идея алгоритма

В данной работе будет предложен алгоритм решения задачи для начального размещения одной виртуальной машины. В качестве вектора метрик будут использовать следующие ресурсы: ЦПУ, память и использование диска.

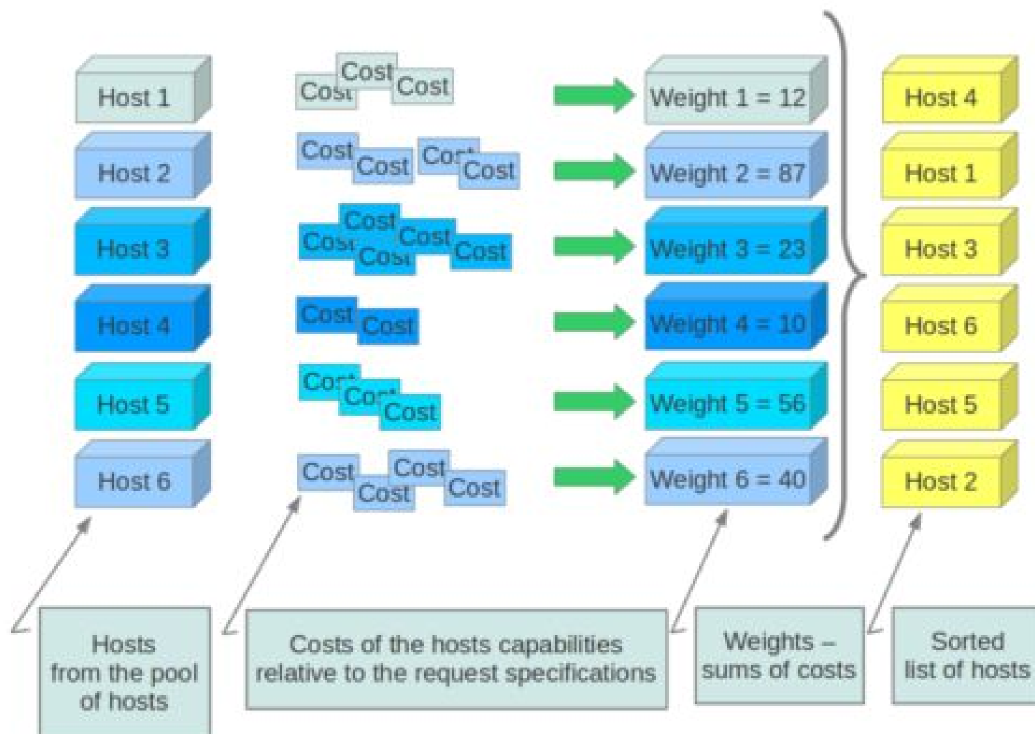
Алгоритм основан на максимизации параметра B_i для этого мы будем основываться на алгоритме Min-Min (вместо загруженности CPU будет сортировать по среднему использованию всех ресурсов) отдавая приоритет тем серверам, размещение на которых улучшит параметр B_i.

Алгоритм реализован в рамках существующего планировщика OpenStack.

4.2 Существующее решение

В текущей реализации модуля Nova уже реализован планировщик для изначального размещения виртуальных машин. Он оценивает “вес” каждого физического сервера для добавляемой виртуальной машины и выбирает лучший из них. Оценка в свою очередь производится по таким параметрам, как объем свободной оперативной памяти, занятость диска, количество устройств и связь виртуальных машин в группы. Каждый из этих параметров вычисляется для каждого сервера, нормируется и учитывается в суммарном “весе” виртуальной машины с некоторым коэффициентом. Хост для размещения в нем виртуальной машины выбирается в приоритете, заданном весами.

В существующем подходе никак не решается проблема сбалансированности ресурсов. То есть если в облако поступит виртуальная машина с большими потребностями в каком-то ресурсе, то текущий алгоритм может разместить его на машину с дефицитом этого ресурса и избытком других и как результат невозможность добавить туда новые виртуальные машины.



4.3 Внедрение решения в существующую архитектуру

В качестве улучшения существующего алгоритма размещения предложено также учитывать степень сбалансированности использования ресурсов, как один из вкладов в вес сервера.

Вкладом в вес будет является величина B_i описанная ранее в данной работе. Кроме того, при вычислении использования ресурсов будет учитываться также ресурсы необходимые для добавляемой виртуальной

машины, то есть мы максимизируем сбалансированность с учетом новой виртуальной машины.

Для внедрения этого алгоритма потребовалось модифицировать код проекта Nova OpenStack, а именно модуль Scheduler. Для внедрения алгоритма потребовалось добавить модуль в список модулей измеряющих веса, который бы вычислял вес по описанному выше алгоритму.

Архитектура модуля измеряющего вес устроена таким образом, что для добавления новых счетчиков достаточно добавить файл с этим счетчиком, унаследованным от базового класса в рабочую директорию с остальными счетчиками.

5. Эксперимент

5.1 Описание стенда

Для проведения эксперимента был собран стенд из 3 машин, одна из которые играла роль управляющего сервера и две остальные - вычислительные сервера.

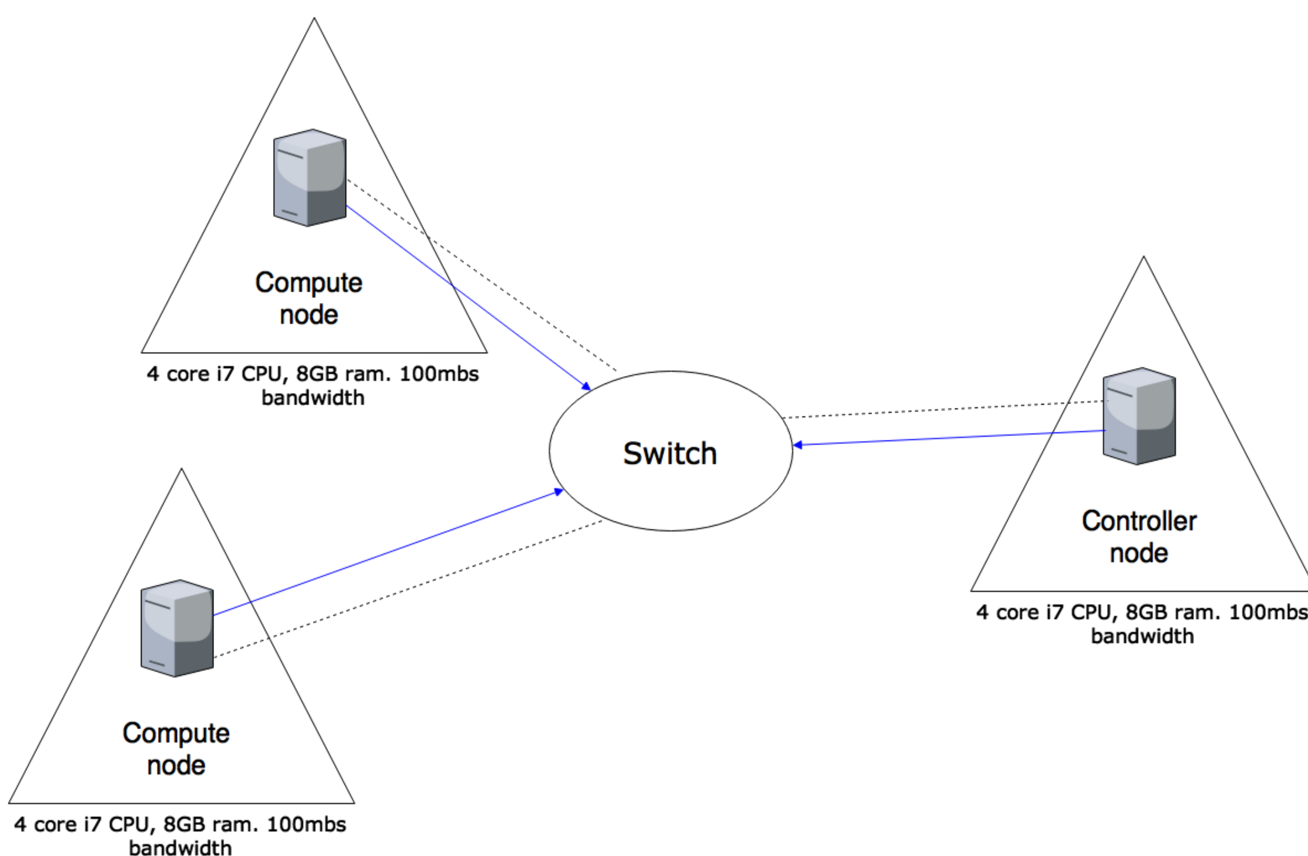
На всех машинах установлены четырехъядерные процессоры Intel Core i7-3770k (3.5 ггц), 8 гигабайт оперативной памяти и 100 мегабит в секунду сетевой канал.

У каждой машины два сетевых интерфейса. Один из них используется для сервисов OpenStack. Другой же через бридж пробрасывается в виртуальные машины.

Основной набор сервисов был установлен на управляющий сервер, в том числе планировщик. На вычислительные сервера были установлены и сконфигурированы агенты Nova для обеспечения работы виртуальных машин

и Neutron для создания виртуальных сетевых адаптеров в виртуальных машинах.

Окружение разворачивалось с минимальным набором сервисов. Кроме базового набора (Nova, Neutron, Horizon, Glance, Keystone, Cinder) был установлен Ceilometer для сбора телеметрии с виртуальных машин. Процесс конфигурирования дополнительных сервисов оказался довольно плохо документирован. [8]



5.2 Выбор модели

Алгоритм был реализован и протестирован на стенде, но для проверки состоятельности алгоритма в реальных условиях этого недостаточно.

Эффективность алгоритма была бы более наглядна на системах с большим количеством серверов и виртуальных машин.

Для проверки описанного в данной работе алгоритма была подготовлена симуляция на основе фреймворка CloudSim. Для симуляции были реализованы 3 алгоритма: Round-Robin, существующий алгоритм в Nova OpenStack и алгоритм предложенный выше (resource balanced max).

Для проведения симуляции необходимо было подготовить сценарий поступления виртуальных машин в облако. Сценарий содержал в себе информацию о облаке (количество серверов и их ресурсы) и информацию о виртуальных машинах в очереди (количество виртуальных машин и требуемые ресурсы). В качестве ресурсов использовались MIPS (million instructions per second - количество миллионов инструкций в секунду, выполняемых процессором), RAM (в мегабайтах) и ширина сетевого канала (в килобитах в секунду). Было создано три сценария:

- В первом сценарии облако содержало 10 физических серверов и 100 виртуальных машин в очереди. Объемы ресурсов, предоставляемые физическим серверами и запрашиваемые виртуальными машинами выбирался следующим образом:
    - MIPS
        - Сервер: выбирался случайно из пула (5000, 10000, 20000)
        - ВМ: выбирался по равномерному распределению из отрезка от 500 до 1500
    - RAM
        - Сервер: выбирался случайно из пула (8 Тб, 16 Тб и 32 Тб)
        - ВМ: выбирался по равномерному распределению из отрезка от 100 Гб до 300 Гб
    - Ширина сетевого канала

        - Сервер: выбиралась случайно из пула (500 мб/сек, 1 гб/сек и 2 гб/сек)
        - ВМ: выбиралась по равномерному распределению из отрезка от 50 мб/сек до 150 мб/сек
- Во втором сценарии облако содержало 50 физических серверов и 750 виртуальных машин в очереди. Объемы ресурсов:
    - MIPS
        - Сервер: выбирался случайно из пула (5000, 10000, 20000, 40000)
        - ВМ: выбирался случайно из пула (1000, 2000, 4000, 8000, 16000)
    - RAM
        - Сервер: выбирался случайно из пула (8 Тб, 16 Тб, 32 Тб и 64 Тб)
        - ВМ: выбирался случайно из пула (20 Гб, 40 Гб, 80 Гб, 160 Гб, 320 Гб)
    - Ширина сетевого канала
        - Сервер: выбиралась случайно из пула (500 мб/сек, 1 гб/сек, 2 гб/сек и 4 гб/сек)
        - ВМ: выбиралась случайно из пула (10 мб/сек, 20 мб/сек, 40 мб/сек, 80 мб/сек, 160 мб/сек)
- В третьем сценарии облако содержало 50 физических серверов (те же, что и во втором) и 650 виртуальных машин в очереди. Объемы ресурсов:
    - MIPS
        - ВМ: выбирался из нормального распределения с параметрами (600, 300)

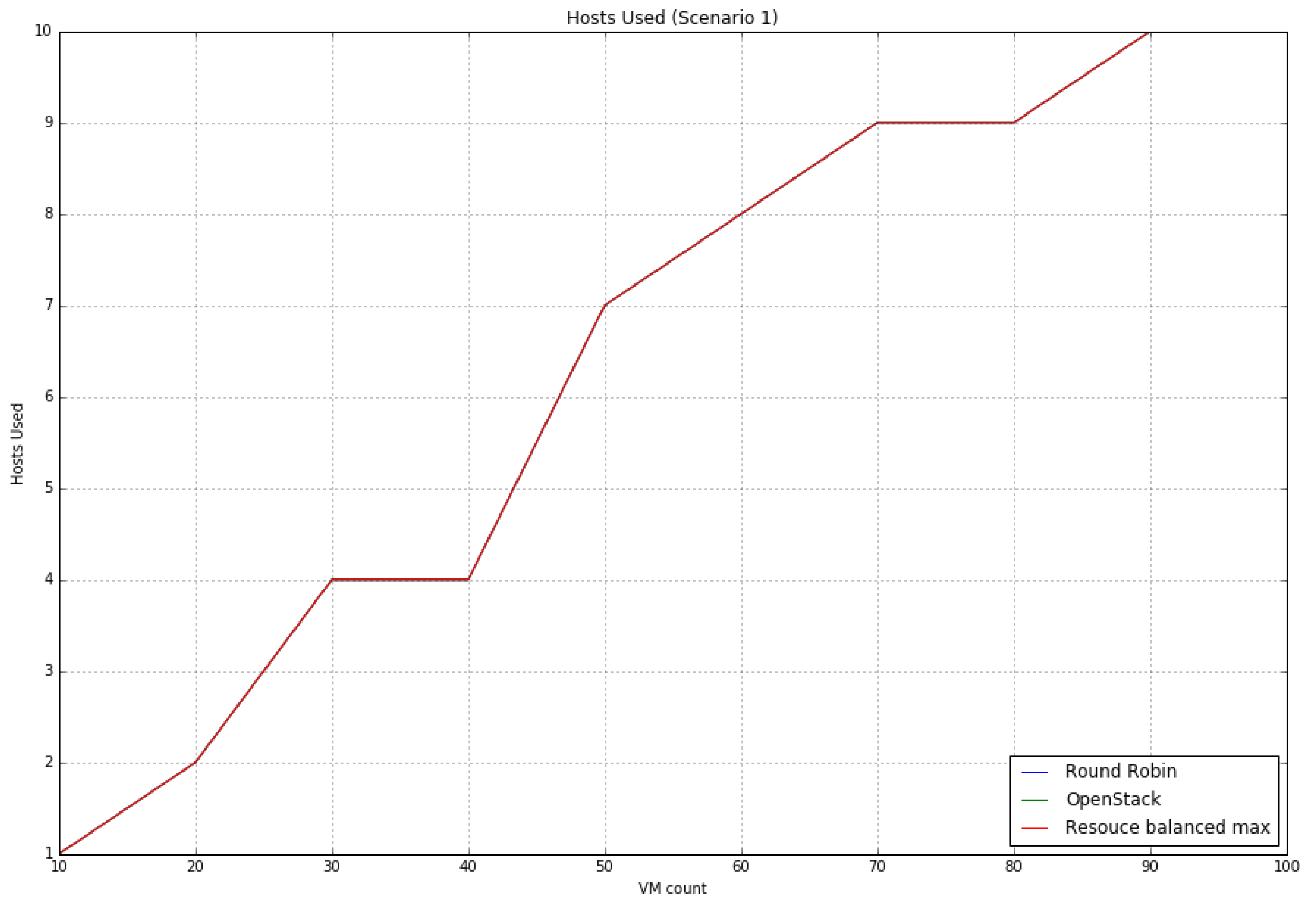
- RAM
  - ВМ: выбирался из нормального распределения с параметрами (125000, 50000)
- Ширина сетевого канала
  - ВМ: выбиралась из нормального распределения с параметрами (60000, 30000)

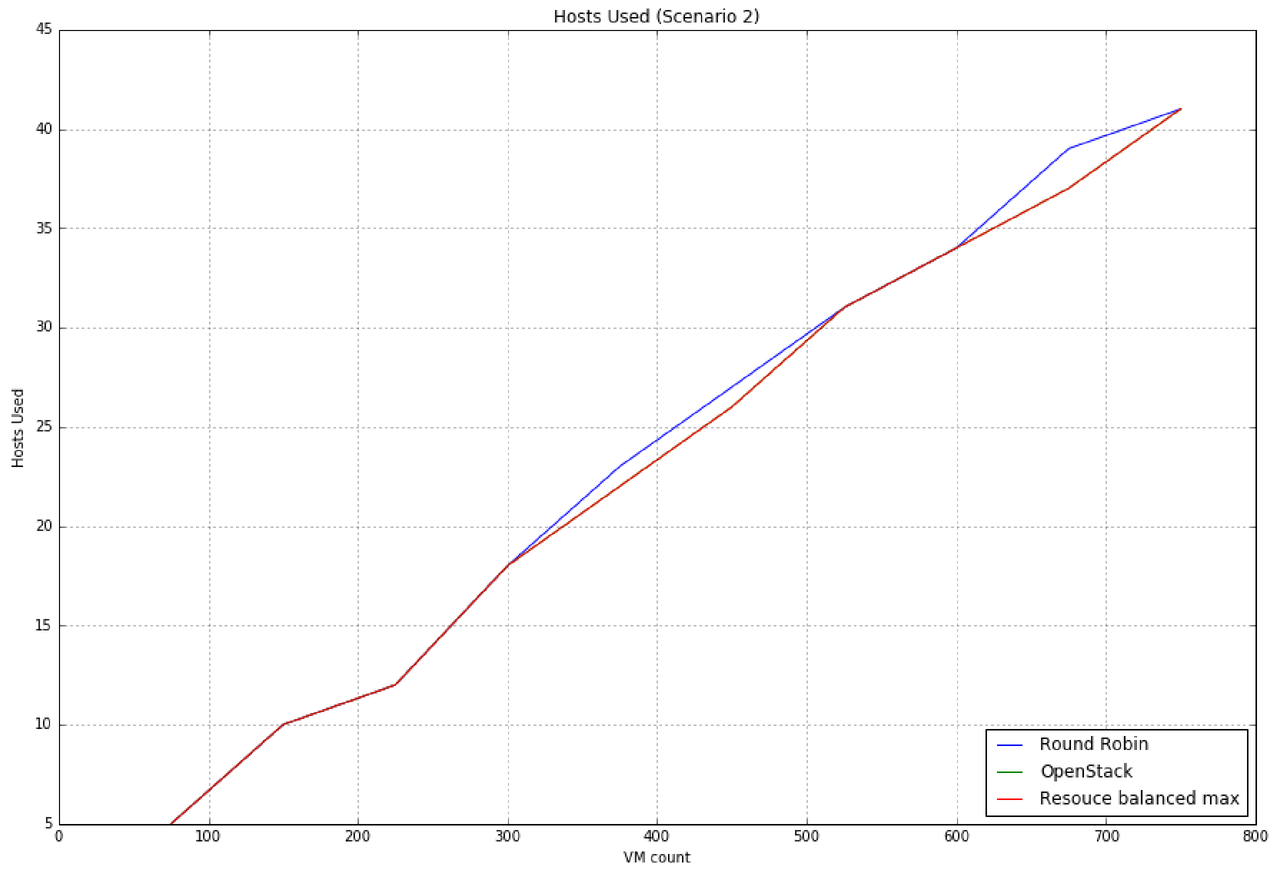
Для оценки полученных размещений были использованы 3 метрики:

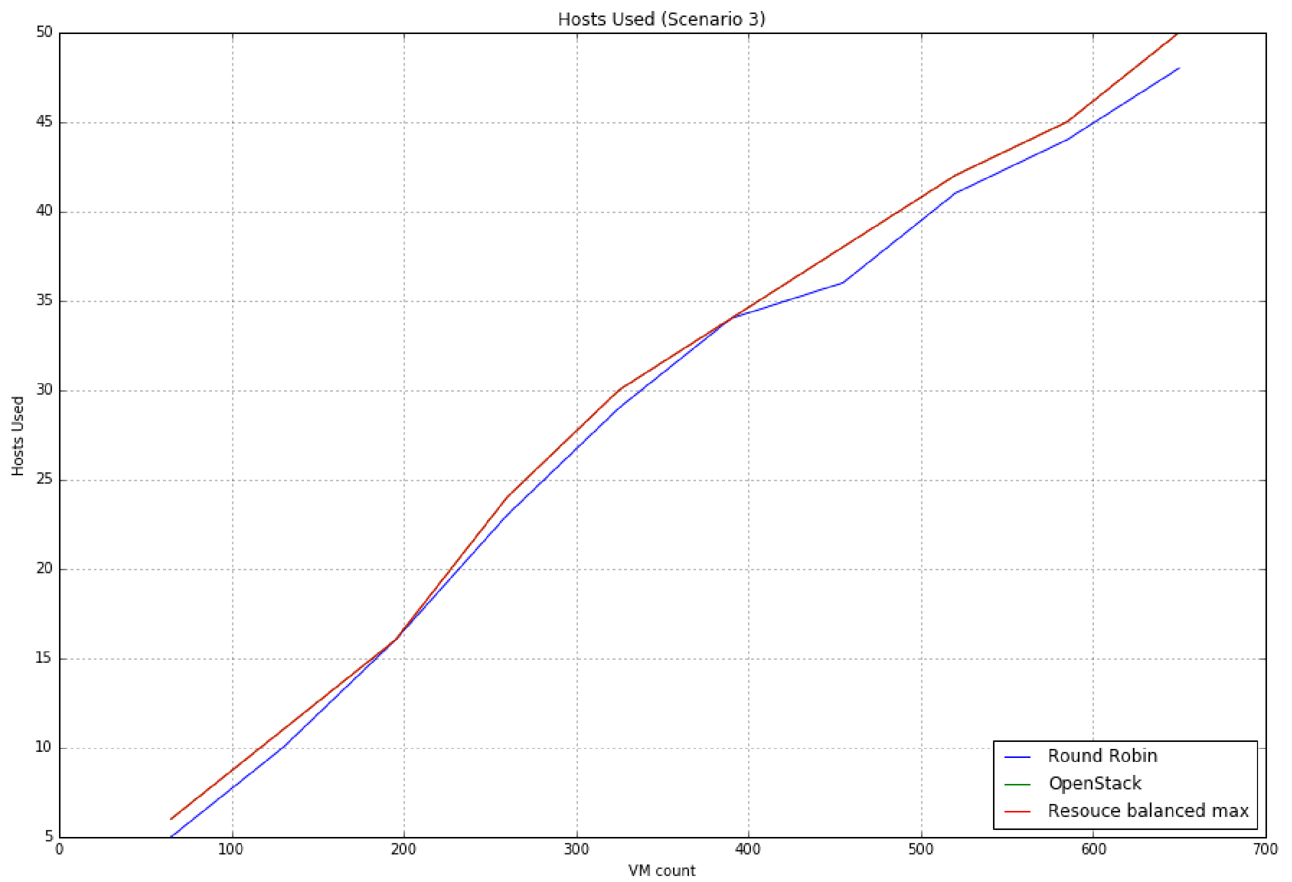
- количество использованных алгоритмом физических серверов
- средняя использование наименее используемого ресурса на серверах (для активных серверов)
- средняя величина параметра B_i (см раздел 2.2) (для активных серверов)

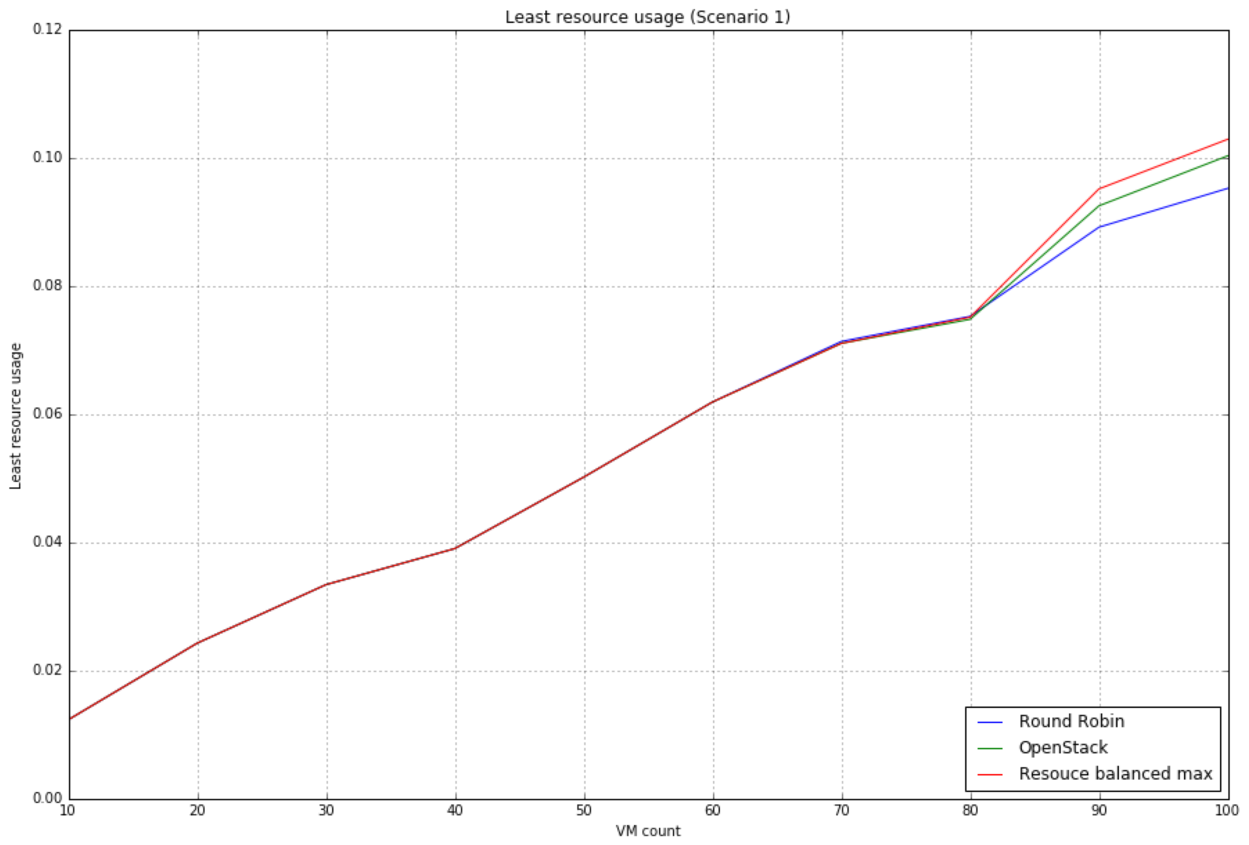
Ниже приведены результаты симуляции на описанных выше данных.

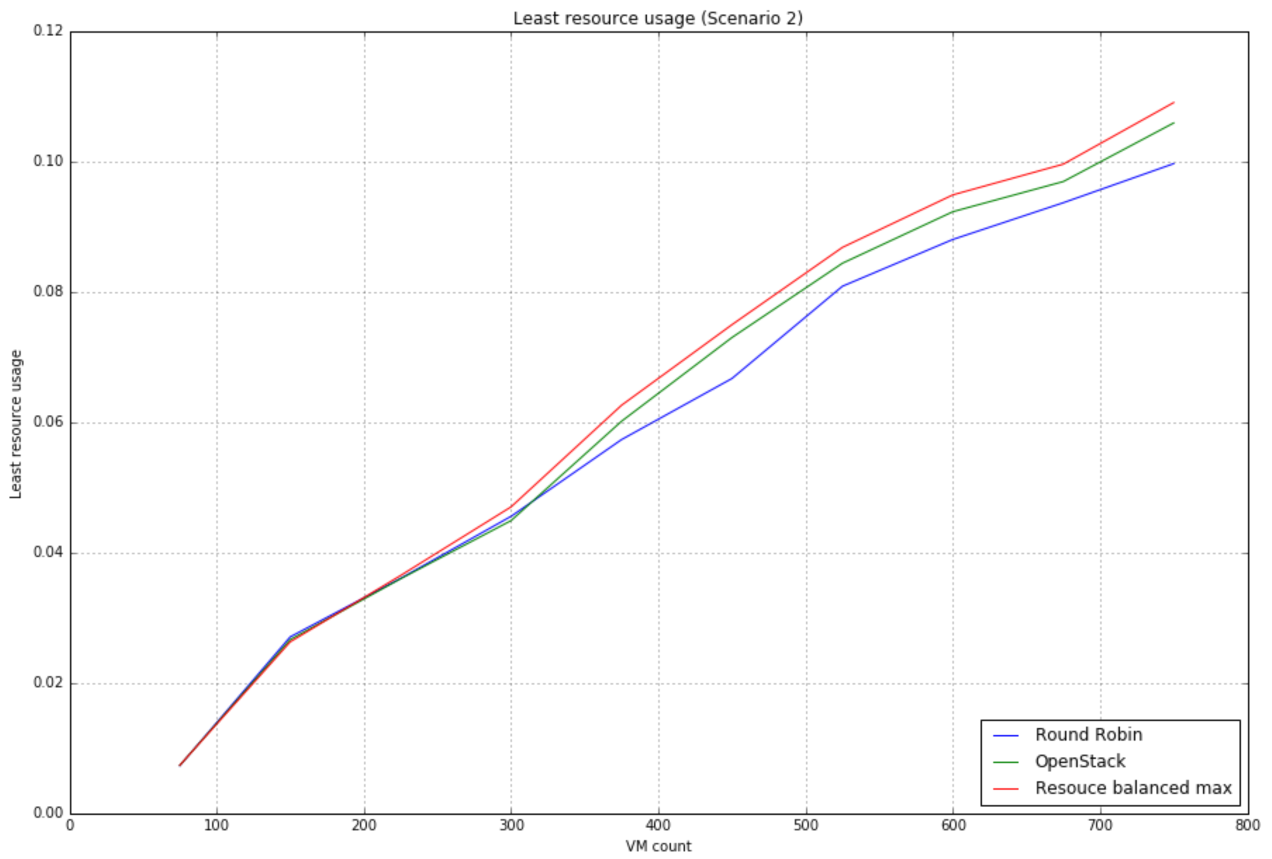
5.3 Результаты

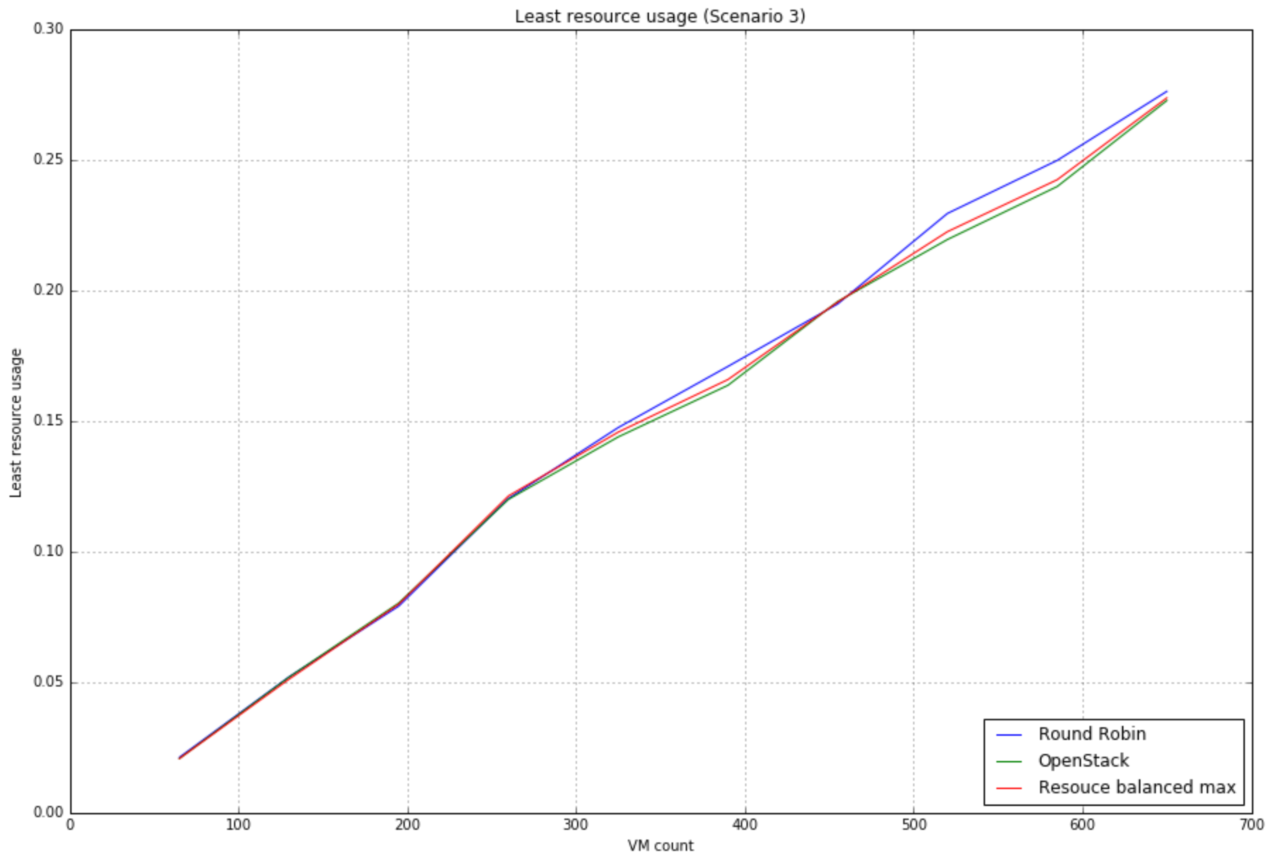


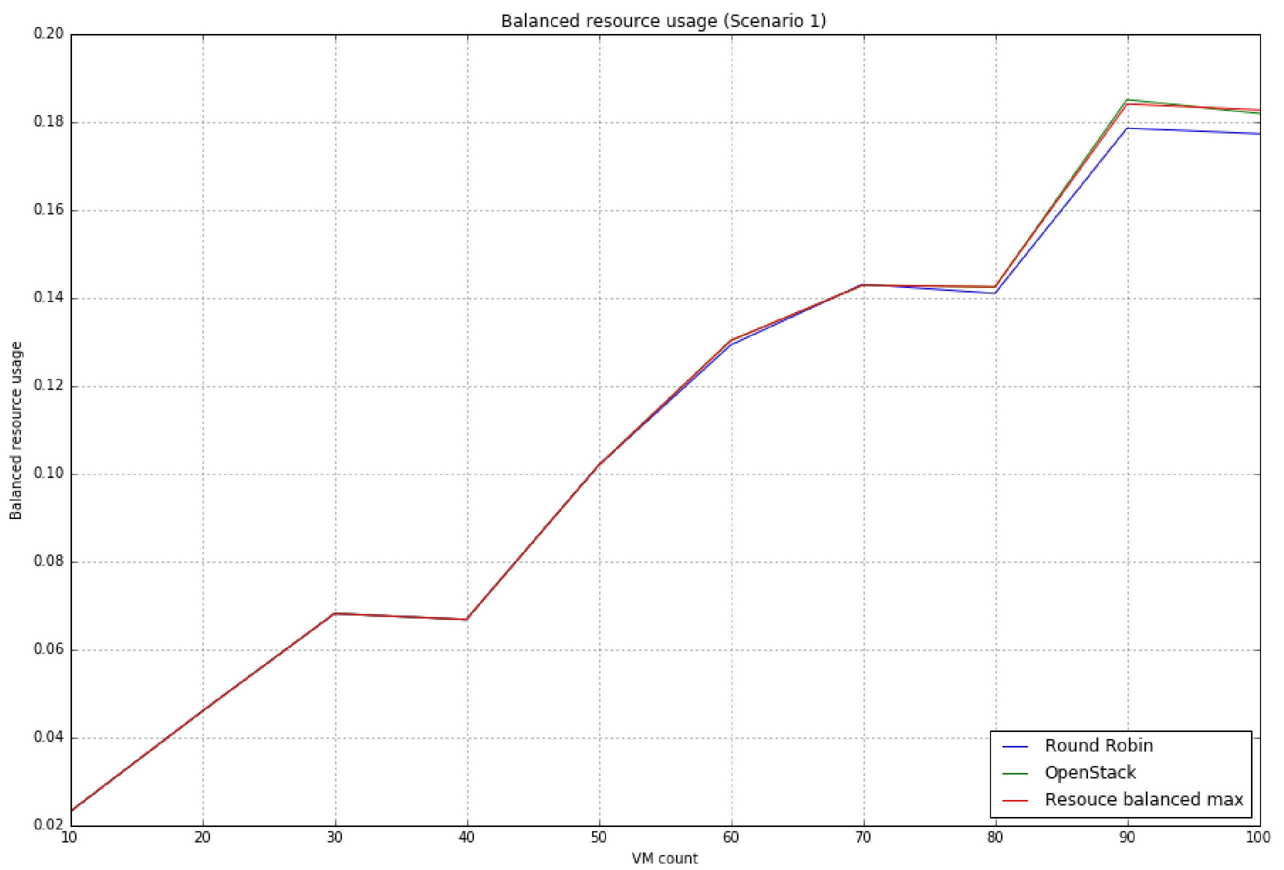

Hosts Used (Scenario 2)
Hosts Used
VM count
Round Robin
OpenStack
Resouce balanced max
0
100
200
300
400
500
600
700
800
5
10
15
20
25
30
35
40
45

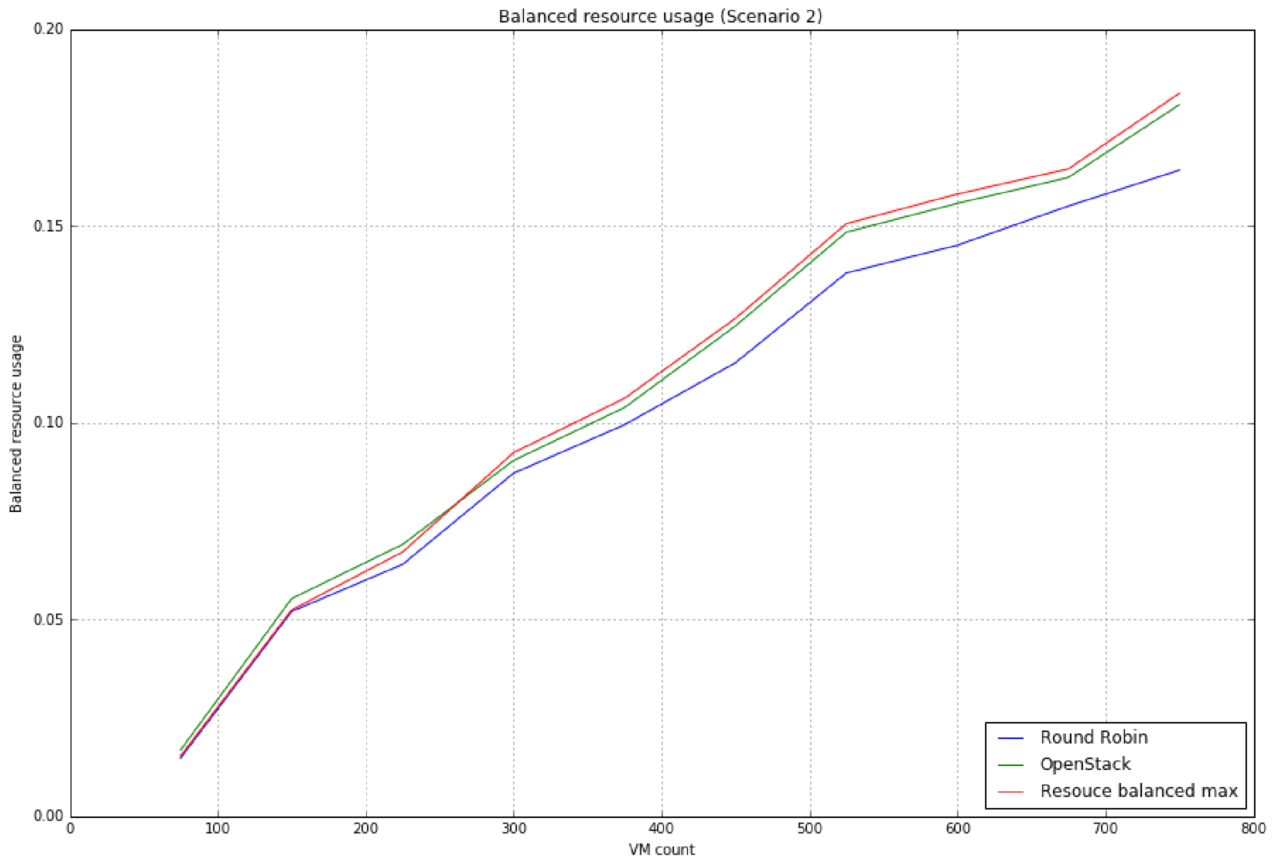

Hosts Used (Scenario 3)
Hosts Used
VM count
Round Robin
OpenStack
Resouce balanced max
0
100
200
300
400
500
600
700
5
10
15
20
25
30
35
40
45
50

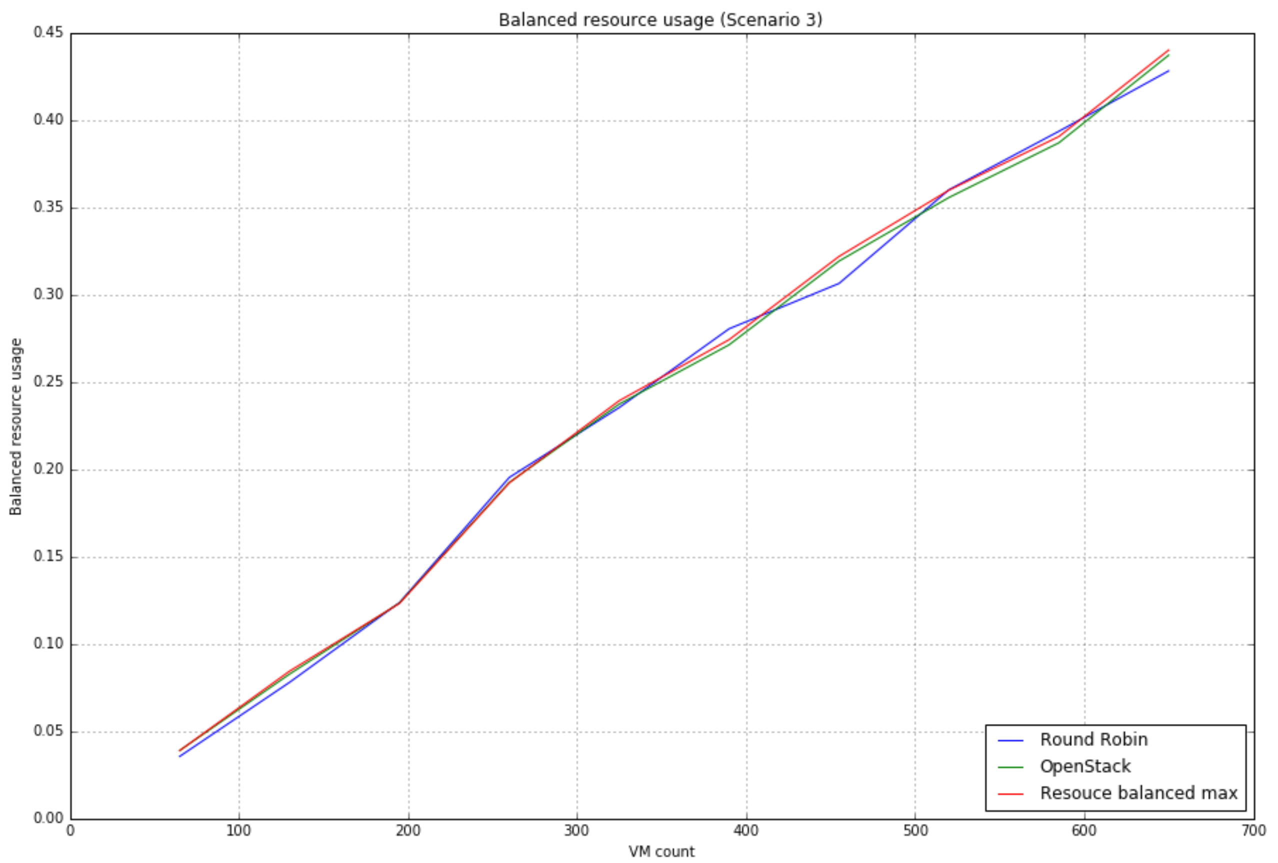

Least resource usage (Scenario 1)
Least resource usage
VM count
Round Robin
OpenStack
Resouce balanced max
0.00
0.02
0.04
0.06
0.08
0.10
0.12
10
20
30
40
50
60
70
80
90
100


Least resource usage (Scenario 2)
Least resource usage
VM count
Round Robin
OpenStack
Resouce balanced max
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0
100
200
300
400
500
600
700
800


Least resource usage (Scenario 3)
Least resource usage
VM count
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
100
200
300
400
500
600
700
Round Robin
OpenStack
Resouce balanced max


Balanced resource usage (Scenario 1)
Balanced resource usage
VM count
0.20
0.18
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
10
20
30
40
50
60
70
80
90
100
Round Robin
OpenStack
Resouce balanced max


Balanced resource usage (Scenario 2)
Balanced resource usage
VM count
0.20
0.15
0.10
0.05
0.00
0
100
200
300
400
500
600
700
800
Round Robin
OpenStack
Resouce balanced max


Balanced resource usage (Scenario 3)
Balanced resource usage
VM count
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
100
200
300
400
500
600
700
Round Robin
OpenStack
Resouce balanced max

По полученным данным видно, что эффективность того или иного алгоритма проявляется при большом количестве серверов и виртуальных машин. При данных метриках предложенный алгоритм показывает немного лучшие результаты по сравнению с двумя другими, при этом эффективность алгоритмов также зависит от модели потребления.

Как результат создание более эффективных алгоритмов может быть основано на предсказании моделей потребления и выборе наиболее эффективного алгоритма для данной модели.

6. Выводы

Была поставлена задача как DRS в целом, так и задача изначального размещения в частности.

Было проведено исследование существующих алгоритмов размещения для облачных систем. Рассмотрены различные подходы к решению этой задачи.

Был предложен алгоритм, улучшающий работу планировщика OpenStack. Данный алгоритм был реализован в рамках существующего проекта Nova OpenStack и протестирован на экспериментальном стенде. Также был проведен сравнительный анализ ряда существующих алгоритмов с предложенным. При помощи симуляции была показана эффективность предложенного алгоритма в большинстве сценариев.

Список источников

1. *Peter Mell, Timothy Grance*, The NIST Definition of Cloud Computing
2. *Minxian Xu, Wenhong Tian, Rajkumar Buyya*, A Survey on Load Balancing Algorithms for Virtual Machines Placement in Cloud Computing
3. *Nguyen Trung Hieu, Mario Di Francesco, Antti Yla-Jaaski*, A Virtual Machine Placement Algorithm for Balanced Resource Utilization in Cloud Data Centers
4. *Tracy D. Braun, Howard Jay Siegel, Noah Beck*, A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems
5. *Thiago Cordeiro, Douglas Damalio, Nadilma Pereira, Patricia Endo, André Palhares, Glauco Gonçalves, Djamel Sadok, Judith Kelner*, Open Source Cloud Computing Platforms
6. *vSphere Resource Management*, VMware, Inc https://pubs.vmware.com/vsphere-60/topic/com.vmware.ICbase/PDF/vsphere-esxi-vcenter-server-601-resource-management-guide.pdf
7. *Martin Sivák*, Optaplanner integration with scheduling https://www.ovirt.org/develop/release-management/features/sla/optaplanner/
8. OpenStack Installation Tutorial for Ubuntu https://docs.openstack.org/ocata/install-guide-ubuntu/
9. *Roy Thomas Fielding*, Architectural Styles and the Design of Network-based Software Architectures
10. *Sushil Kumar*, Various Dynamic Load Balancing Algorithms in Cloud Environment: A Survey

11. *Ефанов Н.Н*, Динамическое управление питанием в задаче динамического управления ресурсами: вариации планировщика и эксперименты.
12. *Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, Andrew Warfield*, Live Migration of Virtual Machines
13. *Ching-Chi Lin, Pangfeng Liu, Jan-Jan Wu*, Energy-efficient Virtual Machine Provision Algorithms for Cloud Systems