СОЗДАНИЕ ЭФФЕКТИВНЫХ АЛГОРИТМОВ ФУНКЦИОНИРОВАНИЯ ОБЛАЧНЫХ СИСТЕМ

Полежаев П.Н.
Оренбургский государственный университет, г. Оренбург

В рамках настоящего исследования рассматриваются проблемы эффективного назначения облачных сервисов (приложений) на виртуальные машины и маршрутизации потоков данных между ними с использованием программно-конфигурируемых сетей.

Каждый программный сервис (приложение) облачной системы может быть представлен в виде динамического взвешенного ориентированного графа:

$$G(t) = (V, E, d(e,t), w(v)),$$

где V – множество вершин, представляющих собой компоненты сервиса; E – максимальное полное множество дуг (сетевых соединений), допустимых между вершинами V; $d(e,t)$ – функция, определяющая количество передаваемых данных по дуге $e \in E$ в момент времени $t \geq 0$; $w(v)$ – характеристики вершины $v \in V$.

Множество компонентов представляет собой следующее объединение:

$$V = P \cup Q \cup D \cup S,$$

где P – набор запущенных программ (процессов), Q – множество используемых очередей, D – множество хранилищ данных, S – множество точек взаимодействия со стандартными сервисами облачной системы.

Задача назначения облачного приложения на виртуальные машины может быть формализована следующим образом.

Будем считать, что каждое облачное приложение $G(t)$ имеет микросервисную архитектуру, основанную на контейнерах [1]. Микросервис $v \in V \setminus S$ запускается в отдельном контейнере $c(v)$. Вершины $v \in S$ не участвуют в задаче назначения, т.к. по сути они являются точками подключения (привязки) к другим облачным приложениям или к стандартным сервисам облачной системы. Однако для подобных вершин важна коммуникационная составляющая.

Пронумеруем все вершины, содержащие микросервисы, следующим образом:

$$V \setminus S = \{v_1, v_2, \ldots, v_{n(G)}\},$$

где $n(G) = |V \setminus S|$ – количество микросервисов.

Пусть $VMs(t)$ – существующие в ЦОД виртуальные машины на момент времени t, которые готовы принимать для назначения контейнеры, $VMs^*(t)$ – максимальные по конфигурации виртуальные машины, которые могут быть запущены на оставшихся ресурсах физических серверов на момент времени t (по одной на каждый сервер).

Каждый контейнер $c(v)$ с микросервисом облачного приложения должен быть развернут внутри одной из запущенных виртуальных машин $VM \in VMs(t)$ облачного ЦОД или в новой виртуальной машине $VM^* \in VMs^*(t)$, которая будет запущена на одном из физических серверов $Server^* \in Servers$ (используя все свободные ресурсы).

С целью формализации соответствия микросервис-контейнер-виртуальная машина введем следующий вектор:

$$\varphi_G = (VM_1, VM_2, VM_3, ..., VM_{n(G)}),$$

где $\varphi_G(v_i) = VM_i \in VMs(t) \cup VMs^*(t)$ – виртуальная машина, на которую назначен контейнер $c(v_i)$, содержащий микросервис v_i.

Для оценки эффективности φ_G может быть построена следующая функция, значение которой должно быть оптимизировано:

$$I(\varphi_G) = \alpha I_{comp}(\varphi_G) + (1-\alpha) I_{comm}(\varphi_G) \to \max,$$

где $\alpha \in [0,1]$ – весовой коэффициент, $I_{comp}(\varphi_G)$ – оценка эффективности использования вычислительных ресурсов, на которые назначены микросервисы приложения G, $I_{comm}(\varphi_G)$ – оценка использования сети в результате назначения всех дуг $e \in E$ на маршруты передачи данных в облачном ЦОД.

Должны выполняться следующие ограничения:

а) ресурсные требования v_i соответствуют свободным ресурсам VM_i, т.е.

$$m_{v_i} \le RAM_{VM_i} - m_{VM_i}(t),$$

$$d_{v_i} \le HDD_{VM_i} - d_{VM_i}(t),$$

$$u_{v_i} \le Cores_{VM_i} - u_{VM_i}(t),$$

где m_{v_i} и d_{v_i} – соответственно запрашиваемые размеры оперативной и дисковой памяти, необходимой для функционирования микросервиса v_i; u_{v_i} – доля запрашиваемых микросервисом вычислительных ядер; RAM_{VM_i} и HDD_{VM_i} – соответственно размеры оперативной и дисковой памяти виртуальной машины VM_i; $Cores_{VM_i}$ – количество вычислительных ядер у VM_i, $m_{VM_i}(t)$ и $d_{VM_i}(t)$ – соответственно размеры занятой оперативной и дисковой памяти внутри виртуальной машины VM_i в текущий момент времени t, $u_{VM_i}(t)$ – доля использования вычислительных ядер виртуальной машины VM_i в момент времени t.

б) также должны выполняться сетевые ограничения.

В качестве $I_{comp}(\varphi_G)$ предлагается использовать функцию следующего вида:

$$I_{comp}(\varphi_G) = \frac{1}{n(G)} \sum_{i=1}^{n(G)_i} \overline{U}_{VM.i},$$

где $\overline{U}_{VM.i}$ – оценка средней загруженности i-й виртуальной машины.

Задачу оптимизации функции $I(\varphi_G)$ будем решать с помощью генетического алгоритма, вычисляющего оптимальное назначение φ_G и вложенного алгоритма маршрутизации потоков данных, который прокладывает маршруты передачи данных между вершинами облачного приложения G с учетом φ_G. Для оценки эффективности проложенных маршрутов используется величина $I_{comm}(\varphi_G)$.

Хромосома в генетическом алгоритме будет представлять собой вектор φ_G, а его компоненты будут являться генами. Опишем основные генетические операторы (см. алгоритм 1): операция скрещивания – обычное одноточечное скрещивание для двух векторов; мутация – случайная замена одной виртуальной машины на другую, удовлетворяющую ресурсным требованиям соответствующего микросервиса, селекция – элитный отбор в сочетании с рулеткой. Критерии окончания работы алгоритма – превышение предельного времени T_{max} и отсутствие улучшений в среднем значении оптимизируемой функции на протяжении нескольких поколений.

Алгоритм 1 – Генетический алгоритм назначения облачных приложений

Шаг 1. Замерить текущий момент времени T_{start}.

Шаг 2. Создать $Population_1$ начальную популяцию размера N путем случайного выбора назначаемых виртуальных машин для микросервисов.

Шаг 3. Положить в качестве номера итерации значение $i := 1$.

Шаг 4. Пока $T_{current} - T_{start} < T_{max}$ и имеются улучшения в среднем значения функции $I(\varphi_G)$ на протяжении K поколений необходимо выполнить следующие шаги:

Шаг 4.1. Над хромосомами $Population_i$ с вероятностью P выполнить операции скрещивания, объединив родительские хромосомы в случайные пары. Получаемые дочерние хромосомы сохранить в $Population'_i$.

Шаг 4.2. Для хромосом $Population'_i$ выполнить операцию мутации с вероятностью Q.

Шаг 4.3. Объединить родительские и дочерние популяции $Population''_i := Population_i \cup Population'_i$

Шаг 4.4. Для каждой хромосомы $\varphi_G \in Population''_i$ запустить алгоритм маршрутизации с целью проактивного вычисления наилучших маршрутов передачи данных и получения оценки $I_{comm}(\varphi_G)$.

Шаг 4.5. Для каждой хромосомы $\varphi_G \in Population''_i$ вычислить оценку $I_{comp}(\varphi_G)$.

Шаг 4.5. Для $Population''_i$ выполнить операцию селекции, используя в качестве функции соответствия функцию $I(\varphi_G)$, выбранные хромосомы сохранить в $Population_{i+1}$.

Шаг 4.6. Увеличить на единицу номер итерации $i := i + 1$ и перейти к шагу 4.

Шаг 5. Выполнить назначение в соответствии с лучшим значением φ_G^{max} последнего поколения, при необходимости запустить новые виртуальные машины и проложить вычисленные маршруты передачи данных для φ_G^{max}.

Формализуем задачу реактивной и проактивной маршрутизации потоков данных между серверами и виртуальными машинами, исполняющими облачные сервисы.

Пусть $G_T(t) = (V_T, E_T)$ – ориентированный мультиграф, описывающий текущую топологию сети в некоторый момент времени t. Множество его вершин V_T является объединением множества узлов (серверов, виртуальных машин, контейнеров) и других сетевых устройств (коммутаторов, шлюзов, СХД и т.п.).

Каждая дуга $e_T \in E_T$ соответствует некоторой сетевой связи между вершинами $beg(e_T) \in V_T$ и $end(e_T) \in V$. У нее также есть противоположная дуга, т.к. связь дуплексная. Между двумя вершинами может быть несколько параллельных дуг, например, параллельные соединения между маршрутизаторами.

На множестве дуг E_T заданы две функции:

а) $b : E_T \to R^+ \cup \{0\}$ – отображение, характеризующее текущую пропускную способность каждой дуги в момент времени t.

б) $s : E_T \to R^+ \cup \{0\}$ – задержка на соответствующем выходном порту дуги в момент времени t.

Пусть для каждой дуги $e \in E$ облачного приложения $G(t)$ заданы значения:

а) $\underline{b}(e) = \max\limits_t d(e,t)$ – минимальная гарантированная пропускная способность потоков данных, соответствующих данной дуге.

б) $\bar{s}(e) = s = const$ – максимальная гарантированная задержка (задается в конфигурации облачного приложения).

в) $\hat{s}(e)$ – оценка средней задержки, которая возникнет при обработке пакетов потоков данных на портах сетевых устройств.

Для вычисления $I_{comm}(\varphi_G)$ может быть использована функция:

$$I_{comm}(\varphi_G) = \max_R I_{comm}(\varphi_G, R) =$$

$$\sum_{\substack{e_i \in E: \\ \min\limits_{e_T \in r_i}\{b(e_T) - \sum\limits_{c \in \psi(e_T)\setminus\{e_i\}} \underline{b}(c)\} \geq \underline{b}(e_i) \& \\ \sum\limits_{e_T \in r_i}(s(e_T) + \sum\limits_{c \in \psi(e_T)\setminus\{e_i\}} \hat{s}(c)) \leq \bar{s}(e_i)}} \left[\alpha_b (\min_{e_T \in r_i}\{b(e_T) - \sum_{c \in \psi(e_T)/\{e_i\}} \underline{b}(c)\} - \underline{b}(e_i)) + \alpha_s (\bar{s}(e_i) - \sum_{e_T \in r_i} s(e_T) - \sum_{c \in \psi(e_T)\setminus\{e_i\}} \hat{s}(c)) \right] +$$

$$+ \beta_b \cdot \sum_{\substack{e_i \in E: \\ \min\limits_{e_T \in r_i}\{b(e_T) - \sum\limits_{c \in \psi(e_T)/\{c_i\}} \underline{b}(c)\} < \underline{b}(e_i)}} \left[\min_{e_T \in r_i}\{b(e_T) - \sum_{c \in \psi(e_T)\setminus\{e_i\}} \underline{b}(c)\} - \underline{b}(e_i) \right] +$$

$$+ \beta_s \cdot \sum_{\substack{e_i \in E: \\ \sum\limits_{e_T \in r_i}(s(e_T) + \sum\limits_{c \in \psi(e_T)\setminus\{e_i\}} \hat{s}(c)) > \bar{s}(e_i)}} \left| \bar{s}(e_i) - \sum_{e_T \in r_i} s(e_T) - \sum_{c \in \psi(e_T)\setminus\{e_i\}} \hat{s}(c) \right|$$

где $R = (r_1, ..., r_{|E|})$ – вектор, задающий маршруты передачи потоков данных, причем $r_i = \delta(e_i)$ – соответствие между коммуникационной связи $e_i \in E$ облачного приложения $G(t)$ и маршрутом r_i, вдоль которого планируется передача данных; $\psi(e_T) = \{e \in E \mid r = \delta(e) \,\&\, e_T \in r\}$ – множество коммуникационных связей облачного приложения, маршруты которых проходят через e_T, $\alpha_b > 0$ и $\alpha_s > 0$ – поощрения за соблюдение соответствующих ограничений по пропускной способности и задержкам, $\beta_b < 0$ и $\beta_s < 0$ – штрафы за их несоблюдение. Функция $I_{comm}(\varphi_G, R)$ отражает поощрения за соблюдение гибких ограничений по пропускной способности и задержкам, устанавливает штрафы за их нарушения.

Также имеются жесткие ограничения на маршруты $r_i = (e'_{i1}, ..., e'_{in_i})$:

1. r_i действительно является маршрутом:

$$\forall r_i \;\; \forall j = \overline{1, n_i - 1} \;\; end(e'_{ij}) = beg(e'_{ij+1}).$$

2. r_i должен начинаться в вершине, соответствующей началу дуги e_i и заканчиваться в вершине, соответствующей концу e_i, т.е.:

$$\forall r_i \;\; beg(e_{i1}) = beg(e_i) \,\&\, end(e_{in_i}) = end(e_i).$$

3. r_i не должен проходить несколько раз через одну и ту же вершину, т.е.:

$$\forall r_i \;\; \forall j, k = \overline{1, n_i} \;\; j \neq k \Rightarrow beg(e_{ij}) \neq beg(e_{ik}).$$

Для оптимизации функции $I_{comm}(\varphi_G, R)$ по R в рамках данной работы предлагается использовать генетический алгоритм (см. алгоритм 2). Хромосомой будет являться вектор R. Алгоритм использует генетические операторы, аналогичные алгоритму 1: одноточечное скрещивание, элитный отбор в сочетании с рулеткой. Операция мутации заключается в замене одного проложенного маршрута передачи данных другим альтернативным.

Алгоритм 2 – Генетический алгоритм проактивной маршрутизации потоков данных

Шаг 1. Замерить текущий момент времени T_{start}.

Шаг 2. Создать $Population_1$ начальную популяцию размера M. В качестве одной из хромосом R выбрать маршруты, проложенные с помощью алгоритма Дейкстры, запущенного из каждой вершины V, руководствуясь минимизацией суммарных ограничений по задержкам. Остальные хромосомы сгенерировать случайным образом.

Шаг 3. Положить в качестве номера итерации значение $i := 1$.

Шаг 4. Пока $T_{current} - T_{start} < T_{max}$ и имеются улучшения в среднем значения функции $I_{comm}(\varphi_G, R)$ на протяжении K поколений необходимо выполнить следующие шаги:

Шаг 4.1. Над хромосомами $Population_i$ с вероятностью P' выполнить операции скрещивания, объединив родительские хромосомы в случайные пары. Получаемые дочерние хромосомы сохранить в $Population'_i$.

Шаг 4.2. Для хромосом $Population'_i$ выполнить операцию мутации с вероятностью Q'.

Шаг 4.3. Объединить родительские и дочерние популяции $Population''_i := Population_i \cup Population'_i$

Шаг 4.4. Для $Population''_i$ выполнить операцию селекции, выбранные хромосомы сохранить в $Population_{i+1}$.

Шаг 4.5. Увеличить на единицу номер итерации $i := i + 1$ и перейти к шагу 4.

Шаг 5. Вернуть все маршруты из лучшей хромосомы $Population_i$.

Для реактивной маршрутизации, когда маршрут передачи потока данных в момент его появления в сети, разработан алгоритм 4.

Алгоритм 4 – Реактивная маршрутизация потоков данных

Шаг 1. Прочитать вершины отправителя $u \in V_T$ и получателя $u' \in V_T$ потока данных.

Шаг 2. Используя текущий граф топологии $G_T(t)$ с помощью алгоритма Дейкстры рассчитать маршрут передачи данных от u до u', используя в качестве весов остаточные пропускные способности дуг.

Шаг 3. С помощью правил OpenFlow [2] установить вычисленный маршрут в коммутаторы.

Для экспериментального оценивания разработанных алгоритмических решений был создан симулятор облачной системы и программно-конфигурируемой сети. В его основе лежит:

а) Симуляция работы облачной системы – системы управления, работы виртуальных машин, дисковых образов, шаблонов, хранилищ данных,

контейнеров, серверов и т.п. Симуляция позволила на данном этапе НИР сосредоточиться на разработке алгоритмов, а не на реализации технических задач.

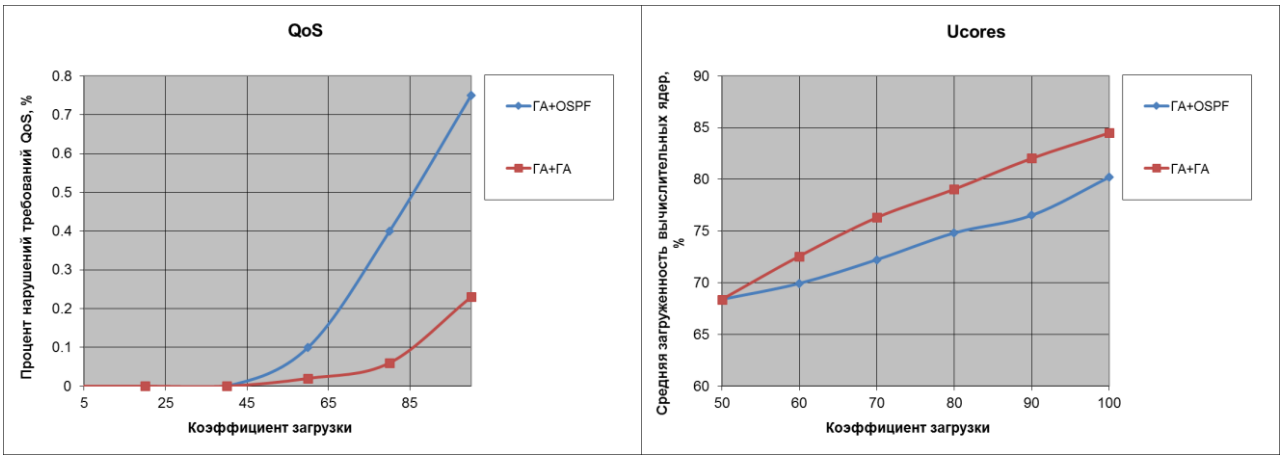
б) Эмуляция работы программно-конфигурируемой сети за счет использования системы Mininet [3], поддерживающей протокол OpenFlow. Использование эмулятора позволило обеспечить детальное моделирование компьютерной сети. В качестве контроллера для программно-конфигурируемой сети был выбран Ryu.

В процессах узлов, имитирующих компоненты облачных приложений, запускаются генераторы трафика с заданными параметрами, которые берутся из компоненты симуляции облачной системы.

Генетические алгоритмы назначения облачных приложений и проактивной маршрутизации потоков данных исследовались совместно, рассматривались сочетания генетического алгоритма назначения с генетическим алгоритмом проактивной маршрутизацией (ГА+ГА) и с проактивным алгоритмом OSPF (ГА+OSPF).

С этой целью генерируется пуассоновский поток заявок на запуск облачных приложений $\sigma_G = \{G_i(t)\}$, интервалы времени между появлением заявок распределены экспоненциально с интенсивностью λ_G. Параметры заявок генерировались с помощью подобранных законов распределения согласно имитационной модели. В качестве физической конфигурации системы была использована топология, ранее нами использованная в работе [4].

На рисунке 1 представлены графики зависимости процента нарушений требований QoS (а) и средней загруженности вычислительных ядер облачной системы (б) от коэффициента загрузки, вычисляемого на основе варьируемой интенсивности λ_G. Каждое значение, отмеченное графике, представляет собой усредненное значение соответствующей метрики на 50 случайно сгенерированных потоках заявок на запуск облачных приложений σ_G.



а) б)

Рисунок 1 – Графики зависимости метрик эффективности от коэффициента загрузки для алгоритмов ГА+ГА и ГА+OSPF: а) процент нарушения требований QoS; б) средняя загруженность вычислительных ядер

Анализ рисунка 1 а) показывает, что сочетания ГА+ГА и ГА+OSPF демонстрирует нулевой процент нарушений требований QoS до тех пор, пока не будет достигнуто значение коэффициента загрузки ~40%. После данного порога сочетание ГА+ГА показывает гораздо лучшие значения данной метрики по сравнению с ГА+OSPF. Рисунок 2 б) демонстрирует преобладание алгоритма ГА+ГА по метрике средней загруженности вычислительных ядер над алгоритмом ГА+OSPF. Улучшения в абсолютном выражении составляют до 6%.

Разработанный реактивный алгоритм маршрутизации потоков данных (РА) исследовался совместно со стандартным реактивным OSPF. Эксперимент проводился на тех же потоках заявок, что и в первом эксперименте. Результаты представлены на рисунке 2.

Анализ графиков показывает рост процента нарушений требований QoS из-за реактивного характера маршрутизации трафика, а также небольшое преимущество предложенного алгоритма реактивной маршрутизации перед OSPF.

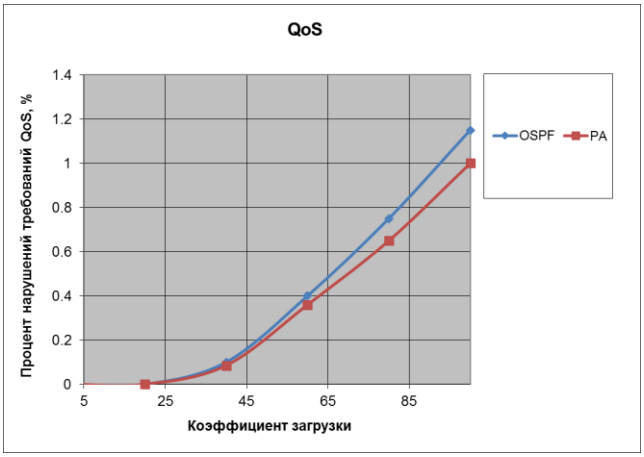


Рисунок 2 – Графики зависимости процента нарушений требований QoS от коэффициента загрузки для реактивных алгоритмов маршрутизации

Исследования проведены при финансовой поддержке РФФИ и Правительства Оренбургской области (проект № 16-47-560335), Президента Российской Федерации, стипендии для молодых ученых и аспирантов (СП-2179.2015.5).

Список литературы

1. Адрова Л.С. Сравнительный анализ существующих технологий контейнеризации / Л.С. Адрова, П.Н. Полежаев // Университетский комплекс как региональный центр образования, науки и культуры [Электронный ресурс]: материалы Всероссийской научно-методической конференции; Оренбург. гос. ун-т. - Электрон. дан. - Оренбург: ОГУ, 2016. – Загл. с этикетки диска. – С. 2473-2477.

2. OpenFlow - Open Networking Foundation [Электронный ресурс] // Open

Networking Foundation. – Электрон. дан. – 2016. Режим доступа: https://www.opennetworking.org/sdn-resources/openflow. Загл. с экрана. - (Дата обращения: 25.11.2016).

3. Introduction to Mininet [Электронный ресурс] // Mininet Project. – Электрон. дан. – 2016. Режим доступа: https://github.com/mininet/mininet/wiki/Introduction-to-Mininet. Загл. с экрана. – (Дата обращения: 25.11.2016).

4. Polezhaev P. Network Resource Control System for HPC based on SDN / P. Polezhaev, A. Shukhman, Yu. Ushakov // Proceedings of 14th International Conference, NEW2AN 2014 and 7th Conference ruSMART 2014, St. Petersburg, Russia. Lecture Notes in Computer Science. – vol. 8638. – PP. 219-230.