РЕШЕНИЕ ЗАДАЧ ПЛАНИРОВАНИЯ ВИРТУАЛЬНЫХ МАШИН И СБОРА СТАТИСТИКИ О РАБОТЕ ОБЛАЧНЫХ ПРИЛОЖЕНИЙ

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

В настоящее время облачные вычисления получили широкое распространение во всех областях жизни человека. Сейчас публичные и приватные облачные системы используются различными организациями для размещения собственных ИТ-инфраструктур, бизнес-приложений, коммуникационных сервисов.

При использовании приватных облачных систем компании получают возможность консолидировать вычислительные и сетевые ресурсы собственного центра обработки данных (ЦОД) и эффективно управлять ими. Достоинства – гибкость и простота развертывания облачных сервисов и приложений, высокая надежность работы, эффективное использование ресурсов за счет виртуализации.

В случае публичных облачных систем компании получают дополнительные преимущества. В первую очередь – это экономия финансовых средств за счет отсутствия необходимости содержать собственный парк серверов, их обслуживать, амортизировать. Также отсутствует необходимость нанимать высокооплачиваемых специалистов для их настройки и эксплуатации. Использование публичных облачных систем позволяет оплачивать фактически используемые облачные ресурсы (время работы виртуальных машин, сетевой трафик, фактически хранящиеся данные и т.п.), это дает значительную гибкость в управлении ИТ-инфраструктурой. В частности, можно масштабировать работу собственных сервисов и приложений в зависимости от ежедневного, еженедельного, ежемесячного или годового цикла работы с ними сотрудников и клиентов компании, увеличивать их масштаб в случае прогнозируемых или непредвиденных пиков работы с ними. Как правило заказ сервисов и их масштабирование осуществляется через портал самообслуживания публичного облачного провайдера, либо это может осуществляться автоматически через предоставляемый API.

Если рассматривать сторону владельца облачной системы – провайдера (в случае публичного облака) или самой компании (в случае приватного облака), то очень важно эффективно использовать имеющиеся ресурсы – вычислительные ресурсы и память серверов, программные лицензии, каналы связи.

В рамках данной работы рассматриваются проблемы эффективного планирования виртуальных машин для облачных систем, а также построения коммуникационных схем сервисов на основе статистики их взаимодействия.

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 $v \in V$.

Если $d(e,t) = 0$, то дуга e отсутствует в момент времени t.

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 (например, сервис СУБД, Hadoop, Machine Learning, входными шлюзами/балансировщиками нагрузки (для запросов из Интернет)).

Построения коммуникационных схем сервисов на основе статистики их взаимодействия заключается в выявлении вершин V и в задании функции $d(e,t)$. Сами вершины выявлять, как правило, не требуется, т.к. у программных сервисов имеется конфигурационный файл, который их описывает. Это может быть конфигурационный файл облачной системы или файл с описанием контейнеров (при использовании микросервисной контейнерной архитектуры), например, docker-compose.yml для Docker.

Для построения функции $d(e,t)$ может быть использовано два подхода:

а) сбор информации о коммуникациях с помощью счетчиков OpenFlow [1–3];

б) получение сведений о коммуникациях с помощью перехвата всех коммуникационных вызовов в сетевом API (библиотека передачи сообщений облачной системы, сетевые сокеты и т.п.).

Подход а) работает в программно-конфигурируемых сетях и является более универсальным по сравнению с б), т.к. последний предполагает реализацию для всех возможных коммуникационных библиотек промежуточного интерфейса API, содержащего вызовы, которые оборачивают оригинальные вызовы и дополнительно сохраняют информацию о передаваемых потоках данных.

Протокол OpenFlow поддерживает счетчик “Received Bytes” категории “Per Flow Entry” (количество байт, переданных данным потоком), который используется разработанным алгоритмом построения коммуникационных схем сервисов на основе статистики их взаимодействия.

Для реализации первого подхода контроллер должен регулярно запрашивать информацию по счетчикам у всех коммутаторов OpenFlow (см. алгоритм 1). В данном алгоритме используется обозначение $Rb(l,t)$ – количество байт переданных по маршруту l с начала его существования до момента времени t. Данная величина вычисляется на основе значениях

счетчиков производительности $Rb(Switch_j, t)$ во всех коммутаторов OpenFlow $Switch_j$, через которые проходит маршрут l, согласно формуле:

$$Rb(l,t) = \frac{1}{k(l)} \sum_{Swich_j \in l} Rb(Switch_j, t),$$

где $k(l)$ – количество коммутаторов OpenFlow в маршруте l.

Фактически $Rb(l,t)$ вычисляется, как среднее арифметическое значение счетчиков со всех коммутаторов вдоль маршрута. Это необходимо для решения проблемы временного рассогласования получаемых значений счетчиков с разных коммутаторов OpenFlow и с наличием пакетов потоков, перемещающихся вдоль маршрута.

Алгоритм 1 – Алгоритм построения коммуникационных схем сервисов на основе статистики их взаимодействия с использованием счетчиков OpenFlow

Шаг 1. Получить граф топологии облачной системы, включая физические узлы, виртуальные машины и контейнеры.

Шаг 2. По таймеру раз в ΔT_{stat} секунд выполнять следующие действия:

Шаг 2.1. Получить список запущенных облачных приложений $Apps(t) = \{G_i(t)\}_i$ облачного ЦОД.

Шаг 2.2. Получить список текущих маршрутов L_{Routes} передачи данных, проложенных алгоритмом маршрутизации. Для каждого нового маршрута $l \in L_{Routes}$ положить в t^l_{prev} текущее время.

Шаг 2.1. Для каждого маршрута $l \in L_{Routes}$:

Шаг 2.1.1. Определить приложение $G_i(t)$ и дугу e, к которым относится данный маршрут l.

Шаг 2.1.2. Взять текущее время t^l_{cur}.

Шаг 2.1.3. Присвоить $d(e,t) := \frac{Rb(l, t^l_{cur}) - Rb(l, t^l_{prev})}{t^l_{cur} - t^l_{prev}}$ для всех $t \in [t^l_{prev}, t^l_{cur}]$.

Шаг 2.1.4. $t^l_{prev} := t^l_{cur}$.

Алгоритм 1 реализуется в виде модуля для контроллера OpenFlow.

Формализуем задачу планирования виртуальных машин с учетом топологии системы и коммуникационных схем сервисов, запускаемых внутри виртуальных машин.

Возможны два случая:

а) Запуск группы виртуальных машин для назначения компонентов запускаемого облачного сервиса (приложения).

б) Запуск одиночной виртуальной машины для размещения компоненты (микросервиса) облачного приложения. Потребность в решении подобной задачи возникает при необходимости увеличить количество экземпляров

микросервиса, например, при его ручном или автоматическом масштабировании.

Рассмотрим первый случай. Коммуникационная схема облачного приложения $G(t)$ может быть задана явно пользователем или получена с помощью алгоритма построения коммуникационных схем сервисов на основе статистики их взаимодействия. В случае отсутствия информации о схеме и невозможности ее получить в ситуации, когда облачное приложение запускается первый раз в облачной системе, будем считать, что коммуникационный паттерн $G(t)$ соответствует полному графу, причем $d(e,t) = \overline{B} = \text{const}$ – т.е. для любой дуги в любой момент времени количество передаваемых данных равно средней скорости передачи данных $\overline{B}$, вычисленной по всем ранее запущенным в облачной системе приложениям.

Решение данной задачи сводится к задаче назначения облачного сервиса на новые виртуальные машины $VMs^*(t)$, которые могут быть запущены с использованием остаточных ресурсов физических серверов. Формируется список подобных потенциальных виртуальных машин, затем осуществляется на них назначение с помощью алгоритма, описанного в статье Полежаева П.Н. «Создание эффективных алгоритмов функционирования облачных систем», опубликованной в материалах данной конференции.

Второй случай. Пусть v' – новый экземпляр микросервиса. Обозначим новый граф облачного приложения в виде:

$$G'(t) = (V', E').$$

Здесь $V' = V \cup \{v'\}$, $E' = E \cup \bigcup\limits_{u \in V} \{(u, v')\} \cup \bigcup\limits_{u \in V} \{(v', u)\}$.

Пусть $Type(v)$ – тип вершины v (процесс, очередь, хранилище). Обозначим в качестве $M(v, G) = \{u \in V \mid Type(v) = Type(u)\}$.

Для добавленных дуг задаются значения весовой функции d по формулам:

$$d((u, v'), t) = \frac{1}{|M(v', G)|} \sum_{v \in M(v', G)} d((u, v), t),$$

$$d((v', u), t) = \frac{1}{|M(v', G)|} \sum_{v \in M(v', G)} d((v, u), t).$$

Данные формулы позволяют новым дугам, соединяющим старую вершину $u \in V$ и новую вершину v', приписать для каждого момента времени t средние значения скорости передачи данных, вычисленные по аналогичным дугам, соединяющим вершину $u \in V$ и старые вершины того же типа. В процессе работы данные веса могут уточняться за счет детального сбора статистики.

Пусть $\varphi_{G,VM} = \text{Append}(\varphi_G, VM) = (VM_1, ..., VM_{n(G)}, VM)$, где φ_G – определяет построенное назначение компонентов-микросервисов облачного приложения на виртуальные машин.

Опишем разработанный жадный алгоритм выбора виртуальной машины для запуска нового экземпляра микросервиса v' (см. алгоритм2).

Алгоритм 2 – Алгоритм выбора виртуальной машины для запуска нового экземпляра микросервиса

Шаг 1. По облачному приложению $G(t)$ построить $G'(t)$, уточнить значения весовой функции d.

Шаг 2. Сформировать список виртуальных машин $VMs'(t)$, существующих в ЦОД на момент времени t и удовлетворяющих ресурсным требованиям v'.

Шаг 3. Сформировать список новых виртуальных машин $VMs'^{*}(t)$, которые построены на основе свободных на момент времени t ресурсов серверов и удовлетворяют ресурсным требованиям v'.

Шаг 4. Используя алгоритм проактивной маршрутизации вычислить:

$$VM_{opt} = \arg \max_{VM \in VMs'(t) \cup VMs'^{*}(t)} \{\alpha I_{comp}(\varphi_{G,VM}) + (1-\alpha) I_{comm}(\varphi_{G,VM})\},$$

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

Шаг 5. Если $VM_{opt} \in VMs'^{*}(t)$, то запустить виртуальную машину VM_{opt}.

Шаг 6. Присвоить $\varphi_{G'} := \varphi_{G,VM_{opt}}$. Назначить v' на VM_{opt}.

Шаг 7. Проложить маршруты передачи данных, соответствующие VM_{opt}.

С целью экспериментального оценивания разработанных алгоритмов был создан симулятор облачной системы и программно-конфигурируемой сети. В его основе лежит разработанная имитационная модель [4].

Экспериментальное исследование алгоритма построения коммуникационных схем сервисов на основе статистики их взаимодействие запланировано на следующий этап исследования, когда будет использоваться реальный облачный ЦОД. На данном этапе с помощью симулятора подтверждена его работоспособность и корректность работы.

Исследование предложенного алгоритма планирования виртуальных машин с учетом топологии системы и коммуникационных схем сервисов проводилось в сравнении со стандартным алгоритмом планирования match-making системы OpenNebula [5], основанном на фильтрации и ранжировании серверов. Следует заметить, что он не учитывает топологию системы и коммуникационные схемы сервисов. Это сказалось на полученных результатах (см. рисунок 1).

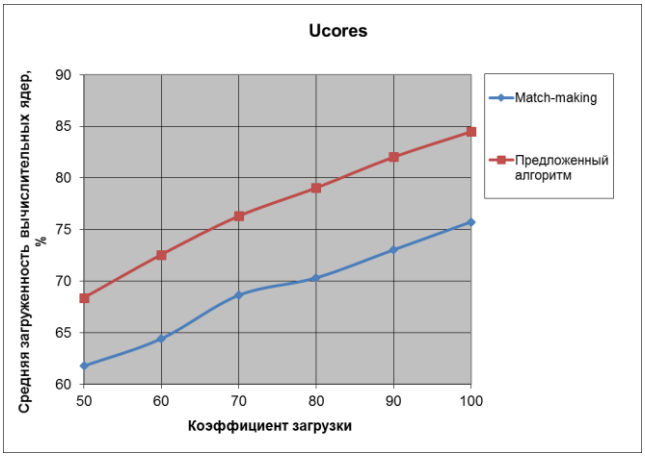

Рисунок 1 – Графики зависимости загруженности вычислительных ядер облачной системы от коэффициента загрузки для алгоритмов планирования виртуальных машин

Для предложенного алгоритма планирования в качестве алгоритма назначения использовался разработанный генетический алгоритм в сочетании с созданным алгоритмом проактивной маршрутизации, а для алгоритма match-making применялся принцип назначения Best Fit в сочетании с реактивным алгоритмом маршрутизации OSPF – характерное сочетания для обычных облачных систем.

Рисунок 1 показывает преимущество предложенного алгоритма в сравнении со стандартными решения на 6-9 % в зависимости от коэффициента загрузки.

В рамках данной статьи были описаны созданные решения – алгоритмы планирования виртуальных машин с учетом топологии системы и коммуникационных схем сервисов, запускаемых внутри виртуальных машин, а также алгоритм построения коммуникационных схем сервисов на основе статистики их взаимодействия. Оба алгоритма были исследованы с помощью симулятора облачной системы и программно-конфигурируемой сети.

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. *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).*

2. *Полежаев П.Н. Изучение возможностей контроллера Ryu и оценка эффективности программно-конфигурируемой сети / П.Н. Полежаев, В.И. Чернов, С.Ю. Шиховцов // Прикладные информационные системы: третья Всероссийская НПК (г. Ульяновск, 30 мая – 12 июня 2016 г.): сборник научных трудов / под ред. Е. Н. Эгова. – Ульяновск : УлГТУ, 2016. – С. 151-158.*

3. Чернов В.И. Сравнительный анализ существующих контроллеров для программно-конфигурируемых сетей / В.И. Чернов, С.Ю. Шиховцов, 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. – С. 2524-2529.

4. Легашев Л.В. Имитационная модель облачного ресурсного центра / Л.В. Легашев, И.П. Болодурина, П.Н. Полежаев // Интеллект. Инновации. Инвестиции. – 2016. – №2. – С. 113-116.

5. Scheduler – OpenNebula 5.0.2 documentation [Электронный ресурс] // OpenNebula Project (OpenNebula.org). – Электрон. дан. – 2016. Режим доступа:http://docs.opennebula.org/5.0/operation/host_cluster_management/scheduler.html. Загл. с экрана. - (Дата обращения: 25.11.2016).