Санкт-Петербургский государственный университет
Математико-механический факультет
Кафедра системного программирования

Адаптивное распределение ресурсов в облачных системах

Курсовая работа студентки 344-ой группы
Михайловой Александры Алексеевны

Научный руководитель . . . . . Белогрудов В.М.

Санкт-Петербург
2013

Оглавление

Введение

В настоящее время всё большую популярность приобретает облачная архитектура с идеей абстракции логики приложения от физических ресурсов. На машинах-”хостах” запускаются виртуальные машины (далее VM) с клиентскими службами, которые можно разделить на два больших класса: приложения для пакетной обработки данных и интерактивные сервисы. Для последних особенно важны производительность и время отклика, поскольку от этого напрямую зависит выполнение SLA[1]. Поэтому необходима организация постоянного мониторинга загрузки физических серверов в системе с облачной архитектурой, определение пиков нагрузки и их оперативная ликвидация, так как повышение нагрузки на машину опасно поломкой машины и, следовательно, остановкой запущенных пользовательских приложений и нарушением SLA. Такая ситуация недопустима. Под загрузкой сервера здесь понимается количество физических ресурсов (процессорное время, оперативная память, сеть и др.), потребляемое VM, работающими в данный момент на этом сервере.

Острой становится проблема обработки возникающих пиков нагрузки. Обычной практикой в таком случае является миграция одной или нескольких VM с перегруженного сервера на более “свободный”, что означает остановку работы приложения на первом сервере, передачу всех нужных данных с первого сервера на второй и повторный запуск приложения уже на второй машине. Осуществление миграции требует дополнительных физических ресурсов, которых в какой-то момент может просто не быть. Однако в случае пика нагрузки миграция должна быть срочно выполнена, иначе возникает риск отказа перегруженного сервера. Следовательно, в таком случае приходится приостанавливать некоторые приложения на других машинах, чтобы освободить ресурсы, необходимые для проведения миграции. Из-за этого время отклика приостановленных приложений сильно растёт, что может быть крайне нежелательно в облачных системах с преобладанием клиентских приложений интерактивного типа. Отсюда вытекает решение проблемы путём минимизации количества миграций на всём временном отрезке работы облачной системы.

[1] Service Level Agreement.

Чтобы минимизировать общее число миграций в системе, можно либо осуществлять миграции реже, либо минимизировать количество миграций в каждый момент времени. Рис. 1 и 2 иллюстрируют разницу между этими двумя подходами.

рис.1

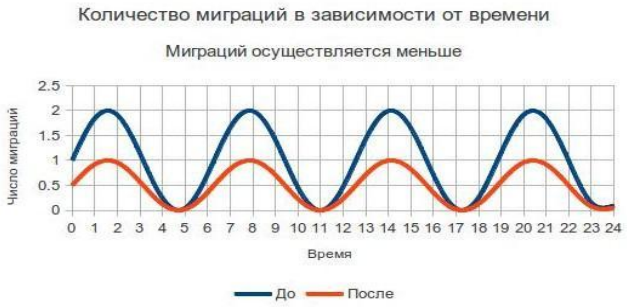

рис.2

Первый подход естественным образом связан со стремлением к равномерному распределению нагрузки по всем серверам:
$min_{\text{по всем картам распределения } VM}\ max_{p_1,p_2 \in P} \mid load(p_1) - load(p_2) \mid$, где P -- множество имеющихся физических серверов. Эта идея нашла своё применение в том, чтобы всегда вычислять оптимальное распределение VM по серверам и реализовывать его. Этот подход требует существенных затрат, даже несмотря на то, что миграции производятся редко, так как обычно всякий раз приходится осуществлять большое количество миграций. Поэтому данное решение не может применяться повсеместно.

Второй подход заключается в заблаговременном определении возможных пиков нагрузки и их предотвращении. Существующие подходы в этой области реализованы для некоторых конкретных систем с облачной архитектурой и в расчёте на периодичность нагрузки системы (часы, дни, месяцы).

Здесь и родилась идея теоретической части данной работы, заключающаяся в модификации и последующем сравнении обоих подходов, а также изготовлении расширяемого и легко внедряемого продукта -- адаптивного планировщика распределения нагрузки в системе. В качестве модификации первого подхода рассматривается поиск не глобально оптимального, но близкого к оптимальному распределения. Модификация второго подхода предполагает runtime-анализ временных рядов загрузки машин.

Облачные системы различаются по преобладающему типу клиентских приложений, работающих на них. Возникает вопрос: какие модели планировщика подходят к различным системам? Данный вопрос послужил отправной точкой для практической части работы.

В качестве тестового стенда для работы, а также первого потребителя программного продукта была выбрана существующая актуальная система мониторинга Cirrocumulus [1].

Обзор существующих решений

Мониторинг и распределение загрузки серверов в системах с облачной архитектурой -- свежая сфера исследований, только начавшая набирать популярность. Многие открытые облачные системы, например, OpenStack [2], сейчас используют простейшие планировщики нагрузки на физические сервера.

В настоящее время исследования в данной области сфокусированы на двух предметах: поиске оптимального решения в зависимости от различных метрик (в основном в перегруженных[2] системах) и анализе заблаговременно собранной статистики загрузки серверов с расчётом на её периодичность.

Первый подход предлагает постоянный мониторинг облачной системы и регулярную оценку её загруженности. На основе этой оценки принимается решение о необходимости перераспределения VM между физическими серверами. В случае необходимости миграций вычисляется карта оптимального относительно выбранной метрики распределения VM по серверам. Целью перераспределения является равномерность нагрузки на все сервера в сравнении их друг с другом. F. Ma, F. Liu и Z. Liu предлагают в своей работе [3] для вычисления оптимальной карты миграций модификацию метода TOPSIS [4]: выбор сервера для миграции VM осуществляется на основе заранее выбранных признаков. Для каждой VM выбирается наиболее подходящий относительно заданных параметров сервер.

Модификации первого подхода нацелены на реализацию полученной оптимальной карты миграций при условии минимизации количества миграций. Авторы этих исследований предлагают генетический алгоритм для вычисления самого “дешёвого” пути реализации оптимальной карты перемещений VM [5]. Также проводятся исследования, отдельно посвящённые перегруженным системам. Перегрузка системы наступает в тот момент, когда она начинает обслуживать в каждый момент времени большее количество пользовательских приложений (VM), чем позволяют ресурсы физических серверов. Благодаря тому, что каждое приложение при разворачивании всегда требует большее количество ресурсов (памяти), чем оно на самом деле будет использовать большую часть своей “жизни”, перегрузка системы даёт значительный выигрыш в

[2] Systems with resource over-commit.

скорости работы клиентских сервисов, так как позволяет одновременно работать большему количеству приложений. Однако встаёт острый вопрос риска внезапной нехватки ресурсов каким-то приложениям и, следовательно, нарушения SLA. Соответственно, в данном случае акцент в исследованиях делается на необходимости своевременного и точного определения пиков загрузки серверов в облаке и вычисления риска отказа машин [6], а также на минимизации ресурсов, затрачиваемых на осуществление перемещений VM [7].

Второй подход предполагает, что загруженность серверов облака периодична, то есть на них запущены задания, повторяющиеся раз в несколько часов, дней, недель и т.д. Основной идеей этого подхода является анализ заранее собранной статистики за значительный срок для выявления точных периодов с помощью методов машинного обучения и анализа временных рядов. Целью существующих алгоритмов является обеспечение как можно более редкой необходимости каких-либо миграций и практически полное избавление от пиков загрузки системы: заранее зная, как меняется степень нагрузки на сервера во времени, можно обеспечить реализацию оптимальной карты миграций единовременно и избежать каких-либо перемещений VM в будущем. Z. Zhang, H. Wang, L. Xiao и L. Ruan в своей работе [8] представляют в качестве периодов часы и время суток. Также часто рассматриваются недельные периоды (загрузка системы в зависимости от дня недели). Владислав Белогрудов в своих исследованиях на основе платформы OpenStack [2] расширяет этот подход, изучая влияние сезонных трендов (месяцев и времён года) на загрузку серверов в системе с облачной архитектурой [9].

Алгоритм

Метрики загруженности серверов и системы

Загруженность физического сервера зависит от количества и типов ресурсов, потребляемых VM, которые запущены на этом сервере. В данной работе принимаются в рассмотрение такие ресурсы, как процессорное время[3] и оперативная память (RAM). Акцент делается на равномерности нагрузки на сервера, поэтому в качестве метрики загруженности системы была выбрана функция среднеквадратичного отклонения. Соответственно, для вычисления нагрузки на физический сервер p в конкретный момент времени и меры загруженности всей облачной системы предложены следующие формулы.

1. Предварительные вычисления.

   I. Ресурсы, потребляемые одной VM.

   Каждой VM сопоставляется вектор ресурсов, занимаемых ей в данный момент времени: $(\mathrm{vm.cpu}, \mathrm{vm.ram})$. Далее все такие вектора рассматриваются в совокупности и нормализуются стандартным способом по каждому параметру: $x' = \frac{x - min}{max - min}$. Такая нормализация входных параметров часто используется в методах анализа данных и машинного обучения.

   Ресурсы, потребляемые конкретной VM, вычисляются по простой формуле

   $$vmload(vm) = \theta_1 vm.cpu + \theta_2 vm.ram. \quad (1)$$

   Коэффициенты θ_1, θ_2 -- неотрицательные вещественные числа, отражающие важность различных типов ресурсов для конкретной облачной системы. В рассмотренной формуле большое значение θ_1 означает особую ценность процессорного времени, тогда как большое значение θ_2 -- необходимость экономии RAM.

   II. Вычисление занятых ресурсов сервера (нагрузки на сервер).

   Пусть p -- сервер, V -- множество виртуальных машин, работающих в данный момент на p. Тогда количество занятых на p ресурсов

   $$load(p) = \sum_{vm \in V} vmload(vm). \quad (2)$$

[3] CPU time.

III. Вычисление средней загруженности по всем серверам.

Обозначим P множество всех серверов и будем вычислять среднюю загруженность по всем серверам как среднее арифметическое занятых ресурсов на каждом сервере системы.

$$mean = \frac{1}{|P|}\sum_{p \in P} load(p) \qquad (3)$$

2. Наконец, мера загруженности всей облачной системы вычисляется как среднеквадратичное отклонение от средней загруженности по всем физическим серверам.

$$L = \frac{1}{|P|}\sum_{p \in P} (load(p) - mean)^2 \qquad (4)$$

Реактивные миграции

Была рассмотрена модификация первого из вышеозначенных подходов к миграции -- миграции на пике загрузки облачной системы, или реактивные миграции.

Выделяются следующие шаги алгоритма:

1. Организовать мониторинг системы -- регулярный сбор следующей информации о каждом сервере:
   I. какие виртуальные машины запущены на сервере в данный момент времени;
   II. какова ресурсная загрузка сервера в данный момент времени.
2. По формуле (4) определить меру загруженности системы в данный момент времени L.
3. Сравнить L с пороговым значением ρ. Значение ρ зависит от мощности (т.е. количества ресурсов) конкретной облачной системы и является параметром задачи.
   I. Если $L < \rho$, вернуться к п.1.
   II. Если $L \geq \rho$, вычислить карту миграций и перейти к п.4.
4. Реализовать полученную карту миграций и перейти к п.1.

Вычисление и реализация карты миграций

Вычисление карты миграций зависит от мощности системы. В данной работе предполагается, что в клиентской облачной системе зарезервировано некоторое количество ресурсов, достаточное для осуществления любых миграций и доступное в любой момент времени. С практической точки зрения это означает, что есть отдельный сервер, на котором не запущено ни одной VM и который отводится для вычислений, связанных с миграциями. Например, этот сервер может быть промежуточным в том случае, если невозможно сразу мигрировать выбранную VM с сервера A на сервер B из-за нехватки ресурсов для запуска этой VM на сервере B в данный момент времени.

В зависимости от мощности системы выбирается допустимое число миграций. Алгоритм построения карты миграций заключается в переборе всех распределений VM по серверам и выборе оптимального относительно меры загруженности системы (4) распределения с учётом ограничения на количество миграций. Данный алгоритм был оптимизирован с помощью жадного относительно уменьшения меры загруженности системы (4) выбора проблемных, т.е. в данном случае потребляемых наибольшее число ресурсов, VM и серверов, куда эти VM нужно мигрировать. Жадный алгоритм выигрывает у изначального в сложности (скорости), но проигрывает в точности, так как жадные алгоритмы не всегда находят оптимум.

Реализация карты миграций остаётся на усмотрение клиентской облачной системы, однако рекомендуется начинать с миграции тех VM, которые “направлены” на наименее загруженные сервера. Выполнение этой рекомендации уменьшит количество обращений к зарезервированным серверам.

Проактивные миграции

В противовес ряду существующих решений, основанных на предподсчёте по заранее собранным статистическим данным, был рассмотрен новый вариант второго из описанных подходов, проактивная миграция. Основная идея нового метода заключается в принятии решения о необходимости миграции на основе последних результатов мониторинга совместно с новыми данными.

Алгоритм состоит из следующих шагов:

1. Организовать мониторинг системы (см. п. ”Реактивные миграции”), причём в каждый момент времени хранить результаты трёх последних сборов информации про сервера.
2. В данный момент времени собрать новые данные и высчитать меру загруженности системы. Таким образом, есть уже 4 точки во времени.
3. Провести линейную интерполяцию по имеющимся точкам. Рис.3 иллюстрирует результат.

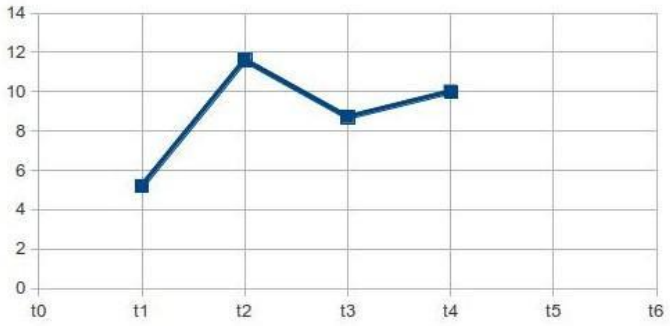

рис. 3

4. Из п.3 получаем значения производных нагрузки по времени на отрезках (t_1, t_2), (t_2, t_3) и (t_3, t_4) -- a_1, a_2 и a_3 соответственно. Производные равны коэффициентам наклона звеньев интерполяционной ломаной.
5. Сравнить с τ величины $a_2 - a_1$, $a_3 - a_2$. Если обе величины меньше τ, перейти к п.6. Иначе перейти к п.1. Здесь τ зависит от конкретной облачной системы и является параметром задачи.
6. Вычислить карту миграций по схеме, предложенной выше в п. “Вычисление и реализация миграций”.
7. Реализовать полученную карту миграций и перейти к п.1.

Предложенная модель, использующая 4 последних результата о мере загруженности системы, направлена на выявление и предотвращение внезапных пиков загрузки. В зависимости от системы также можно использовать линейную комбинацию n производных $\sum_{i=1}^{n} \gamma_i a_i$ с целью распознавания паттернов, присущих конкретной системе. Величина n зависит от конкретной системы.

Реализация

Был создан продукт на языке Python, реализующий описанные выше модификации обоих подходов. Продукт предоставляет широкий ряд возможностей для расширения: можно рассматривать новые модели загруженности как отдельных серверов, так и всей системы. Для этого достаточно написать собственную функцию загрузки и передавать её в качестве аргумента уже реализованным функциям высших порядков. В качестве примера такого поведения можно привести вычисление загрузки сервера и меры загруженности системы.

1. Загрузка сервера.

```
def server_load(ps, vm_load_function=vm_load_simple, vm_rm=None, vm_add=None):
```

В вычислении загрузки сервера используется передаваемая аргументом функция, считающая количество ресурсов, потребляемых VM.

2. Мера загруженности системы.

```
def mse(server_load_impl=server_load_simple, ps_from=None, ps_to=None, vm=None):
```

Мера загруженности системы вычисляется в зависимости от выбранной функции загрузки сервера.

Благодаря своей масштабируемости продукт может служить инструментом для сравнения различных подходов к распределению ресурсов в системах с облачной архитектурой.

API программы представляет собой функции от ID виртуальных машин и физических серверов, в то время как внутри хранится список объектов классов VM и Server, содержащих характеристики виртуальных машин и серверов соответственно. Поэтому модуль может быть легко интегрирован в существующие системы мониторинга. В частности, проведена интеграция модуля в актуальную систему мониторинга Cirrocumulus [1], написанную на языке Ruby.

Код программы находится по ссылке https://github.com/red-cheese/cirrocumulus-vm-balancer/tree/scheduler-dev.

Предваряющий интеграцию анализ данных, заключающийся в предподсчёте параметров системы $(\theta_1, \theta_2, \rho, \tau)$, был проведён в среде RStudio. Эта среда также использовалась для визуализации данных и для исследования алгоритмов анализа.

Для оценки эффективности реактивного подхода были использованы реальные данные, собранные в процессе работы системы Cirrocumulus [1]. Для оценки проактивного подхода был использован собственный генератор данных загрузки, имитирующий работу крупных облачных систем с большим и постоянно изменяющимся количеством клиентов (т.е. VM).

Результаты

Результаты работы алгоритма

В качестве метрики для сравнения различных подходов и алгоритмов была выбрана визуализация зависимости меры загруженности облачной системы от времени. Такой выбор объясняется самими целями данной работы: сделать нагрузку на сервера в облаке равномерной и предотвратить (или оперативно исправить) пики загрузки конкретных серверов. Таким образом, можно утверждать, что алгоритм работает эффективно, если построенный график является графиком либо почти везде близкой к константе функции, либо функции, близкой к периодической.

Исследование показало, что для небольших облачных систем с постоянной клиентурой (примером такой системы может как раз служить Cirrocumulus [1]) наиболее подходящим является реактивный подход, поскольку пики нагрузки случаются редко и обычно знаменуют собой нестандартные ситуации в работе клиентских сервисов (например, ошибки в программном обеспечении или недостаток ресурсов для обработки слишком большого количества данных). Поэтому даже нечастое перераспределение VM помогает поддерживать систему в состоянии максимально равномерной серверной загрузки.

Напротив, для больших облаков с непрекращающимися активными изменениями в структуре клиентских приложений (постоянное появление новых приложений, удаление старых, частое изменение конфигураций запущенных сервисов) лучше подходит проактивный подход к планированию ресурсов и осуществлению миграций, так как загруженность таких систем меняется часто и резко.

Ниже рис.4 иллюстрирует разницу между двумя вышеозначенными типами облачных систем и двумя подходами к планированию ресурсов и реализации миграций соответственно.

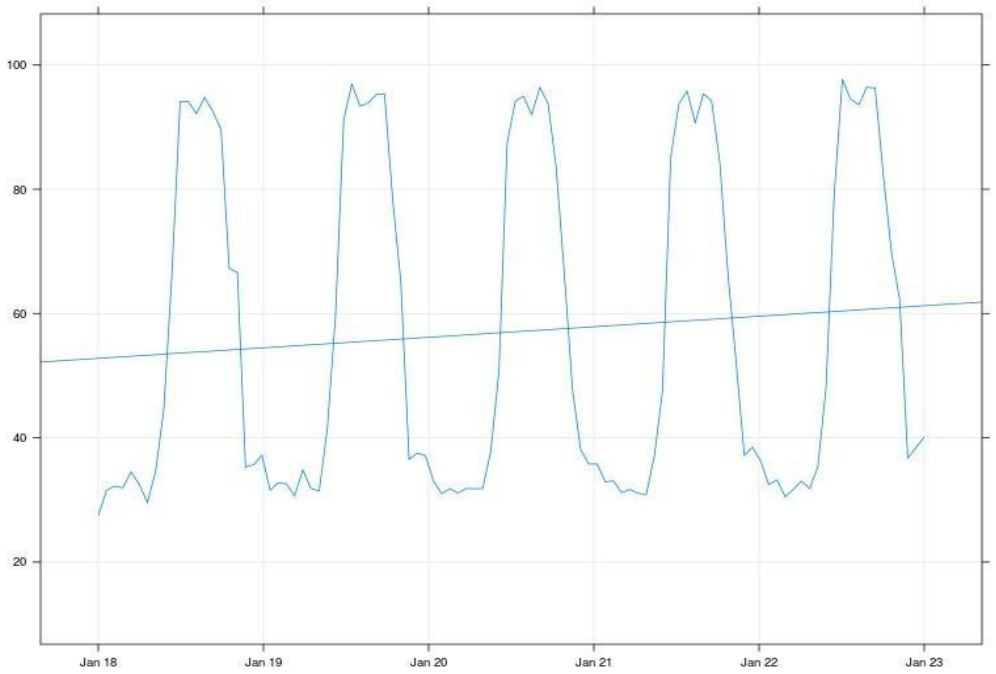

рис. 4

Выводы

В процессе проведения данного исследования были выполнены все поставленные в начале работы задачи:

1. был проведён анализ существующих реализованных подходов к анализу меры загруженности системы с облачной архитектурой и к миграции виртуальных машин;
2. были разработаны новые алгоритмы для анализа загрузки облачной системы и организации перемещений VM с одного сервера на другой;
3. был создан легко внедряемый в систему мониторинга и расширяемый программный продукт;
4. разработанный программный продукт был опробован на одной из актуальных систем мониторинга;
5. была исследована применимость различных моделей в облачных системах разных типов;
6. была организована платформа для оценки и сравнения алгоритмов распределения ресурсов в системе с облачной архитектурой;

7. были поставлены интересные и актуальные вопросы, изучение которых может продолжить исследование выбранной темы в целом и данную работу в частности.

Дальнейшие исследования

Данная работа открыла большой простор для дальнейших исследований в сфере динамического планирования ресурсов в системах с высоким уровнем абстракции, в частности поставив несколько интересных задач, достойных упоминания.

В проведённых исследованиях предполагалось, что в системе всегда зарезервировано некоторое количество ресурсов, достаточное для проведения любых миграций. Интересным представляется вопрос создания механизма осуществления миграции в случае отсутствия свободных ресурсов.

В случае проактивного подхода к миграции интересно изучить применимость нелинейной интерполяции, сравнив новые результаты с уже полученными результатами интерполяции линейной. Здесь можно рассматривать интерполяцию как полиномами, так и другими элементарными функциями.

Также интересно продолжить исследования для облачных систем разного типа, пробуя разные функции для вычисления ресурсов, потребляемых VM, нагрузки на сервер и меры загруженности системы. Например, функции класса сигмоид могут стать хорошей альтернативой взвешенной линейной комбинации (1).

Список литературы

[1] Косякин А.Н., Cirrocumulus -- система управления IT-инфраструктурой // Компьютерные инструменты в образовании, №1, 2012 г., стр. 4-13.

[2] http://www.openstack.org/ OpenStack Open Source Cloud Computing Software

[3] F. Ma, F. Liu, Z. Liu, Distributed Load Balancing Allocation of Virtual Machine in Cloud Data Center // IEEE 3rd International Conference on Software Engineering and Service Science, 2012, pp. 20-23.

[4] C.L. Hwang, K. Yoon, Multiple attributes decision making methods and applications: a state of the art Survey // Springer-Verlag, New York, 1981.

[5] J. Hu, J. Gu, G. Sun, T. Zhao, A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment // 3rd International Symposium on Parallel Architecture, Algorithms and Programming (PAAP), 2010, pp. 89-96.

[6] R. Ghosh, V.K. Naik, Predictive Analysis for Resource Over-commit in IaaS Cloud // IEEE 5th International Conference on Cloud Computing, 2012, pp. 25-32.

[7] X. Zhang, Z.Y. Shae, S. Zheng, H. Jamjoom, Virtual Machine Migration in an Over-committed Cloud // IEEE Network Operations and Management Symposium, 2012, pp. 196-203.

[8] Z. Zhang, H. Wang, L. Xiao, L. Ruan, A Statistical based Resource Allocation Scheme in Cloud // International Conference on Cloud and Service Computing, 2011, pp. 266-273.

[9] V. Belogrudov, Tenant Behaviour-driven Scheduler in OpenStack Cloud // Cork Institute of Technology, Department of Computing, 2013.