Московский государственный университет имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра автоматизации систем вычислительных комплексов

Отчет по решению задания 3

Разработка и реализация алгоритма решения прикладной задачи и проведение его экспериментального исследования

Задание выполнил студент 421 группы
Алексей Ершов

Москва
2019

1 Содержательная постановка задачи

Для выполнения задания требуется реализовать алгоритм для решения задачи распределения виртуальных машин (ВМ) по серверам ЦОД. В условии задания была представлена следующая содержательная постановка задачи.

Дано N заявок на размещение виртуальных машин в ЦОД. Каждая виртуальная машина характеризуется количеством запрашиваемых ядер и требуемым объемом оперативной памяти. Каждый сервер характеризуется количеством ядер и объёмом оперативной памяти. Виртуальная машина должна полностью разместиться на одном сервере (т.е. ядра и память, выделенные виртуальной машине должны быть на одном и том же сервере).

Задача заключается в отыскании отображения из множества номеров заявок на размещение ВМ во множество номеров серверов, которое ставит в соответствие номеру заявки номер сервера, на котором данная ВМ будет размещена. Для некоторых номеров заявок такое отображение построить невозможно - это означает, что данную заявку алгоритм разместить не смог.

В качестве алгоритма был предложен алгоритм, сочетающий жадные стратегии и ограниченный перебор.

2 Математическая постановка задачи

Дано:

- $S = \{ <C_1, R_1>, ..., <C_n, R_n> \}$ - множество серверов, где пара $<C_i, R_i>$ определяет характеристики i-го сервера, а именно: количество ядер и количество оперативной памяти в гигабайтах.
- $V = \{ <c_1, r_1>, ..., <c_k, r_k> \}$ - множество заявок на размещение виртуальных машин, где пара $<c_j, r_j>$ определяет характеристики j-ой ВМ, аналогично характеристикам сервера.

Функция $M(i) = j$, если i-ая ВМ размещена на j-ом сервере и -1, если i-ая ВМ не размещена.

Отображение M корректно $\Leftrightarrow$

$$\forall j \in [1, n] \sum_{i \in [1, k]: M(i) = j} c_i \le C_j \;\&\; \forall j \in [1, n] \sum_{i \in [1, k]: M(i) = j} r_i \le R_j.$$

Функция $D(i) = 1$, если $M(i) > 0$ и 0, если $M(i) < 0$.

Целевая функция: $f_M = \sum_{i=1}^{k} D(i)$.

Требуется найти: M, при котором достигается $\max f_M$, где максимум берётся по всем возможным M.

3 Алгоритм решения задачи

Для решения задачи будем использовать алгоритм, сочетающий жадные стратегии и ограниченный перебор. Опишем его схему.

3.1 Выбор жадного критерия

Первым шагом нужно определить жадный критерий: количество ядер или объём оперативной памяти.

Для этого сравним следующие 2 числа:

$$\rho_1 = (\sum_{i=1}^{n} C_i) / (\sum_{i=1}^{k} c_i)$$

$$\rho_2 = (\sum_{i=1}^{n} R_i) / (\sum_{i=1}^{k} r_i).$$

Если $\rho_1 < \rho_2$, то количество ядер суть жадный критерий. В противном случае - объём оперативной памяти.

3.2 Сортировка

Для простоты будем считать, что множества S и V суть списки.

На данном этапе мы сортируем эти 2 списка следующим образом: если количество ядер есть жадный критерий, то сортируем список S по убыванию значения C_i, а список V - по возрастанию c_j. Иначе - аналогично, но по параметрам R и r соответственно.

3.3 Размещение ВМ

На каждом шаге алгоритм выбирает по порядку сервер из отсортированного списка S и пытается разместить первую неразмещённую машину из списка V, проверяя критерий корректности отображения M (для $\forall i \in [1, k]$, для которых соответствующая i-ая ВМ уже размещена). Иными словами, зададим новый список пар $<CL_m, RL_m>$, $m \in [1, n]$, где CL_m суть сумма всех c_i, где i - номер уже размещённой в данный момент машины на сервере с номером m, а RL_m - аналогично сумма всех таких r_i. Таким образом, на каждом шаге алгоритм проверяет корректность

отображения *M,* суммируя значения c_i и r_i текущей заявки с номером *i* со значениями CL_j, RL_j для текущего сервера с номером *j* и проверяя, что данные суммы не превысят значений C_j и R_j соответственно.

Если же для данного сервера критерий корректности не выполняется, то алгоритм пытается “разместить” данную ВМ на следующем сервере и т.д.

При успешном размещении всех ВМ, алгоритм завершается. Если алгоритм не может найти такой сервер, на котором можно разместить текущую заявку, то он переходит к п. 3.4

3.4 Ограниченный перебор

Пусть для ВМ с номером *p* не нашлось доступного для размещения сервера. Тогда алгоритм запускает так называемую процедуру ограниченного перебора, суть которой в следующем. Составляется новый список серверов S^*, в который входят все сервера из списка *S*, для которых верно следующее: $\forall j \in [1, n]$ сервер с номером *j* включается в список S^*, если $c_p \le C_j \wedge r_p \le R_j$. Далее данный список сортируется в порядке убывания доступного количества критического ресурса, то есть в порядке убывания значения $(C_j - CL_j)$ - в случае, когда критическим ресурсом является количество ядер, и $(C_j - RL_j)$ - в случае количества оперативной памяти. Далее выбираются первые λ серверов из списка S^*, где λ - параметр алгоритма. Затем алгоритм освобождает данные сервера от размещённых на них машин и заново пытается переразместить с помощью жадного алгоритма данные машины, включая машину с номером *p,* на первые λ серверов из списка S^*. Если это не удаётся, то $M(p) = -1$.

3.5 Останов

Алгоритм заканчивает работу, когда для $\forall i \in [1, k]$ определено значение $M(i)$, равное либо номеру соответствующего сервера, либо *-1*.

4 Экспериментальное исследование

Для проведения экспериментального исследования была сгенерирована выборка запросов и конфигураций серверов. Данные записывались в отдельные файлы в формате XML, каждый из которых содержал либо конфигурацию запроса на размещение виртуальных машин, либо конфигурацию серверов. Формат представления данных описан в приложении.

Алгоритм выполнялся для всех возможных пар файлов из множества файлов запросов и множества файлов серверов. Таким образом, имея n запросов и k конфигураций серверов, получаем $n \cdot k$ размещений виртуальных машин.

Данные генерировались с помощью Python-скрипта *generate_data.py,* написанным при помощи библиотеки *lxml.* Данный скрипт позволял генерировать нужное количество запросов и конфигураций серверов с возможностью задания пользователем фиксированного размера запроса и конфигурации серверов. В противном случае, размеры запросов конфигураций серверов задавались случайными числами из заданного интервала. Информацию о параметрах скрипта можно получить с помощью команды: *./generate_data.py --help* или *./generate_data.py -h.*

По результатам экспериментального исследования, проведённого на выборке размера 10×10 (количество запросов и конфигураций серверов соответственно), при соотношении среднего размера запроса ко среднему размеру конфигурации серверов *2:1* более *80%* успешно размещённых запросов, когда все ВМ из запроса размещаются на сервера, достигается при параметре алгоритма λ, равным ⅓ от среднего размера конфигурации серверов.

Стоит подметить, что при малых значениях λ, когда алгоритм был близок к классическому жадному алгоритму, результаты были хуже, что показывает преимущества сочетания жадных стратегий и ограниченного перебора.

5 Выводы

В данной работе было рассмотрено решение прикладной задачи размещения виртуальных машинах на серверах ЦОД при помощи алгоритма, сочетающего жадные стратегии и ограниченный перебор.

По результатам экспериментального исследования дополнение к классическому жадному алгоритму процедуры ограниченного перебора показало более успешные результаты в сравнении с обычным жадным алгоритмом, эмуляция которого достигалась при небольших значениях параметра алгоритма λ.

6 Приложение

Данные представлялись в следующем виде:

<configuration n= n **>**

<vm core_num=" c_0 **" ram="** r_0 **"/>**

<vm core_num=" c_1 **" ram="** r_1 **"/>**

…

<vm core_num=" c_k **" ram="** r_k **"/>**

</configuration>

- для описания запросов на размещения виртуальных машин и

<configuration n= n **>**

<serv core_num=" C_0 **" ram="** R_0 **"/>**

<serv core_num=" C_1 **" ram="** R_1 **"/>**

…

<serv core_num=" C_k **" ram="** R_k **"/>**

</configuration>

- для описания конфигурации серверов ЦОД

При этом каждый *i-ый* тэг **vm** или **serv** соответствует *i-ой* ВМ и серверу.

Для генерации XML-данных использовалась библиотека языка Python lxml (https://lxml.de/). Для парсинга в C++ коде использовалась библиотека pugixml (https://pugixml.org/).