АЛГОРИТМ РАСПРЕДЕЛЕНИЯ РЕСУРСОВ МЕЖДУ АКТИВНЫМИ ПОДСИСТЕМАМИ

Вахранев А.В.
ФИЦ «Информатика и управление» РАН, Москва, Россия
anton22255@ya.ru

Аннотация. Модели с приоритетом в действиях центра могут использоваться для анализа схем управления. При управлении должны учитываться интересы активных подсистем. Интерес вызывают задачи распределения ресурсов дата-центром. Ниже приводится алгоритм нахождения наибольшего гарантированные выигрыш центра при распределении ресурсов между подсистемами.

Ключевые слова: распределение ресурсов, активные подсистемы, дата-центр.

Введение

В сети Интернет вещей (IoT) наблюдается резкий рост числа подключённых устройств. По прогнозам [1–3] в 2030 году в мире будет около 500 млрд устройств IoT. Глобальная сеть IoT будет заниматься интеграцией людей, процессов и технологий. Уже сейчас доступно множество приложений и сервисов. Ресурсы хранения и вычисления ограничены ввиду физических ограничений на размеры устройств. Для решения данной проблемы используется технология облачных вычислений (cloud computing), которая представляет из себя один или несколько дата-центров, которые предоставляют ресурсы приложениям и сервисам. Как правило, дата-центр состоит из кластеров со значительной вычислительной мощностью и ресурсами хранения данных. Доступ осуществляется в сети интернет через входные каналы контролера, который распределяет потоки данных. Упрощенную схема дата-центра представлена на рис.1.

Многие приложения IoT требуют реагирования в режиме реального времени. Поэтому особенно остро стоит задача распределения ресурсов. В данной работе рассматривается одна из возможных задач распределения ресурсов.

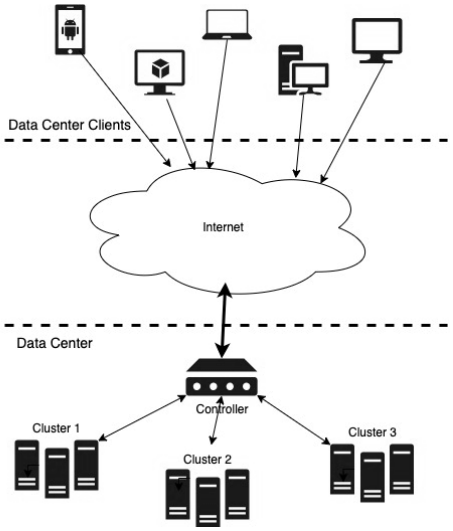

Рис. 1. Модель дата-центра

1. Пример

Рассмотрим задачу планирования ресурсов дата-центра, которая учитывает иерархическую структуру органов управления. Перед дата-центром стоит цель обработать задач типа $\{t_p\}$, $1 \le p \le q$. Пусть планирующий орган Π_0 (далее будем употреблять термин-центр) влияет на производительность подсистем Π_J, выделяя им дополнительные ресурсы $\{u_i^j\}$ для обработки задач. Положим, что подсистемы Π_J независим друг от друга. Каждая из Π_J изначально имеет $r_i\{r_i^j\}$ ресурсов. У каждой задачи типа t_p есть потребность в вычислительных ресурсах b_p. За исполнение

задачи типа t_p Π_J получает награду w_p. Кроме того, работа ресурса r_i требует затраты энергии в размере e_i. Положим стоимость единицы энергии равной для всех подсистем со значением Q.

Для подсистемы Π_J запишем ограничения: $T_0 = \sum_p t_p b_p \le \sum_i r_i + \sum_i u_i$.

Функцию выигрыша Π_J определим следующим образом: $F_i = \max\left(\sum_p w_p t_p - Q * \sum_p b_p e_p \right)$. Так как подсистемы независимы, то функцию выигрыша можно объединить: $T = \sum F_i = \max\left(\sum_p w_p t_p - Q * \sum_p b_p e_p \right)$

Пусть задача Центра заключается в том, чтобы обработать максимальное количество задач. Функцию цели можно сформулировать следующим образом: $F_0 = \max \sum_i t_i$. Получили задачу распределения вычислительных ресурсов дата-центра.

2. Задача

В работе [5] отмечено, что сформулированная выше задача по распределению Центром некоторого ресурса может быть записана в двухуровневой системе в общем виде:

найти

$$\max_{u \in D} \left(\min_{x \in T(u)} \sum_{j=1}^{n} k_j x_j \right) = \max_{u \in D} F(u), \tag{1}$$

где

$$T(u) = \left\{ x \mid x \in T_{\boldsymbol{o}}(u),\ \sum_{j=1}^{n} c_j x_j = \max_{y \in T_0(u)} \sum_{j=1}^{n} c_j y_j \right\}, \tag{2}$$

$$T_{\boldsymbol{o}}(u) = \left\{ x \mid x \in E^n, x \ge 0,\ \sum_{j=1}^{n} a_{ij} x_j = b_i + \sum_{l=1}^{k} b_{il} u_l,\ i = 1,...,m \right\}, \tag{3}$$

$$D = \left\{ u \mid u \in E^n,\ u \ge 0,\ \sum_{l=1}^{k} d_{rl} u_l \le d_r,\ r = 1,...,p \right\}. \tag{4}$$

Существует алгоритм решения задачи (1–4), описанный в работе [5]. Алгоритм опирается на доказанную теорему.

Теорема. Существует $\delta_0 > 0$ такое, что при всех δ, $0 < \delta < \delta_0$, функция $F_0^{\delta} = \sum_{j=1} k_j x_j$, где $x \in T^{\delta}(u)$, однозначна и $F(u) = F_0^{\delta}(u)$,

$$T^{\delta}(u) = \left\{ x \,\middle|\, x \in T^0(u), \sum_{j=1}^{n} (c_j - \delta k_j) x_j = \max_{y \in T_0(u)} \sum_{j=1}^{n} (c_j - \delta k_j) y_j \right\}. \tag{5}$$

Реализация алгоритма требует некоторой подготовки. В данной работе предложим иной вариант решения задачи, который в дальнейшем поможет в отладке реализации алгоритма [5], в сравнении, полученных результатов и затраченного времени.

3. Схема алгоритма

Для иллюстрации решения задачи рассмотрим пример:

$$F_0(u) = \sum_{j=1}^{6} k_j x_j = x_1 + 5x_2 + 6x_4 + 8x_5 + x_6 \tag{6}$$

$$\sum_{j=1}^{6} c_j x_j = x_1 + 5x_2 + 5x_3 + 3x_4 + 2x_5 + x_6 \tag{7}$$

$$T_0(u)=\begin{cases}6x_1+4.2x_2+3.6x_3+1.8x_4+1.8x_5=6-0.6u,\\ 2.4x_2+3.2x_3+5.6x_4+5.5x_5+8x_6=0.8u\\ x\in E^6, x_i\geq 0\end{cases} \quad (8)$$

$$D=\{u|u\in E^1, u\in[0,10]\} \quad (9)$$

Часть решения представим в коде, чтобы проиллюстрировать выбранное решение.

Перепишем исходную задачу (1–4) через последовательность подзадач и запишем решение в коде на языке Python:

- подзадача $P_0(u)$ определяет область в зависимости от параметра u

$$P_0(u)=\begin{cases}D\\ T_0(u)\end{cases} \quad (10)$$

- подзадача $P_1(u)$ объединяет область $P_0(u)$ и $T(u)$

$$P_1(u)=\begin{cases}P_0(u)\\ T(u)\end{cases} \quad (11)$$

$P_1(u)$ – задача линейного программирования, которую можно решать с помощью математических пакетов программирования. К сожалению, не все пакеты находят все решения задачи линейного программирования. Воспользуемся пакетом gurobipy [6], в котором есть параметры PoolSearchMode, PoolSolutions, PoolGap, которые отвечают за количество найденных решений. Для примера (6–9) запишем решение в коде:

```
import gurobipy as gp
from gurobipy import *

def function_P1(u):
    # Create a model
    model = gp.Model("P_1")

    # Parameters
    model.Params.PoolSearchMode = 2
    model.Params.PoolSolutions = 10**8
    model.Params.PoolGap = 0.0

    # Create variables
    x = model.addVars(6, vtype='C', name="x")
    # x1, x2, x3, x4, x5, x6 = [model.addVars(f"x{i+1}") for i in range(6)]

    # Add linear constraints:
    c1= [6,4.2,3.6,1.8,1.8, 0]
    c2= [0, 2.4, 3.2,5.6,5.6,8.0]
    u1 = 6.0-0.6*u[0]
    u2 = 0.8*u[0]
    model.addConstr(sum(c1[i]*x[i] for i in range(6)) == u1, "c1")
    model.addConstr(sum(c2[i]*x[i] for i in range(6)) == u2, "c2")

    obj = [1, 5, 5, 3, 2, 1]
    # Set objective: maximize x
    model.setObjective(quicksum(obj[i]*x[i] for i in range(6)), GRB.MAXIMIZE)

    # Optimize
    model.optimize()
    return model

```

- Все решения для задачи найдем с помощью следующей функции find_all_solution():

```
def find_all_solution(model):
    solution = []
    # Iterate over all found solutions
    for k in range(model.SolCount):
```

```
5.          m.Params.SolutionNumber = k
6.          solution.append([var.Xn for var in model.getVars()])
7.      return solution
8.
```

- подзадача $P_2(u)$ объединяет область $P_1(u)$ и $T(u)$

$$P_2(u) = \begin{cases} P_1(u) \\ F(u) = \min\limits_{x \in T(u)} \sum\limits_{j=1}^{n} k_j x_j \end{cases} \tag{12}$$

Для $P_2(u)$ напишем код, который реализует её логику

```
1. coef_F = [1.0, 5.0, 0.0, 6.0,8.0, 1.0]
2.
3. def find_target_F(model):
4.     res_values = []
5.     for sol in find_all_solution(model):
6.         res = sum(coef_F[i]*x[i] for i in range(6))
7.         res_values.append(res)
8.     return res_values
9.
10. def calculate_all_variants(model):
11.     all_values = find_target_F(model)
12.     min_value = min(all_values)
13.     return min_value
14.
15. def function_P2(u):
16.     model = function_p1(u)
17.     p2_u = calculate_all_variants(model)
18.     return p2_u
```

Имея реализацию функции, $P_2(u)$ можно найти значения на области определения $u \in [0,10]$. Получившийся график для $P_2(u)$ представлен ниже на рис. 2. Предварительно заметим, что оптимальное значение функции $P_2(u)$ приближается к 6 при u близком к 7.

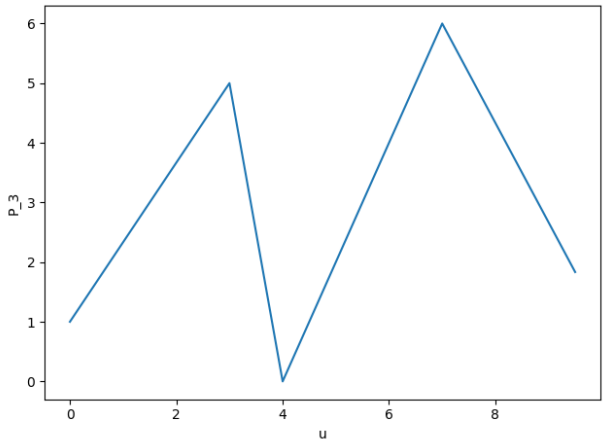

Рис. 2. График функции $P_2(u)$

- финальная подзадача $P_3(u)$

$$P_3(u) = \max_{u \in D} P_2(u) \tag{13}$$

Получаем эквивалент исходной задачи. Для поиска решения $P_3(u)$ будем пользоваться генетическим алгоритмом, реализованным в пакете *c*. Вначале запишем функцию приспособленности (fitness function):

```
1. import pygad
2. def fitness_func(ga_instance, solution, solution_idx):
3.     fitness = function_P2(solution)
4.     return fitness
5.
```

Теперь запишем код для поиска решения с помощью пакета PyGAD [7]

```
 1. import pygad
 2. ga_instance = pygad.GA(num_generations=16,
 3.                        num_parents_mating=5,
 4.                        sol_per_pop=25,
 5.                        num_genes=1,
 6.                        gene_space={"low": 0, "high": 10},
 7.                        mutation_by_replacement=True,
 8.                        fitness_func=fitness_func,
 9.                        on_generation=on_generation)
10. ga_instance.run()
11. ga_instance.plot_fitness()
12.
```

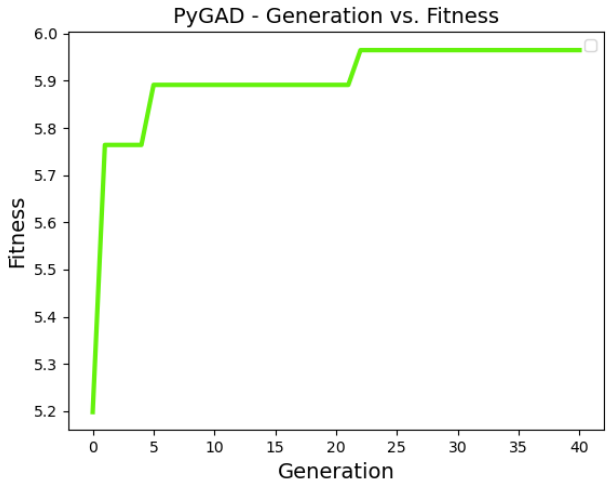

Рис. 3. График работы генетического алгоритма PyGAD

Результатом работы будет итеративный прогон генетического алгоритма: программа выведет оптимальное значение функции fitness_func для разных поколений и график рис. 3. В результате получим оптимальное значение: $P_3(7.01610915) = 5.973151420883319$.

4. Заключение

Как отмечалось ранее, реализованный метод является лишь началом работы над алгоритмом [5] и ставит цель: предварительные расчёты для отладки алгоритма. К преимуществам данного метода можно отнести время реализации. Кроме того, используя генетический алгоритм, можно решать не только задачи линейного программирования: ограничение накладывает программный пакет gurobipy. К недостаткам можно отнести скорость работы: для задачи P_3 приходится находить все решения. В дальнейшем планируется продолжение работы: реализация алгоритма [5]и его сравнение с текущим.

Отметим, что в данной работе рассматривается некоторая упрощённая модель дата-центра. Более сложную модель можно также описать и представить в виде иерархической структуры. Данная работа фокусируется главным образом на алгоритме распределения ресурсов. Поэтому такой упрощённой модели достаточно для постановки базовой задачи.

Литература

1. *Vni C.* Cisco visual networking index: Forecast and trends, 2017–2022 white paper. Technical report (2019)
2. *F., F., K., C., S., N.* Intelligent Internet of Things: From Device to Fog and Cloud. Springer, Springer Nature Switzerland AG (2020).

3. *Andrew, W., Anurag, A., Li, X.*: The Internet of things – a survey of topics and trends. // Information Systems Frontiers 17(2), 2015. – P. 261–274.
4. *Kaur, M., Aron, R.* Withdrawn: Energy-aware load balancing in fog cloud computing. // MaterialsToday: Proceedings, 2020.
5. *Ерешко Ф. И., Злобин А. С.* Алгоритм централизованного распределения ресурса между активными подсистемами. // Экономика и мат. методы, 1977. – Т. 13. Вып. 4. – С. 703–713.
6. Gurobi optimization [Электронный ресурс]. URL: www.gurobi.com/ (дата обращения 17.08.2024)
7. PyGAD // Python Genetic Algorithm [Электронный ресурс]. URL: pygad.readthedocs.io/en/latest/ (дата обращения 17.08.2024)