МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики

Кафедра информационных технологий

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 5
по дисциплине
«МНОГОАГЕНТНОЕ МОДЕЛИРОВАНИЕ»

Выполнил студент группы 45/2 _____ Т. Э. Айрапетов

Направление подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем
Курс 4

Отчет принял доктор физико-математических наук,
профессор _____ А.И. Миков

Краснодар
2024 г.

Задание:

1. Разработать алгоритм распределения и динамического перераспределения нагрузки в многоагентной системе. Входными (изменяемыми) данными являются граф нагрузки и граф МАС (многоагентной системы).

2. Реализовать модель нагрузки, модель многоагентной системы.

3. Реализовать алгоритм распределения и динамического перераспределения нагрузки в многоагентной системе для совместной работы с моделями нагрузки и МАС.

4. Продемонстрировать работу алгоритма на 3 − 4 тестовых примерах с различными моделями нагрузки и МАС (графы нагрузки и МАС и поэтапное изменение нагрузки агентов модулями приложения). Для каждого теста определить общее время выполнения приложения.

Решение.

Для решения задачи были разработаны классы LP и Agent. Класс LP представляет собой программный модуль и соответствующий ему логический процесс. В нем реализованы следующие поля:

- *load* − величина нагрузки;
- *loading* − время с начала запуска задачи;
- *before* − множество процессов-родителей;
- *after* − множество процессов-потомков;
- *denial_time* − время отказа, либо -1;
- *agent* − агент, выполняющий процесс.

Также в классе LP реализован метод *possible_to_start*, возвращающий True, если все процессы-родители были выполнены и False иначе. Для определения времени отказа (*denial_time*) была реализована функция, которая с вероятностью 0.05 возвращает время отказа и с вероятностью 0.95 возвращает -1 (отказа не будет).

В классе Agent реализованы следующие поля:

- *ended_tasks* – количество выполненных задач;
- *cur_task* – ссылка на текущий LP;
- *tasks* – список набранных задач;
- *neighbors* – множество соседей агента.

Также в классе реализована функция подсчёта общей нагрузки агента (сумма нагрузок всех задач в списке + нагрузка текущей задачи).

Модель нагрузки представляется ориентированным нагруженным графом. Для создания тестовых используется функция *gen_lp*, принимающая на вход кол-во «слоёв» программы (независимые друг от друга программные модули, находящиеся на одном уровне от стартовых модулей), минимальную нагрузку модуля и максимальную нагрузку модуля. Функция случайным образом в заданных диапазонах создает граф нагрузки, связывая каждый следующий слой с предыдущим так, чтобы у каждой задачи в слое был как минимум предок. На рисунке 1 можно увидеть пример создания графа нагрузки. Для каждого процесса в указана информация в формате:

(<нагрузка>) LP<индекс> i/o=<степень по входу>/<степень по выходу>

```
слой 0:
 {(4) LP0 i/o=0/2}
слой 1:
 {(1) LP2 i/o=1/4, (3) LP1 i/o=1/2}
слой 2:
 {(4) LP6 i/o=2/1, (5) LP4 i/o=1/1, (5) LP5 i/o=1/2, (1) LP3 i/o=2/2}
слой 3:
 {(3) LP7 i/o=3/0, (4) LP8 i/o=3/0}
```

Рисунок 1 - Пример графа нагрузки (без указания отдельных ребёр)

Модель многоагентной системы представляется неорентированным графом, поэтому для создания тестовых примеров была реализована функция *gen_agents*, принимающая на вход кол-во агентов в системе. Функция создает *n* агентов и связывает их таким образом, чтобы каждый

агент имел как минимум одного соседа. При этом не гарантируется, что в графе будет только 1 компонента связности. На рисунке 2 можно увидеть пример создания графа многоагентной системы.

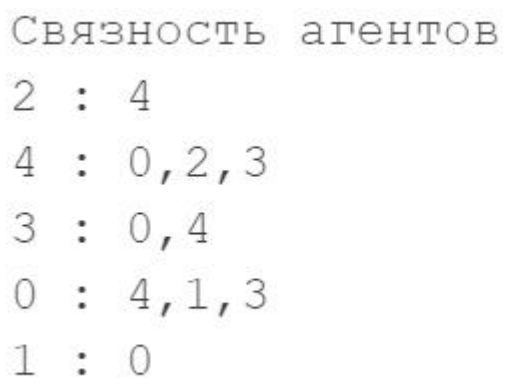

Рисунок 2 - Список связности графа многоагентной системы

Для моделирования распределения нагрузки в системе была создана функция *system*, проводящая как первичное распределение, так и перераспределение.

При распределении нагрузки используется функция *get_most_idle_agent*, которая для всех агентов в системе вычисляет наиболее свободного по 2 показателям: его общая нагрузка (метод *total_load*) и кол-во соседей. То есть вначале функция смотрит кто из агентов менее загружен задачами, а если есть совпадения, то выбирается агент с наибольшим числом соседей. Таким образом, агент с большим числом соседей будет иметь возможность перераспределить нагрузку более эффективно.

Далее в цикле с условием на завершение всех задач, происходит проверка всех агентов на предмет завершения какой-либо задачи, либо отказ в обслуживании. При отказе происходит перераспределение задачи на наиболее свободного агента. Пример моделирования можно увидеть на рисунке 3. Во время работы программы в консоль выводятся логи событий (завершение задачи, отказ в обслуживании). Система завершает свою работу с завершением последнего процесса. После завершения системы

также выводится и максимальное время работы (время выполнения всех задач при последовательном выполнении).

```
4.0: Задача 0 завершена агентом 4
5.0: Задача 2 завершена агентом 0
6.9: Задача 1 завершена агентом 3
8.0: Задача 3 завершена агентом 3
10.0: Задача 4 завершена агентом 1
10.1: Задача 5 завершена агентом 0
10.9: Задача 6 завершена агентом 2
13.8: Задача 7 завершена агентом 4
14.1: Задача 8 завершена агентом 3
Максимальное время работы программы: 30
```

Рисунок 3 - Пример запуска системы

Также после завершения работы можно вывести количество решенных задач по агентам. На рисунке 4 представлен вывод этих данных. Можно обратить внимание, что агенты с меньшим кол-вом соседей получают задачи в среднем реже.

```
Агент 2, кол-во реш. задач: 1
Агент 4, кол-во реш. задач: 2
Агент 3, кол-во реш. задач: 3
Агент 0, кол-во реш. задач: 2
Агент 1, кол-во реш. задач: 1
```

Рисунок 4 - Количество решенных задач агентами

Далее представлены еще несколько запусков при разных графах нагрузки и многоагентной системы.

```
слой 0:
 {(7) LP4 i/o=0/2, (6) LP2 i/o=0/3, (8) LP0 i/o=0/1, (10) LP3 i/o=0/2, (8) LP1 i/o=0/1}
слой 1:
 {(2) LP5 i/o=2/1, (5) LP8 i/o=1/1, (1) LP7 i/o=2/1, (10) LP6 i/o=2/1, (9) LP9 i/o=2/1}
слой 2:
 {(5) LP10 i/o=5/3}
слой 3:
 {(1) LP12 i/o=1/1, (4) LP11 i/o=1/2, (7) LP13 i/o=1/1}
слой 4:
 {(2) LP16 i/o=1/0, (1) LP14 i/o=1/0, (5) LP17 i/o=1/0, (1) LP15 i/o=1/0}
```

Рисунок 5 - Граф нагрузки 5 слоев, мин. нагрузка 1, макс. нагрузка 10

```
Связность агентов
2 : 0,1
0 : 2
1 : 2
```

Рисунок 6 - Граф МАС для 3 агентов

```
6.1: Задача 2 завершена агентом 0
7.1: Задача 4 завершена агентом 2
8.1: Задача 0 завершена агентом 1
15.2: Задача 1 завершена агентом 2
16.2: Задача 3 завершена агентом 0
17.2: Задача 9 завершена агентом 1
19.2: Задача 5 завершена агентом 1
20.3: Задача 7 завершена агентом 1
21.3: Задача 8 завершена агентом 0
25.3: Задача 6 завершена агентом 2
30.3: Задача 10 завершена агентом 1
31.4: Задача 12 завершена агентом 0
32.4: Задача 15 завершена агентом 1
35.4: Задача 11 завершена агентом 0
37.4: Задача 13 завершена агентом 2
37.4: Задача 16 завершена агентом 0
38.5: Задача 14 завершена агентом 0
40.4: Задача 17 завершена агентом 1
Максимальное время работы программы: 92

  Агент 2, кол-во реш. задач: 4
  Агент 0, кол-во реш. задач: 7
  Агент 1, кол-во реш. задач: 7
```

Риснуок 7 - Запуск системы при условиях рис. 5 и 6

```
слой 0:
 {(54) LP1 i/o=0/2, (57) LP0 i/o=0/1, (68) LP2 i/o=0/2, (24) LP3 i/o=0/2}
слой 1:
 {(13) LP4 i/o=3/2, (86) LP6 i/o=1/2, (40) LP7 i/o=1/3, (14) LP5 i/o=2/1}
слой 2:
 {(45) LP9 i/o=2/2, (32) LP8 i/o=2/1, (75) LP10 i/o=2/1, (53) LP11 i/o=2/3}
слой 3:
 {(76) LP15 i/o=2/1, (44) LP13 i/o=2/1, (56) LP12 i/o=1/1, (33) LP16 i/o=1/1, (88) LP14 i/o=1/1}
слой 4:
 {(49) LP19 i/o=1/1, (26) LP17 i/o=2/1, (10) LP18 i/o=2/2}
слой 5:
 {(62) LP21 i/o=2/3, (25) LP20 i/o=2/4}
слой 6:
 {(83) LP24 i/o=1/1, (53) LP25 i/o=2/1, (65) LP22 i/o=1/1, (66) LP26 i/o=2/1, (55) LP23 i/o=1/1}
слой 7:
 {(83) LP27 i/o=5/2}
слой 8:
 {(11) LP28 i/o=1/2, (30) LP29 i/o=1/1}
слой 9:
 {(24) LP31 i/o=1/0, (62) LP30 i/o=2/0}
```

Рисунок 8 - Граф нагрузки 10 слоев, мин. нагрузка 10, мак. нагрузка 100

Связность агентов

8 : 6,7,4
0 : 4,7
7 : 8,0,4,2,5,9
4 : 8,0,7
1 : 9
2 : 7,3
5 : 7
9 : 1,7,3
6 : 8
3 : 2,9

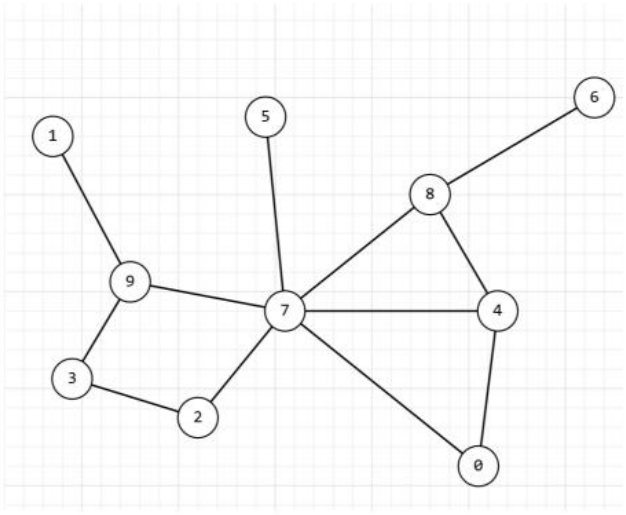

Рисунок 9 - Граф MAC 10 агентов

```
24.0: Задача 3 завершена агентом 9
36.0. Отказ в обслуживании на агенте 8 (Задача 0). Перенос на агента 7
54.0: Задача 1 завершена агентом 7
68.0: Задача 2 завершена агентом 4
81.1: Задача 4 завершена агентом 0
82.0: Задача 5 завершена агентом 1
111.0: Задача 0 завершена агентом 7
140.0: Задача 6 завершена агентом 2
150.9: Задача 7 завершена агентом 3
182.9: Задача 8 завершена агентом 6
193.0: Задача 11 завершена агентом 1
195.9: Задача 9 завершена агентом 5
225.9: Задача 10 завершена агентом 0
226.0: Задача 16 завершена агентом 7
251.8: Задача 12 завершена агентом 6
269.0: Задача 15 завершена агентом 3
269.8: Задача 13 завершена агентом 9
274.9: Задача 19 завершена агентом 1
284.0: Задача 14 завершена агентом 5
295.0: Задача 17 завершена агентом 8
305.1: Задача 18 завершена агентом 8
330.0: Задача 20 завершена агентом 9
367.0: Задача 21 завершена агентом 4
395.0: Задача 22 завершена агентом 8
413.0: Задача 24 завершена агентом 6
...
551.0: Задача 31 завершена агентом 3
563.4: Задача 29 завершена агентом 9
625.4: Задача 30 завершена агентом 4
Максимальное время работы программы: 1562
```

```
Агент 8, кол-во реш. задач: 3
Агент 0, кол-во реш. задач: 3
Агент 7, кол-во реш. задач: 4
Агент 4, кол-во реш. задач: 3
Агент 1, кол-во реш. задач: 3
Агент 2, кол-во реш. задач: 2
Агент 5, кол-во реш. задач: 2
Агент 9, кол-во реш. задач: 5
Агент 6, кол-во реш. задач: 3
Агент 3, кол-во реш. задач: 4
```

Рисунок 10 - Запуск системы при условиях рис. 8 и 9

Текст программы на языке Python:

```
import numpy as np


# время отказа, либо -1
def denial_of_service(load, p=0.05):
    b=np.random.choice([0,1], p=[1-p,p])
    if b:
        return round(np.random.choice(np.arange(0, load, 0.1)), 2)
    return -1


class LP:
    def __init__(self, ind, load):
        self.ind=ind
        self.load = load
        self.loading = 0
        self.before = set()
        self.after = set()
        self.denial_time = denial_of_service(load)
        self.agent=None


    def possible_to_start(self):
        return all(map(lambda x:x.load<=x.loading, self.before))


    def __repr__(self):
        return f"({self.load}) LP{self.ind} i/o={len(self.before)}/{len(self.after)}"


    def __str__(self):
        return self.__repr__()




class Agent:
    def __init__(self, ind):
        self.ind=ind
        self.ended_tasks = 0
        self.cur_task = None
```

```
        self.tasks = list()
        self.neighbors = set()

    def total_load(self):
        return sum([i.load for i in self.tasks]) + (self.cur_task.load if self.cur_task else 0)

    def __repr__(self):
        return f"Agent{self.ind}, load={self.total_load()}, {len(self.neighbors)} neighbors"

    def __str__(self):
        return self.__repr__()


from numpy.random import choice

def gen_lp(n_layers=4, min_load=1, max_load=5):
    ind = 0
    arch = choice(5, n_layers)+1
    def gen(k):
        nonlocal ind
        lps = set()
        for _ in range(k):
            lps.add(LP(ind, np.random.randint(min_load, max_load+1)))
            ind += 1
        return lps
    def connect(la,lb):
        n = len(la)
        for i in lb:
            for j in choice(list(la), choice(n//2+1)+1):
                i.before.add(j)
                j.after.add(i)
        for i in la:
            if not i.after:
                t = choice(list(lb), choice(1)+1)
                for j in t:
                    i.after.add(j)
                    j.before.add(i)
```

```
    st = gen(arch[0])
    la = st
    min_time = max(la, key=lambda x:x.load).load
    # print("слой 0:\n",la)
    j = 1
    for i in arch[1:]:
        lb = gen(i)
        connect(la,lb)
        print(f"слой {j-1}:\n",la)
        j+=1
        la = lb
        min_time += max(la, key=lambda x:x.load).load
    print(f"слой {j-1}:\n",lb)

    return st, min_time



def gen_agents(n=4):
    agents = set()
    for i in range(n):
        agents.add(Agent(i))

    for a in agents:
        if not a.neighbors:
            t = choice(list(agents-set([a])), choice(n) + 1)
            for b in t:
                if a != b:
                    a.neighbors.add(b)
                    b.neighbors.add(a)
    print("Связность агентов")
    for a in agents:
        print(a.ind, ': '+','.join([str(i.ind) for i in a.neighbors]))
    return agents
```

```
def get_most_idle_agent(agents):
    n = len(agents)
    return sorted(agents, key=lambda x:(x.total_load(), n-len(x.neighbors)))[0]



def system(st, agents):
    visited = set()
    stack = list(st)
    max_time = 0
    while stack:
        i = stack.pop(0)
        if not (i in visited or i.agent):
            max_time += i.load
            ag = get_most_idle_agent(agents)
            ag.tasks.append(i)
            i.agent = ag
            visited.add(i)
            stack.extend(list(i.after))


    dt = 0.1
    t = 0
    while visited:
        t += dt
        for a in agents:
            if a.tasks or a.cur_task:
                if not a.cur_task:
                    for i in range(len(a.tasks)):
                        if a.tasks[i].possible_to_start():
                            a.cur_task = a.tasks.pop(i)
                            break
                    else:
                        continue
                task = a.cur_task
                task.loading += dt
                if (task.denial_time != -1) and (task.denial_time <= task.loading):
```

```
                        ag1 = get_most_idle_agent(a.neighbors)
                        i = 0
                        for i in range(len(ag1.tasks)):
                            if not ag1.tasks[i].possible_to_start():
                                break
                        ag1.tasks.insert(i, task)
                        a.cur_task = None
                        task.loading=0
                        task.denial_time = -1
                        print(f"{t:.1f}. Отказ в обслуживании на агенте {a.ind} (Задача {task.ind}).
Перенос на агента {ag1.ind}")
                    elif task.loading >= task.load:
                        print(f"{t:.1f}: Задача {task.ind} завершена агентом {a.ind}")
                        a.ended_tasks += 1
                        visited.remove(task)
                        a.cur_task = None
        print("Максимальное время работы программы:", max_time)



    st, min_time = gen_lp(10, 10, 100)
    agents = gen_agents(10)
    system(st, agents)
```

Выводы. В многоагентных системах стратегия распределения нагрузки является важным аспектом для её работоспособности, так как грамотная балансировка может во много раз уменьшить время выполнения задач. Так, в нашем случае, стратегию можно улучшить, более детально сравнивая агентов между собой. Также можно участить операции по перераспределению, однако это может и увеличить время работы системы.