1. **Система узлов (NODES):**

# Создаем список узлов

NODES = [Node(i) for i in *range*(3)]

# Функция для инициализации сети

*async* *def* initialize\_network():

    NODES[0].neighbors = [NODES[1], NODES[2]]

    NODES[1].neighbors = [NODES[0], NODES[2]]

    NODES[2].neighbors = [NODES[0], NODES[1]]

У нас есть 3 узла, каждый из которых может обрабатывать до 100% нагрузки.

1. **Балансировка нагрузки (Wave Algorithm):**

# Волновой алгоритм Финна для сбора данных о загрузке

*async* *def* finn\_wave\_algorithm(*source\_node*):

    logging.info(*f*"Запуск волнового алгоритма от узла {*source\_node*.node\_id}")

    # Сбрасываем флаги посещения

    for node in NODES:

        node.visited = False

        node.distance = *float*('inf')

*source\_node*.visited = True

*source\_node*.distance = 0

    # Очередь для обработки узлов

    queue = [*source\_node*]

Каждые 10 секунд запускается волновой алгоритм, который:

* Начинается с узла с минимальной нагрузкой
* Передает данные между узлами
* Принимает решение о необходимости переноса нагрузки

1. **Принятие решений о балансировке:**
2. # Централизованное принятие решения о балансировке
3. *async* *def* make\_balancing\_decision(*node\_loads*):
4. logging.info("Начало процесса принятия решения о балансировке")
6. # Находим узел с максимальной нагрузкой
7. max\_load\_node = max(*node\_loads*, key=*lambda* *x*: *x*['load'])
8. logging.info(*f*"Узел с максимальной нагрузкой: {max\_load\_node['node\_id']} ({max\_load\_node['load']}%)")
10. # Находим узел с минимальной нагрузкой
11. min\_load\_node = min(*node\_loads*, key=*lambda* *x*: *x*['load'])
12. logging.info(*f*"Узел с минимальной нагрузкой: {min\_load\_node['node\_id']} ({min\_load\_node['load']}%)")
14. # Проверяем, нужно ли переносить задачу
15. if max\_load\_node['load'] > 70 and min\_load\_node['load'] < 30:
16. logging.info("Превышены пороги нагрузки - требуется перенос задачи")
18. # Имитация переноса задачи
19. max\_load\_node = [node for node in NODES if node.node\_id == max\_load\_node['node\_id']][0]
20. min\_load\_node = [node for node in NODES if node.node\_id == min\_load\_node['node\_id']][0]
22. # Переносим задачу
23. max\_load\_node.decrease\_load(10)
24. min\_load\_node.update\_load(10)
26. logging.info(*f*"Перенос 10% нагрузки с узла {max\_load\_node.node\_id} на узел {min\_load\_node.node\_id}")
28. return {
29. 'action': 'transfer',
30. 'from\_node': max\_load\_node.node\_id,
31. 'to\_node': min\_load\_node.node\_id,
32. 'load\_transferred': 10
33. }
34. else:
35. logging.info("Пороги нагрузки не превышены - нет необходимости в переносе")
37. return {'action': 'no\_action'}

Система проверяет:

* Узел с максимальной нагрузкой
* Узел с минимальной нагрузкой. Если разница нагрузок больше 20%, происходит перенос нагрузки.

При переносе:

* Вычисляется количество нагрузки для переноса (половина разницы)
* Нагрузка уменьшается на отдающем узле
* Нагрузка увеличивается на принимающем узле

# В функции handle\_data\_input

node = min(NODES, key=lambda node: node.load)

node.update\_load() # Увеличиваем нагрузку при начале запроса

# ... выполнение запроса ...

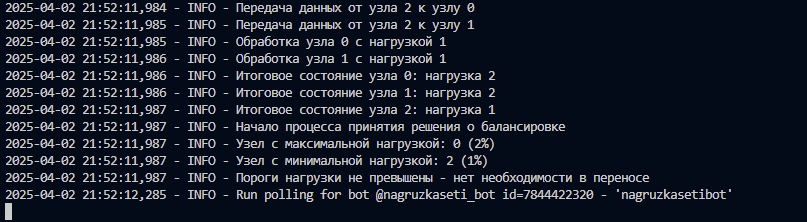
node.decrease\_load() # Уменьшаем нагрузку после выполнения

save\_node\_loads() # Сохраняем состояние нагрузки

При каждом запросе:

* Выбирается узел с минимальной нагрузкой
* Нагрузка увеличивается при начале запроса
* Нагрузка уменьшается после выполнения
* Состояние сохраняется

Вывод в консоли:



Вывод в консоли через 2 минуты:

