https://doi.org/10.25559/SITITO.020.202403.573-582
УДК 004.75
Оригинальная статья

# Организация распределенных вычислений с помощью Docker Swarm и алгоритма Round Robin для балансировки нагрузки

**К. А. Блинов[1*], И. И. Курочкин[2]**
[1] ФГАОУ ВО «Национальный исследовательский технологический университет «МИСИС», г. Москва, Российская Федерация
Адрес: 119049, Российская Федерация, г. Москва, Ленинский пр., д. 4, стр. 1
[2] ФГБУН «Институт проблем передачи информации им. А. А. Харкевича Российской академии наук», г. Москва, Российская Федерация
Адрес: 127051, Российская Федерация, г. Москва, Большой Каретный переулок, д. 19, стр. 1
[*] blinov.kirill.a@yandex.ru

**Аннотация**
Балансировка нагрузки в облачных вычислениях и грид-системах представляет собой важнейшую задачу, поскольку правильное распределение задач и ресурсов между узлами напрямую влияет на производительность и эффективность использования инфраструктуры. С ростом числа пользователей и увеличением сложности вычислительных задач, особенно в средах с высокими требованиями к обработке данных, разработка эффективных алгоритмов балансировки нагрузки становится неотъемлемой частью современных исследований. В статье рассматриваются как статические, так и динамические подходы к распределению нагрузки, с акцентом на использование Docker Swarm и алгоритма Round Robin для грид-систем.
Для иллюстрации предложенного метода используется инфраструктура на основе NGINX и FastAPI, которая позволяет гибко и равномерно распределять входящие запросы на сервера с минимальной загрузкой, снижая риск перегрузки отдельных узлов. В качестве ключевых показателей эффективности предложены загрузка процессора (CPU) и оперативной памяти (RAM). Проводится анализ в контексте их влияния на стабильность и производительность системы. Результаты исследования демонстрируют, что использование Docker Swarm позволяет поддерживать стабильную работу при нагрузке до 60 клиентов, обеспечивая равномерное распределение задач и минимизируя потребление ресурсов.
Представленные результаты распределения вычислительных задач, показывают устойчивость к нагрузке и сокращение времени простоя, что особенно важно для высокоприоритетных задач. Результаты тестирования подтверждают целесообразность применения данного подхода, а также открывают возможности для дальнейшего расширения системы и её адаптации для более крупных вычислительных мощностей.

**Ключевые слова:** балансировка нагрузки, Docker Swarm, Round Robin, грид-системы

**Финансирование:** работа выполнена в рамках государственного задания Министерства науки и высшего образования Российской Федерации (тема № 122040800139-4 «Развитие методов и технологий решения сложных прикладных и научных задач в распределенных вычислительных средах»).

**Конфликт интересов:** авторы заявляют об отсутствии конфликта интересов.

**Для цитирования:** Блинов К. А., Курочкин И. И. Организация распределенных вычислений с помощью Docker Swarm и алгоритма Round Robin для балансировки нагрузки // Современные информационные технологии и ИТ-образование. 2024. Т. 20, № 3. С. 573-582. https://doi.org/10.25559/SITITO.020.202403.573-582

Original article

# Organization of Distributed Computing Using Docker Swarm and the Round Robin Algorithm for Load Balancing

**K. A. Blinov**[a*], **I. I. Kurochkin**[b]
[a] National University of Science and Technology «MISIS», Moscow, Russian Federation
Address: 4 Leninsky pr., build. 1, Moscow 119049, Russian Federation
[b] Institute for Information Transmission Problems of the Russian Academy of Sciences (Kharkevich Institute), Moscow, Russian Federation
Address: 19 Bolshoy Karetny per., build.1, Moscow 127051, Russian Federation
[*] blinov.kirill.a@yandex.ru

## Abstract

Load balancing in cloud computing and grid systems is a crucial task, since the correct allocation of tasks and resources between nodes directly affects the performance and efficiency of infrastructure use. With the growing number of users and increasing complexity of computational tasks, especially in environments with high data processing requirements, the development of effective load balancing algorithms is becoming an integral part of modern research. The article discusses both static and dynamic approaches to load distribution, focusing on the use of Docker Swarm and the Round Robin algorithm for grid systems.

To illustrate the proposed method, we use an NGINX and FastAPI-based infrastructure, which allows for flexible and even distribution of incoming requests among servers with minimal load, reducing the risk of overloading individual nodes.

Key performance indicators, such as processor (CPU) and RAM usage, are analyzed in the context of their impact on system stability and performance. The study demonstrates that using Docker Swarm allows you to maintain stable operation with a load of up to 60 clients, ensuring an even distribution of tasks and minimizing resource consumption.

The presented test results confirm the optimal distribution of computing tasks, showing increased load tolerance and reduced downtime, which is especially important for mission-critical applications. The test results confirm the feasibility of using this approach, as well as open up opportunities for further expansion of the system and its adaptation to larger clusters.

**Keywords:** Load balancing, Docker Swarm, Round Robin, grid systems

**Funding:** The research was carried out within the state assignment of Ministry of Science and Higher Education of the Russian Federation (theme No. 122040800139-4).

**Conflict of interests:** The authors declares no conflict of interest.

**For citation:** Blinov K.A., Kurochkin I.I. Organization of Distributed Computing Using Docker Swarm and the Round Robin Algorithm for Load Balancing. *Modern Information Technologies and IT-Education.* 2024;20(3):573-582. https://doi.org/10.25559/SITITO.020.202403.573-582

## Введение

Распределенные вычисления становятся все более важным элементом современной вычислительной техники. Их можно определить как использование нескольких компьютеров, которые соединены через сеть, охватывающую обширные географические области, вплоть до глобальной сети Интернет. Такие системы предоставляют доступ к общим вычислительным ресурсам, что открывает возможности для решения сложных вычислительных задач в различных отраслях. Одним из главных преимуществ распределенных вычислений является их масштабируемость и возможность организованного использования виртуальных ресурсов. Это позволяет гибко увеличивать или уменьшать доступные вычислительные мощности в зависимости от текущих потребностей, что является важным фактором в условиях быстро меняющейся нагрузки и роста объема данных. Более того, распределенные вычисления предоставляют возможность эффективного управления вычислительными ресурсами, что снижает затраты на содержание и обслуживание оборудования.

Однако вместе с преимуществами распределенные системы сталкиваются и с рядом недостатков. Одной из основных проблем является необходимость абсолютного контроля над ресурсами и нагрузкой. Это усложняет управление такими системами и требует внедрения специальных механизмов, таких как балансировка нагрузки, чтобы избежать перегрузки отдельных элементов сети [1].

Для повышения эффективности распределенных вычислительных систем часто используется балансировщик нагрузки, который принимает задачи из различных мест и распределяет их на централизованные ресурсы, такие как центры обработки данных. Это позволяет лучше использовать доступные ресурсы, повышая общую производительность системы и снижая вероятность простоев или перегрузок. Балансировщики нагрузки играют важную роль в распределенных системах, так как они обеспечивают динамическое распределение задач и ресурсов между различными узлами сети [2].

С развитием таких технологий, как облачные вычисления, распределенные системы и Интернет вещей (Internet of Things, IoT), наблюдается значительное увеличение числа персональных устройств, подключенных к глобальной сети. Эти инновации обеспечивают дальнейшее развитие в области распределенных вычислений, открывая новые возможности для использования виртуальных ресурсов и обработки данных [3]. В контексте IoT балансировщики нагрузки оказывают критическое влияние на эффективность обработки данных. Системы, использующие балансировщики, позволяют более точно и быстро обрабатывать информацию, связанную с транспортом, мобильной связью, домашним оборудованием и другими устройствами. Это обеспечивает высокую доступность и надежность данных, что особенно важно для распределенных систем IoT [4, 5]. Для таких сред важно соблюдать высокие стандарты безопасности, особенно с использованием технологии «Bring Your Own Device» (BYOD), которая позволяет сотрудникам или пользователям подключать личные устройства к корпоративной сети.

В системах, работающих с большими данными (Big Data), балансировка нагрузки также играет ключевую роль. Она включает в себя контроль сетевых подключений к кластеру данных, а также планирование распределения вычислительных ресурсов. Это помогает избежать перегрузки системы и эффективно использовать мощности для анализа и обработки огромных потоков информации. Технология BDaaS (Big Data as a Service) помогает унифицировать и инкапсулировать большие объемы данных, предоставляя централизованные услуги для их обработки и анализа [1].

В облачных системах балансировка нагрузки становится особенно сложной задачей из-за неоднородности инфраструктуры и различных требований к качеству обслуживания для разных приложений. Это приводит к необходимости разработки и внедрения более сложных алгоритмов распределения задач между виртуальными машинами, которые позволяют поддерживать стабильную работу систем даже при высоких нагрузках [6, 7].

Распределенные вычисления продолжают играть важную роль в современном мире, особенно в условиях стремительного роста объемов данных и числа подключенных устройств. Балансировка нагрузки в таких системах является важнейшим механизмом, который обеспечивает эффективное использование ресурсов, повышает производительность и надежность систем.

## Литературный обзор

В зависимости от подхода к распределению задач, алгоритмы балансировки нагрузки делятся на три типа: статические, динамические и динамические алгоритмы, основанные на природных процессах.

Статические алгоритмы балансировки нагрузки основываются на предварительной информации о состоянии системы и используемых приложениях. Эти алгоритмы распределяют задачи на основе известной статистической информации о системе в данный момент времени, что позволяет заранее определить, как нагрузка будет распределена между узлами [1]. Процесс работы таких алгоритмов включает учет характеристик узлов, таких как вычислительные мощности, объем доступной памяти и скорость передачи данных. Одним из ключевых преимуществ статических алгоритмов является их предсказуемость и низкие затраты на системные ресурсы, поскольку они не требуют постоянного мониторинга состояния системы.

1) Алгоритмы статической балансировки нагрузки зависят от предварительно собранной информации о системе, включая такие параметры, как скорость передачи данных, объем памяти и вычислительные мощности узлов. Это позволяет заранее оценить возможности каждого узла и распределить задачи равномерно.

2) Одним из недостатков таких алгоритмов является то, что они не учитывают динамические изменения в системе во время выполнения задач. Это может привести к тому, что нагрузка на узлы изменится в процессе работы, что не будет отражено в статической схеме распределения.

3) Статические алгоритмы часто основываются на заранее заданных правилах распределения нагрузки, что делает их более подходящими для стабильных сред с однородной структурой системы. Такие системы менее подвержены изменениям в процессе работы, что позволяет использовать более простые

алгоритмы без необходимости регулярного обновления информации о состоянии узлов.

4) В условиях реального времени требования пользователей могут изменяться, и использование статических алгоритмов может оказаться неэффективным, так как они не способны оперативно реагировать на изменения нагрузки.

5) Тем не менее, статические алгоритмы часто применяются в системах с однородной архитектурой, где все узлы обладают схожими характеристиками и задачи имеют равную сложность. В таких условиях статические алгоритмы демонстрируют хорошую производительность и минимальные затраты на системные ресурсы.

6) Одним из преимуществ статических алгоритмов является снижение общих системных затрат по сравнению с динамическими алгоритмами балансировки нагрузки, которые требуют постоянного мониторинга и обновления информации о состоянии системы.

Одним из наиболее распространенных методов статической балансировки нагрузки является алгоритм круговой балансировки (Round Robin). Этот метод был предложен Sharma и Singh [8]. Принцип его работы заключается в том, что задачи последовательно распределяются между узлами в порядке очереди. Как только задачи попадают в очередь на исполнение, каждая новая задача назначается следующему узлу, пока все задачи не будут распределены. Важной особенностью этого алгоритма является то, что он не требует обмена информацией между узлами о текущей загрузке. Это делает алгоритм особенно эффективным для систем с одинаковыми по вычислительным возможностям узлами, где задачи равномерны по объему. Round Robin минимизирует время простоев и обеспечивает равномерное распределение задач без сложных расчетов или необходимости обмена данными между узлами.

Другой подход к статической балансировке нагрузки был предложен Wang, Yan, Liao и Wang в виде алгоритма Opportunistic Load Balancing (OLB) [9]. Этот метод предполагает случайное распределение задач между доступными узлами, не учитывая текущую загрузку. Алгоритм OLB нацелен на минимизацию общего времени выполнения задач в системе за счет распределения подзадач таким образом, чтобы максимизировать использование доступных ресурсов. Узлы выбираются на основе минимального времени выполнения подзадач, что позволяет достичь более эффективного распределения нагрузки и сократить общее время выполнения задач. Однако, как и алгоритм Round Robin, OLB не принимает во внимание текущее состояние узлов, что может привести к неравномерному распределению нагрузки в случае неоднородных систем.

В отличие от статических алгоритмов, динамические алгоритмы учитывают текущее состояние сети и способны адаптироваться к изменяющимся условиям. Алгоритмы динамической балансировки нагрузки выполняют постоянный мониторинг системы для выявления слабонагруженных серверов и переноса на них задач с более загруженных узлов. Это позволяет равномерно распределять нагрузку в сети и избегать перегрузки отдельных ресурсов. Ключевая особенность динамических алгоритмов заключается в том, что они учитывают предыдущее состояние системы и вносят корректировки на основе актуальных данных.

1) Одним из важных аспектов динамических алгоритмов является учет предыдущего состояния системы. Это помогает алгоритму принимать решения на основе истории загрузки серверов и предсказывать будущие изменения нагрузки.

2) В отличие от статических алгоритмов, динамические не требуют предварительных настроек или условий для распределения задач. Нагрузка распределяется в зависимости от текущих требований системы, что делает этот подход более гибким и адаптивным.

3) Динамические алгоритмы эффективны как в однородных, так и в гетерогенных средах. Они могут адаптироваться к изменяющимся условиям работы системы, что особенно важно для гетерогенных систем, где узлы могут иметь разные вычислительные возможности.

Одним из эффективных подходов к динамической балансировке нагрузки является алгоритм Load Balanced Min-Min (LBMM), предложенный Kokilavani и соавторами [10]. Этот алгоритм распределяет задачи на основе минимального времени их выполнения для каждого узла. Задачи с минимальным временем выполнения на загруженных ресурсах могут быть перенесены на менее загруженные узлы, если время выполнения на новом узле меньше. Это позволяет добиться лучшего распределения нагрузки и уменьшить общее время выполнения задач в системе.

Алгоритм Active Clustering Algorithm (ACA), предложенный Ram Prasad Padhy и Goutam Prasad Rao, использует информацию о текущем состоянии ресурсов для динамического распределения нагрузки[1]. ACA собирает данные о загрузке процессора, памяти и дискового пространства узлов, а также учитывает текущие запросы. Основываясь на полученных данных, алгоритм предсказывает будущие изменения нагрузки и корректирует распределение задач. Это позволяет поддерживать стабильную работу системы даже при изменении условий нагрузки [11].

Еще одним подходом является Biased Random Sampling (BRS), предложенный Rahmeh [12]. Этот алгоритм использует случайное распределение задач между узлами на основе географической близости и доступных ресурсов. BRS просматривает соседние узлы, пока не находит наименее загруженный. Это снижает необходимость в глобальном мониторинге сети, делая алгоритм более эффективным в распределенных системах, особенно тех, которые работают в больших географически распределенных средах.

Одним из наиболее интересных подходов к динамической балансировке нагрузки является использование генетических алгоритмов (Genetic Algorithm, GA). Алгоритм, предложенный Hu Jinhua и соавторами, балансирует нагрузку, основываясь на исторических данных и текущем состоянии системы [13]. В отличие от традиционных методов, GA не ищет оптимальное решение путем последовательного перебора всех возможных вариантов, а использует эволюционные подходы для нахождения лучшего решения в пространстве возможных вариантов.

[1] Prasad Padhy R., Goutam Prasad Rao P. Load balancing in cloud computing system : Bachelor thesis. Rourkela, India: Department of Computer Science and Engineering National Institute of Technology, 2011 [Electronic resource]. URL: http://ethesis.nitrkl.ac.in/2545/1/load_balancing_in_cloud_computing_systems.pdf (дата обращения: 19.06.2024).

Основное преимущество этого метода заключается в его способности эффективно работать в условиях сложных и многомерных задач распределения. Однако одним из недостатков генетических алгоритмов является их сложность, что может усложнять процесс распределения задач и увеличивать время вычислений.

Традиционные методы балансировки нагрузки, несмотря на их простоту и легкость реализации, часто оказываются неэффективными для сложных систем с динамическими условиями работы. В связи с этим все чаще используются алгоритмы, вдохновленные природой. Природные алгоритмы балансировки нагрузки используют модели биологических процессов для оптимизации распределения задач в системах. В основе этих подходов лежит идея математического моделирования естественных процессов, которые демонстрируют эффективные способы взаимодействия агентов в распределенных системах. Эти процессы можно адаптировать для решения задач динамической балансировки нагрузки в облачных системах.

Алгоритм Ant Colony Optimization (ACO), предложенный Ратаном Мишрой и Анантом Джайсвалом, имитирует поведение муравьев, которые ищут путь к источнику пищи, оставляя за собой феромонный след, сигнализирующий другим муравьям о наиболее оптимальном маршруте [14]. В контексте балансировки нагрузки в облачных системах, каждый сервер может быть представлен как муравей, который выполняет задачи и оставляет феромонный след, указывающий на эффективность обработки запросов. Чем выше эффективность сервера, тем больше запросов ему отправляется, что создает динамическую схему распределения задач. Серверы с меньшей нагрузкой обрабатывают меньше запросов, а более эффективные серверы, получающие больше «феромонов», автоматически привлекают к себе больше задач, что обеспечивает эффективную балансировку нагрузки.

Алгоритм Honey Bee Behavior-inspired Load Balancing (HBB-LB), предложенный Динешем Бабу и Венкатой Кришной, основывается на поведении пчел-разведчиков, которые ищут новые источники пищи и передают информацию другим членам улья [15]. В данном алгоритме, перегруженные узлы в вычислительной системе сравниваются с недогруженными по различным критериям, подобно тому, как пчелы находят источники пищи. Когда задача сталкивается с перегруженным сервером, она «танцует», обновляя данные о своей загрузке и передавая информацию другим задачам, находящимся в очереди, что аналогично танцам пчел-разведчиков. Затем задача перемещается на менее загруженный сервер, подобно тому, как пчела-фуражир находит новый источник пищи. Этот процесс повторяется до тех пор, пока система не достигнет состояния сбалансированной нагрузки.

Алгоритм Load Balancing based on Markov Process Modeling использует математическую модель Марковского процесса для оптимизации распределения задач между виртуальными машинами (VM) в облачной инфраструктуре [16]. В этой модели каждый сервер представлен как очередь, а балансировщик нагрузки (LBer) играет роль центрального сервера, который динамически распределяет задачи между узлами на основе вычислительной мощности и текущей загрузки каждого узла. Алгоритм постоянно пересчитывает коэффициенты загрузки серверов и распределяет задачи пропорционально их возможностям, обеспечивая эффективное использование вычислительных ресурсов и динамическую балансировку нагрузки.

Алгоритм Grey Wolf Optimization (GWO) является метаэвристическим методом, вдохновленным иерархической структурой стаи серых волков [17]. В данном подходе вычислительные узлы сравниваются с волками в стае, которые взаимодействуют друг с другом для достижения наилучших результатов. Сначала определяется пороговое значение нагрузки, после чего узлы с загрузкой выше порогового значения считаются перегруженными, а узлы с меньшей загрузкой — недогруженными. Алгоритм направлен на поиск узлов с оптимальными характеристиками, что позволяет распределить задачи таким образом, чтобы максимально эффективно использовать вычислительные ресурсы.

## Метрики

Для успешной балансировки нагрузки в облачных вычислениях необходимо учитывать как общие метрики производительности, так и специфические параметры состояния сети, связанные с загрузкой ресурсов. Каждое направление развития облачных технологий требует применения собственного набора метрик для оценки эффективности алгоритмов балансировки нагрузки [18].

Ключевые метрики, которые могут быть использованы для оценки эффективности, использованные в других исследованиях представлены в таблице 1. Было выделено восемь основных показателей оценки качества работы балансировки нагрузки:

1) Среднее время выполнения – измеряет время, за которое задачи выполняются на виртуальных машинах (ВМ). Это значение важно для оценки общей производительности системы: чем меньше среднее время выполнения, тем эффективнее распределена нагрузка между ВМ.

2) Время ожидания в очереди – показывает, сколько времени задачи проводят в очереди до начала выполнения. Это критическая метрика, особенно для кластеров, где задачи могут накапливаться в ожидании ресурсов.

3) Степень дисбаланса – отражает разницу между максимальной и минимальной нагрузкой на ВМ. Меньшая степень дисбаланса свидетельствует о равномерном распределении задач, что улучшает общую производительность системы.

4) Число миграций задач – показывает, сколько раз задачи были перемещены между ВМ для достижения баланса. Меньшее число миграций снижает накладные расходы на перемещение задач.

5) Пропускная способность – измеряет количество данных, которое может быть обработано за единицу времени, и напрямую влияет на эффективность обработки запросов.

6) Потребление ресурсов – включает использование процессорных ядер (CPU) и оперативной памяти (RAM), что важно для оптимального распределения нагрузки.

7) Среднеквадратичное отклонение – характеризует отклонение фактической нагрузки на ВМ от среднего значения, а также помогает оценить степень сбалансированности нагрузки.

8) Cost divisor – соотношение числа ВМ, требующих миграции, к общему количеству ВМ, что позволяет оценить затраты на балансировку системы.



Таблица 1. **Используемые метрики в научных источниках**
Table 1. **Metrics used in scientific publications**

| Авторы и статья | Среднее время выполнения | Время ожидания в очереди | Степень дисбаланса | Число миграций задач | Пропускная способность | Потребление ресурсов | Среднеквадратичное отклонение | Cost divisor |
|---|---|---|---|---|---|---|---|---|
| Sefati S., Mousavinasab M., Zareh Farkhady R. Load balancing in cloud computing environment using the Grey wolf optimization algorithm based on the reliability: performance evaluation [17] | + | + | | | | + | | |
| Souravlas S., Anastasiadou S. D., Tantalaki N., Katsavounis S. A Fair, Dynamic Load Balanced Task Distribution Strategy for Heterogeneous Cloud Platforms Based on Markov Process Modeling [16] | + | + | + | | | + | | |
| Dhinesh Babu L. D., Krishna P. V. Honey bee behavior inspired load balancing of tasks in cloud computing environments [15] | + | + | + | + | | | | |
| Mishra R., Jaiswal A. Ant Colony Optimization: A solution of Load Balancing in Cloud [14] | | + | | | + | + | | |
| Hu J., Gu J., Sun G., Zhao T. A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment [13] | | + | | | | | + | + |
| Rahmeh O. A., Johnson P., Taleb-Bendiab A. A Dynamic Biased Random Sampling Scheme for scalable and reliable Grid Networks [12] | | + | | | | | | |
| Prasad Padhy R., Goutam Prasad Rao P. Load balancing in cloud computing system | + | | | | | | | |
| Kokilavani T., Amalarethinam D. G. Load balanced MinMin algorithm for static MetaTask scheduling in grid computing. International Journal of Computer Applications [10] | + | | | | | + | | |
| Wang S.-C., Yan K.-Q., Liao W.-P., Wang S.-S. Towards a Load Balancing in a three-level cloud computing network [9] | + | + | | | + | + | | |
| Sharma S., Singh S., Sharma M. Performance analysis of load balancing algorithms [8] | | + | | | | + | | |

*Источник:* здесь и далее в статье все таблицы и рисунки составлены авторами.
*Source:* Hereinafter in this article all tables and figures were made by the authors.

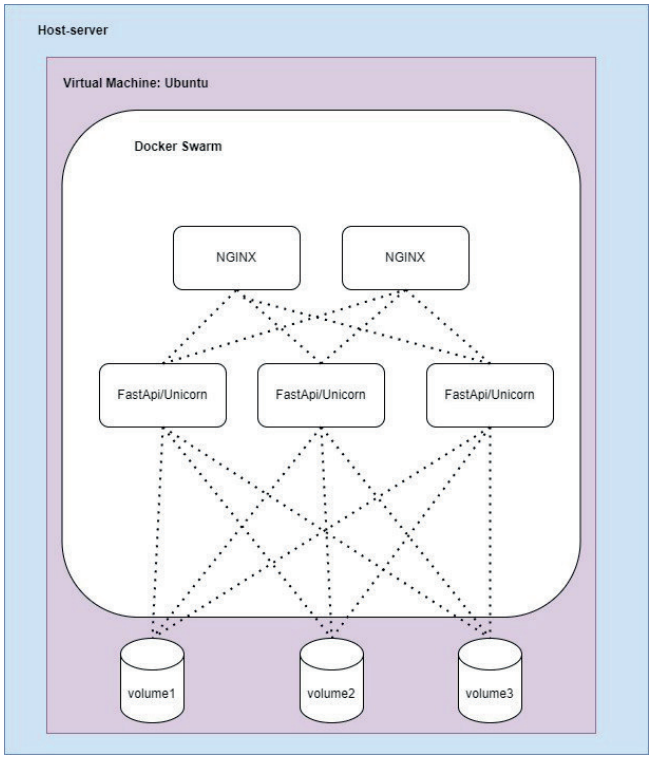
При этом для эффективного управления нагрузкой на уровне кластера важнейшими параметрами являются:
1) Нагрузка CPU на узлах — мониторинг загруженности центрального процессора каждого узла позволяет определить, какие ВМ готовы принять дополнительные задачи, а какие перегружены.
2) Нагрузка RAM на узлах — отслеживание использования оперативной памяти критично для обеспечения того, чтобы ресурсы памяти не становились узким местом при распределении задач.
Данные характеристики состояния сети должны собираться в реальном времени и использоваться для оперативного принятия решений о перераспределении задач. Постоянный мониторинг этих параметров позволяет главному серверу эффективно распределять вычислительные ресурсы, выбирая наиболее свободные узлы в сети и обеспечивая равномерное распределение нагрузки.

## Балансировка нагрузки с помощью Docker Swarm

В Docker Swarm используется механизм под названием «сетка маршрутизации», который объединяет все узлы с помощью «overlay network», основанной на технологии Virtual Extensible LAN (VXLAN)[2] [19]. Эта сеть имеет собственные IP и порты, через которые узлы обмениваются данными, а также осуществляется взаимодействие с внешними клиентами [20]. К overlay network подключены балансировщики нагрузки, размещенные на каждой виртуальной машине, где запущен Docker Swarm.
Контейнер ingress-sbox управляет сетевым трафиком в Docker Swarm и автоматически запускается при создании кластера. Сеть ingress представляет собой оверлейную сеть, которая упрощает балансировку нагрузки между узлами. Когда узел получает запрос на опубликованный порт, он перенаправляет его в модуль IPVS. Эта сеть создается автоматически при инициализации или подключении к кластеру Swarm.
IPVS контролирует все IP-адреса, участвующие в службе, и выбирает один из них для обработки запроса через сеть ingress. IPVS (IP Virtual Server) выполняет балансировку нагрузки с высокой эффективностью, используя алгоритмы со сложностью O(1) [21]. Поддерживаются различные алгоритмы балансировки, такие как циклический перебор (Round Robin) и наименьшее количество подключений (Least Connections) [22]. В алгоритме Least Connections балансировщик направляет новые запросы на сервер с наименьшим количеством активных соединений, что позволяет эффективно управлять нагрузкой [23]. Для достижения оптимальной производительности в кластере Docker Swarm необходимо учитывать состояние его узлов. Это обусловлено тем, что узлы в сети представлены docker контейнерами, выполняющими различные вычислительные задачи, которые затрачивают вычислительные ресурсы сервера. Например, контейнер, запущенный для обработки больших объемов данных, может требовать значительных ресурсов CPU и RAM, в то время как другой контейнер, запущенный для простого веб-сервиса, может использовать минимальное количество ресурсов [24, 25]. Необходимо обеспечить сбалансированное распределение ресурсов в кластере, чтобы гарантировать его эффективность.

## Эксперимент

В качестве эксперимента выбрана задача обработки клиентских запросов. Система узлов состоит из образов. Nginx прослушивает внешний ip адрес сервера и переадресовывать запросы в API-сервис или предоставлять доступ к файлам из набора данных volume. Контейнеры unicorn-server выполняют составление заданий для клиентов. Сеть «custom» объединяет контейнеры «web» и «app» между собой для обеспечения безопасного и эффективного взаимодействия между ними. Тома «first-volume-data», «second-volume-data», «third-volume-data» – это отдельные структуры хранения данных, которые встраивается в файловую систему контейнеров предоставляя доступ к общим файлам в любой момент, обеспечивая сохранность информации и упрощая процессы совместной работы.




Р и с. 1. Структура запущенного кластера выдачи заданий
F i g. 1. The structure of the running task issuing cluster

В данном контексте Docker Swarm функционирует в качестве веб-сервера, осуществляющего маршрутизацию трафика между двумя экземплярами Nginx. Такое архитектурное решение исключает необходимость привязки маршрута к конкретному узлу, что сложно реализовать при использовании только Nginx. После маршрутизации запрос направляется на один из трёх контейнеров, работающих с FastAPI, где осуществляется его обработка.

[2] Soppelsa F., Kaewkasi C. Native Docker Clustering with Swarm. Packt Publishing Ltd, 2016. 280 p.

В ходе тестирования было проведено пять экспериментов. Метриками оценки качества работы системы служили параметры использования центрального процессора (CPUPerc) и объема использованной оперативной памяти (MemPerc). Для объективной оценки тестирования с интервалом в одну секунду четыре раза составлялись списки с оценкой производительности, после чего вычислялось среднее значение четырех замеров по каждому параметру контейнера. Результаты тестирования представлены в таблице 2, а также на рисунках 2 и 3.

Таблица 2. **Средние значения результатов тестирования**
Table 2. **Average values of test results**

| Тестирование | Контейнер | CPUPerc | MemPerc |
|---|---|---|---|
| Система без нагрузки | NGINX-1 | 0 | 15 |
| | NGINX-2 | 0 | 15 |
| | FASTApi-1 | 0,1375 | 0,73 |
| | FASTApi-2 | 0,145 | 0,68 |
| | FASTApi-3 | 0,145 | 0,7 |
| 1 пользователь | NGINX-1 | 2,02 | 15 |
| | NGINX-2 | 1,7625 | 15 |
| | FASTApi-1 | 0,475 | 0,73 |
| | FASTApi-2 | 0,155 | 0,68 |
| | FASTApi-3 | 0,135 | 0,7 |
| 5 пользователей | NGINX-1 | 6,255 | 15,36 |
| | NGINX-2 | 6,2125 | 15,19 |
| | FASTApi-1 | 0,71 | 0,74 |
| | FASTApi-2 | 0,6325 | 0,69 |
| | FASTApi-3 | 0,2 | 0,71 |
| 10 пользователей | NGINX-1 | 11,34 | 15,365 |
| | NGINX-2 | 11,3375 | 15,195 |
| | FASTApi-1 | 0,2325 | 0,74 |
| | FASTApi-2 | 0,21 | 0,69 |
| | FASTApi-3 | 1,21 | 0,71 |
| 15 пользователей | NGINX-1 | 11,79 | 15,38 |
| | NGINX-2 | 11,3775 | 15,125 |
| | FASTApi-1 | 0,68 | 0,74 |
| | FASTApi-2 | 0,1975 | 0,69 |
| | FASTApi-3 | 0,1625 | 0,71 |
| 20 пользователей | NGINX-1 | 11,68 | 15,4625 |
| | NGINX-2 | 11,8175 | 15,335 |
| | FASTApi-1 | 0,8375 | 0,75 |
| | FASTApi-2 | 1,59 | 0,7 |
| 30 пользователей | NGINX-1 | 11,68 | 15,46 |
| | NGINX-2 | 11,82 | 15,34 |
| | FASTApi-1 | 0,84 | 0,75 |
| | FASTApi-2 | 1,59 | 0,7 |
| 40 пользователей | NGINX-1 | 9,98 | 15,48 |
| | NGINX-2 | 9,96 | 15,3675 |
| | FASTApi-1 | 0,215 | 0,75 |
| | FASTApi-2 | 0,6975 | 0,7 |
| 50 пользователей | NGINX-1 | 12,58 | 15,485 |
| | NGINX-2 | 13,28 | 15,3525 |
| | FASTApi-1 | 0,1875 | 0,75 |
| | FASTApi-2 | 0,1725 | 0,7 |
| 60 пользователей | NGINX-1 | 12,58 | 15,485 |
| | NGINX-2 | 13,28 | 15,3525 |
| | FASTApi-1 | 0,1875 | 0,75 |
| | FASTApi-2 | 0,1725 | 0,7 |

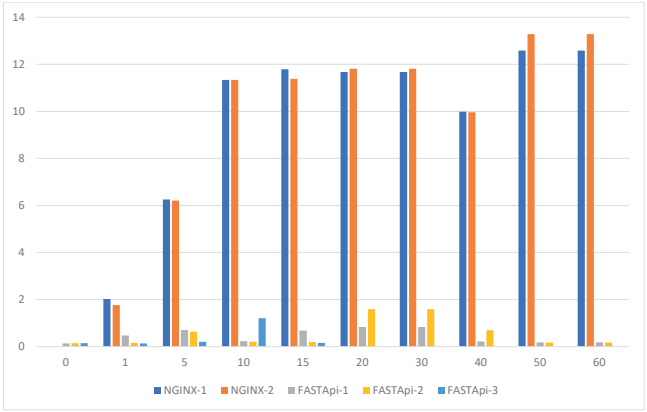



Рис. 2. Использование CPU
Fig. 2. CPU usage

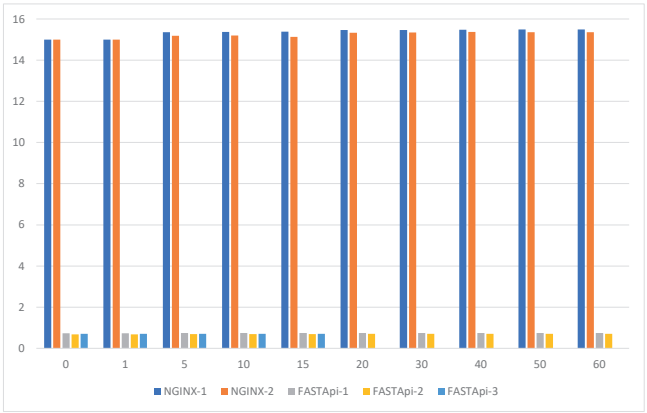



Рис. 3. Использование оперативной памяти
Fig. 3. RAM usage

На основе полученных данных можно сделать несколько выводов:
1) Нагрузка на CPU в контейнерах с NGINX распределяется равномерно, что подтверждает балансировку трафика запросов в системе.
2) Нагрузка на CPU в контейнерах с FastAPI зависит от количества поступающих запросов: чем больше клиентов, тем больше нагрузка распределяется между тремя контейнерами.
3) Оперативная память не оказывает влияния на производительность системы и остается постоянной.

На основе таблицы 2 демонстрируется равномерное распределение нагрузки, как каждый узел выполнял примерно одинаковый объем работы момент времени. Так же применяемы в ходе эксперимента выполняемые задания не повлияли на оперативную память.

## Обсуждение и заключение

Метод Round Robin лучше всего подходит для ситуаций, когда рабочие нагрузки равномерны и предсказуемы, а также когда простота и предсказуемость важнее оптимального использования ресурсов. В результате анализа и экспериментов удалось подтвердить эффективность предложенного подхода, основанного на использовании Docker Swarm для управления распределением задач между контейнерами.

Эксперименты показали, что использование контейнеров NGINX и FastAPI с статическим распределением нагрузки позволяет обеспечить стабильную работу системы при различных уровнях нагрузки, вплоть до 60 клиентов. Нагрузка на CPU распределялась равномерно между контейнерами, что подтверждает корректность работы механизмов балансировки. При этом объем использованной оперативной памяти оставался практически неизменным, что свидетельствует о высокой эффективности использования ресурсов.

Таким образом, предложенная архитектура позволяет улучшить масштабируемость и отказоустойчивость системы, что особенно важно для высоконагруженных вычислительных сред. В перспективе дальнейшего развития можно рассмотреть интеграцию более сложных алгоритмов балансировки, а также расширение системы на больший масштаб узлов и клиентов для повышения общей производительности.

## References

[1] Wang Y.-T. Morris. Load Sharing in Distributed Systems. *IEEE Transactions on Computers*. 1985;C-34(3):204-217. https://doi.org/10.1109/TC.1985.1676564

[2] Calheiros R.N., Ranjan R., Beloglazov A., De Rose C.a.F., Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software Practice and Experience.* 2010;41(1):23-50. https://doi.org/10.1002/spe.995

[3] Varghese B., Buyya R. Next generation cloud computing: New trends and research directions. *Future Generation Computer Systems.* 2017;79:849-861. https://doi.org/10.1016/j.future.2017.09.020

[4] Hou L., Zhao S., Xiong X., Zheng K., Chatzimisios P., Hossain M.S., Xiang W. Internet of Things Cloud: Architecture and implementation. *IEEE Communications Magazine*. 2016;54(12):32-39. https://doi.org/10.1109/mcom.2016.1600398cm

[5] Lopez J., Rios R., Bao F., Wang G. Evolving privacy: From sensors to the Internet of Things. *Future Generation Computer Systems*. 2017;75:46-57. https://doi.org/10.1016/j.future.2017.04.045

[6] Hou L., Zhao S., Xiong X., Zheng K., Chatzimisios P., Hossain M.S., Xiang W. Internet of Things Cloud: Architecture and implementation. *IEEE Communications Magazine*. 2016;54(12):32-39. https://doi.org/10.1109/mcom.2016.1600398cm

[7] Shila D.M., Shen W., Cheng Y., Tian X., Shen A.X.S. AMCloud: Toward a secure autonomic mobile ad hoc cloud computing system. *IEEE Wireless Communications*. 2016;24(2):74-81. https://doi.org/10.1109/mwc.2016.1500119rp

[8] Sharma S., Singh S., Sharma M. Performance Analysis of Load Balancing Algorithms. *World Academy of Science, Engineering and Technology.* 2008;38:269-272.

[9] Wang S.-C., Yan K.-Q., Liao W.-P., Wang S.-S. Towards a Load Balancing in a three-level cloud computing network. In: 2010 3rd International Conference on Computer Science and Information Technology. Chengdu: IEEE Press; 2010. p. 108-113. https://doi.org/10.1109/ICCSIT.2010.5563889

[10] Kokilavani T., Amalarethinam D.G. Load balanced MinMin algorithm for static MetaTask scheduling in grid computing. *International Journal of Computer Applications*. 2011;20(2):42-48. https://doi.org/10.5120/2403-3197

[11] Katoch S., Thakur J. Load Balancing Algoritms in Cloud Computing Environment: A Review. *International Journal on Recent and Innovation Trends in Computing and Communication.* 2014;2(8):2151-2156. https://doi.org/10.17762/ijritcc.v2i8.3673

[12] Rahmeh O.A., Johnson P., Taleb-Bendiab A. A Dynamic Biased Random Sampling Scheme for scalable and reliable Grid Networks. *INFOCOMP Journal of Computer Science*. 2008;7:1-10.

[13] Hu J., Gu J., Sun G., Zhao T. A Scheduling Strategy on Load Balancing of Virtual Machine Resources in Cloud Computing Environment. In: 2010 3rd International Symposium on Parallel Architectures, Algorithms and Programming. Liaoning, China: IEEE Press; 2010. p. 89-96. https://doi.org/10.1109/PAAP.2010.65

[14] Mishra R., Jaiswal A. Ant Colony Optimization: A solution of Load Balancing in Cloud. *International Journal of Web & Semantic Technology*. 2012;3(2):33-50. https://doi.org/10.5121/ijwest.2012.3203

[15] Dhinesh Babu L.D., Krishna P.V. Honey bee behavior inspired load balancing of tasks in cloud computing environments. *Applied Soft Computing*. 2013;13(5):2292-2303. https://doi.org/10.1016/j.asoc.2013.01.025

[16] Souravlas S., Anastasiadou S.D., Tantalaki N., Katsavounis S. A Fair, Dynamic Load Balanced Task Distribution Strategy for Heterogeneous Cloud Platforms Based on Markov Process Modeling. *IEEE Access*. 2022;10:26149-26162. https://doi.org/10.1109/ACCESS.2022.3157435

[17] Sefati S., Mousavinasab M., Zareh Farkhady R. Load balancing in cloud computing environment using the Grey wolf optimization algorithm based on the reliability: performance evaluation. *The Journal of Supercomputing*. 2022;78:18-42. https://doi.org/10.1007/s11227-021-03810-8



[18] Ghomi E.J., Rahmani A.M., Qader N.N. Load-balancing algorithms in cloud computing: A survey. *Journal of Network and Computer Applications.* 2017;88:50-71. https://doi.org/10.1016/j.jnca.2017.04.007
[19] Chiang R.C. Contention-aware container placement strategy for docker swarm with machine learning based clustering algorithms. *Cluster Computing*. 2020;26(1):13-23. https://doi.org/10.1007/s10586-020-03210-2
[20] Ileana M., Oproiu M.I., Viorel Marian C. Using Docker Swarm to Improve Performance in Distributed Web Systems. In: 2024 International Conference on Development and Application Systems (DAS). Suceava, Romania: IEEE Press; 2024. p. 1-6. https://doi.org/10.1109/DAS61944.2024.10541234
[21] Pandey B., Mishra A.K., Yadav A., Tiwari D., Pandey M.S. Virtualization using Docker container. In: Verma A., Verma P., Farhaoui Y., Lv Z. (eds.) Emerging Real-World Applications of Internet of Things. CRC Press-Taylor & Francis Group; 2022. p. 157-181. https://doi.org/10.1201/9781003304203
[22] Kaur S., Kumar K., Singh J., Ghumman N.S. Round-robin based load balancing in Software Defined Networking. In: 2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom). New Delhi, India: IEEE Press; 2015. p. 2136-2139.
[23] Zhu L., Cui J., Xiong G. Improved dynamic load balancing algorithm based on Least-Connection Scheduling. In: 2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC). Chongqing, China: IEEE Press; 2018. p. 1858-1862. https://doi.org/10.1109/ITOEC.2018.8740642
[24] Khalid Y.N., Aleem M., Ahmed U., Islam M.A., Iqbal M.A. Troodon: A machine-learning based load-balancing application scheduler for CPU-GPU system. *Journal of Parallel and Distributed Computing*. 2019;132:79-94. https://doi.org/10.1016/j.jpdc.2019.05.015
[25] Huang D., et al. Achieving Load Balance for Parallel Data Access on Distributed File Systems. *IEEE Transactions on Computers.* 2018;67(3):388-402. https://doi.org/10.1109/TC.2017.2749229

*Поступила 19.06.2024; одобрена после рецензирования 12.08.2024; принята к публикации 26.09.2024.*
*Submitted 19.06.2024; approved after reviewing 12.08.2024; accepted for publication 26.09.2024.*

Об авторах:

**Блинов Кирилл Алексеевич**, аспирант, ФГАОУ ВО «Национальный исследовательский технологический университет «МИСИС» (119049, Российская Федерация, г. Москва, Ленинский пр., д. 4, стр. 1), **ORCID: https://orcid.org/0009-0004-8759-1719,** blinov.kirill.a@yandex.ru

**Курочкин Илья Ильич**, заведующий лабораторией моделирования и анализа телекоммуникационных систем Центра распределенных вычислений, ФГБУН «Институт проблем передачи информации им. А. А. Харкевича Российской академии наук» (127051, Российская Федерация, г. Москва, Большой Каретный переулок, д. 19, стр. 1), кандидат технических наук, **ORCID: https://orcid.org/0000-0002-0399-6208,** qurochkin@gmail.com

*Все авторы прочитали и одобрили окончательный вариант рукописи.*

About the authors:

**Kirill A. Blinov,** Postgraduate Student, National University of Science and Technology “MISIS” (4 Leninsky pr., build. 1, Moscow 119049, Russian Federation), **ORCID: https://orcid.org/0009-0004-8759-1719,** blinov.kirill.a@yandex.ru

**Ilya I. Kurochkin,** Head of the Modeling and Analysis of Telecommunication Networks Lab, Center for Distributed Computing, Institute for Information Transmission Problems of the Russian Academy of Sciences (Kharkevich Institute) (19 Bolshoy Karetny per., build.1, Moscow 127051, Russian Federation), Cand. Sci. (Tech.), **ORCID: https://orcid.org/0000-0002-0399-6208,** qurochkin@gmail.com

*All authors have read and approved the final manuscript.*