ФЕДЕРАЛЬНОЕ АГЕНСТВО СВЯЗИ

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

Московский технический университет связи и информатики

(МТУСИ)

Кафедра «Сетевые информационные технологии и сервисы»

Дисциплина «Анализ производительности информационной системы»

Проект на тему:

«Балансировщик нагрузки»

Выполнили студенты группы М092101(75):

Болябкин М.

Калининский Д.

Горошков Д.

Проверила:

Беленькая М. Н.

Москва 2023

**Оглавление**

[1. Цель работы 3](#_Toc132765381)

[2. Задачи 3](#_Toc132765382)

[3. Исследование балансировщика нагрузки 4](#_Toc132765383)

[3.1. Типы балансировщиков нагрузки 5](#_Toc132765384)

[3.2. Уровни балансировки нагрузки 6](#_Toc132765385)

[3.3. Алгоритмы балансировки нагрузки 7](#_Toc132765386)

[3.4. Рассмотрение применения балансировщика нагрузки в российских фирмах из реестра Роскомнадзора 9](#_Toc132765387)

[3.4.1. Компания NGINX 9](#_Toc132765388)

[3.4.2. Компания Ядро (Core) 10](#_Toc132765389)

[3.4.3. Компания СКАТ 11](#_Toc132765390)

[4. Блок схема балансировщика нагрузки 13](#_Toc132765391)

[5. Разработка кода для балансировщика нагрузки 16](#_Toc132765392)

[6. Вывод 19](#_Toc132765393)

[Список литературы 20](#_Toc132765394)

# **1. Цель работы**

Целью работу является:

1) Изучить принципы и методы балансировки нагрузки в информационных системах;

2) Рассмотреть примеры применения балансировки нагрузки в российских фирмах из реестра Роскомнадзора;

3) Разработать собственное приложение балансировщика на C#.

# **2. Задачи**

Для выполнения цели данной работы необходимо выполнить следующие задачи:

1) Изучить различные алгоритмы балансировки нагрузки и их применение в информационных системах;

2) Разработать приложение балансировщика на языке программирования C#;

3) Реализовать алгоритм round robin в разработанном приложении балансировщика;

4) Протестировать разработанное приложение на примере веб-сервера и оценить его эффективность;

5) Рассмотреть примеры применения балансировки нагрузки в российских фирмах из реестра Роскомнадзора;

6) Проанализировать применение балансировки нагрузки в выбранных фирмах и дать рекомендации по улучшению балансировки нагрузки;

7) Обобщить результаты и сделать выводы о применении балансировки нагрузки в информационных системах;

8) Сформулировать рекомендации по применению балансировки нагрузки в российских фирмах;

9) Описать перспективы развития балансировки нагрузки в информационных системах.

# **3. Исследование балансировщика нагрузки**

Балансировка нагрузки – это метод равномерного распределения сетевого трафика по пулу ресурсов, поддерживающих приложение.

Принципы балансировки нагрузки - это основные правила и стратегии, которые используются при распределении нагрузки на несколько серверов, чтобы обеспечить эффективную и надежную работу системы. Вот несколько основных принципов балансировки нагрузки:

1. Равномерность распределения нагрузки: это один из самых важных принципов балансировки нагрузки. Система должна распределять нагрузку равномерно на все серверы, чтобы избежать перегрузки одних серверов и недостатка нагрузки на других.
2. Определение наиболее эффективных серверов: при балансировке нагрузки система должна определить, какие серверы являются наиболее эффективными для обработки запросов. Это может быть основано на различных факторах, таких как доступность ресурсов, производительность и нагрузка на сервер.
3. Автоматическая настройка: система балансировки нагрузки должна автоматически настраиваться для обеспечения оптимальной работы. Например, если один из серверов отказывает, то нагрузка должна автоматически перенаправляться на другой сервер.
4. Отказоустойчивость: система балансировки нагрузки должна быть отказоустойчивой и способной обработать сбои в работе серверов. Например, если один из серверов выходит из строя, другие серверы должны автоматически принимать дополнительную нагрузку, чтобы сохранить работоспособность системы.
5. Масштабируемость: система балансировки нагрузки должна быть масштабируемой и способной обрабатывать растущую нагрузку на сервера. Например, если число запросов к системе растет, балансировщик нагрузки должен масштабироваться для обработки дополнительных запросов.
6. Безопасность: при балансировке нагрузки система должна обеспечивать безопасность данных и защиту от атак. Например, система должна проверять аутентификацию и авторизацию для доступа к серверам и предотвращать атаки типа DDoS.

Эти принципы обеспечивают эффективную работу системы балансировки нагрузки и помогают обеспечить надежную и безопасную работу информационных систем.

## **3.1. Типы балансировщиков нагрузки**

Существует несколько типов балансировщиков нагрузки, каждый из которых предназначен для определенных целей и задач. Ниже перечислены основные типы балансировщиков нагрузки:

1. Аппаратные балансировщики нагрузки: это физические устройства, которые работают на уровне сети и выполняют функцию балансировки нагрузки между несколькими серверами. Они могут обрабатывать большое количество запросов и обеспечивать высокую производительность и надежность.
2. Программные балансировщики нагрузки: это программное обеспечение, которое выполняет функцию балансировки нагрузки на уровне операционной системы или приложения. Они обычно работают на отдельном сервере и могут быть развернуты как виртуальные машины или контейнеры.
3. DNS-балансировщики нагрузки: это тип балансировщика нагрузки, который использует DNS-сервер для распределения трафика между несколькими серверами. Он работает путем указания нескольких IP-адресов для одного доменного имени, каждый из которых соответствует одному из серверов.
4. Сетевые балансировщики нагрузки: это тип балансировщика нагрузки, который работает на уровне сети и обеспечивает балансировку нагрузки на нескольких узлах внутри сети. Они обычно используются в больших корпоративных сетях для обеспечения высокой доступности и производительности приложений.
5. Cloud-балансировщики нагрузки: это тип балансировщика нагрузки, который работает в облачной среде и обеспечивает балансировку нагрузки на различных облачных серверах. Они могут быть настроены для автоматической масштабируемости и обеспечивают высокую доступность и производительность приложений в облаке.

Каждый тип балансировщика нагрузки имеет свои преимущества и недостатки, и выбор типа зависит от конкретных требований и задач, которые должен решать балансировщик нагрузки.

## **3.2. Уровни балансировки нагрузки**

Балансировка нагрузки может быть выполнена на нескольких уровнях в сетевой инфраструктуре. Ниже перечислены основные уровни балансировки нагрузки:

1. Балансировка нагрузки на уровне DNS: на этом уровне балансировка нагрузки осуществляется путем настройки DNS-сервера для распределения трафика между несколькими серверами, имеющими одинаковый IP-адрес. Когда клиент запрашивает определенный домен, DNS-сервер возвращает один из доступных IP-адресов, и запрос направляется на соответствующий сервер.
2. Балансировка нагрузки на уровне приложения: на этом уровне балансировка нагрузки осуществляется путем настройки программного обеспечения, которое выполняет функции балансировки нагрузки на уровне приложения. Это может быть встроенная функция веб-сервера или отдельный балансировщик нагрузки, работающий на уровне операционной системы.
3. Балансировка нагрузки на уровне сети: на этом уровне балансировка нагрузки осуществляется путем настройки сетевого оборудования, такого как коммутаторы, маршрутизаторы или файрволлы. Балансировка нагрузки может быть выполнена на основе IP-адресов или портов, и запросы могут быть направлены на разные серверы в зависимости от настроек балансировщика нагрузки.
4. Балансировка нагрузки на уровне кластера: на этом уровне балансировка нагрузки осуществляется путем настройки кластера серверов, в котором несколько серверов работают вместе и могут обрабатывать запросы клиентов. Балансировка нагрузки на уровне кластера обеспечивает высокую отказоустойчивость и производительность, поскольку запросы могут быть направлены на другой сервер в случае отказа одного из серверов в кластере.

Каждый уровень балансировки нагрузки имеет свои преимущества и недостатки, и выбор уровня зависит от конкретных требований и задач, которые должен решать балансировщик нагрузки.

## **3.3. Алгоритмы балансировки нагрузки**

Существует множество алгоритмов балансировки нагрузки, каждый из которых имеет свои преимущества и недостатки в зависимости от конкретных требований и сценариев использования. Ниже перечислены некоторые из наиболее распространенных алгоритмов балансировки нагрузки:

1. Round-robin: этот алгоритм распределяет запросы между серверами по круговой схеме. Запросы отправляются на каждый сервер по очереди, начиная с первого сервера в списке и переходя к следующему серверу после каждого запроса. Этот алгоритм прост в реализации и обеспечивает равномерное распределение нагрузки между серверами, но не учитывает текущую нагрузку серверов.

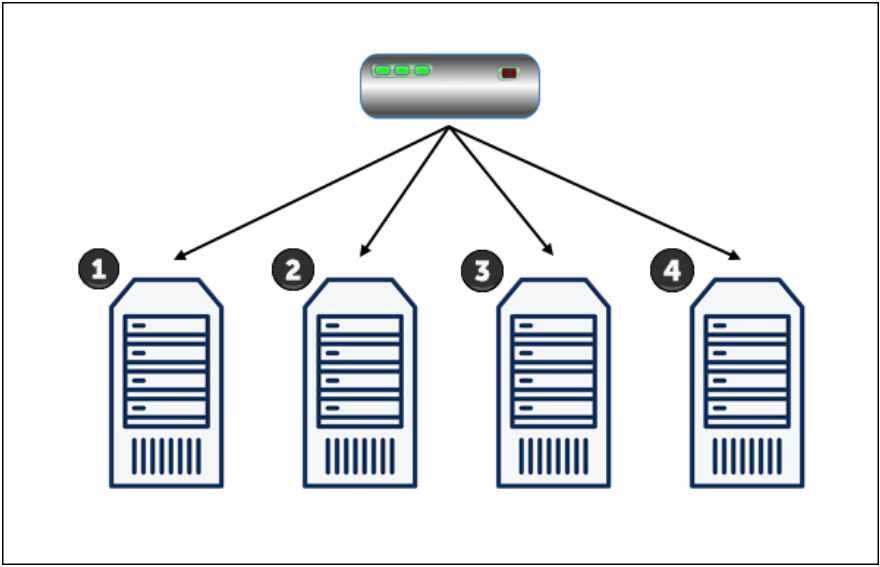


Рисунок 1 - Round-robin

1. Least connections: этот алгоритм отправляет запросы на сервер с наименьшим количеством активных соединений. Этот алгоритм обеспечивает более равномерное распределение нагрузки между серверами и учитывает текущую нагрузку на каждом сервере.

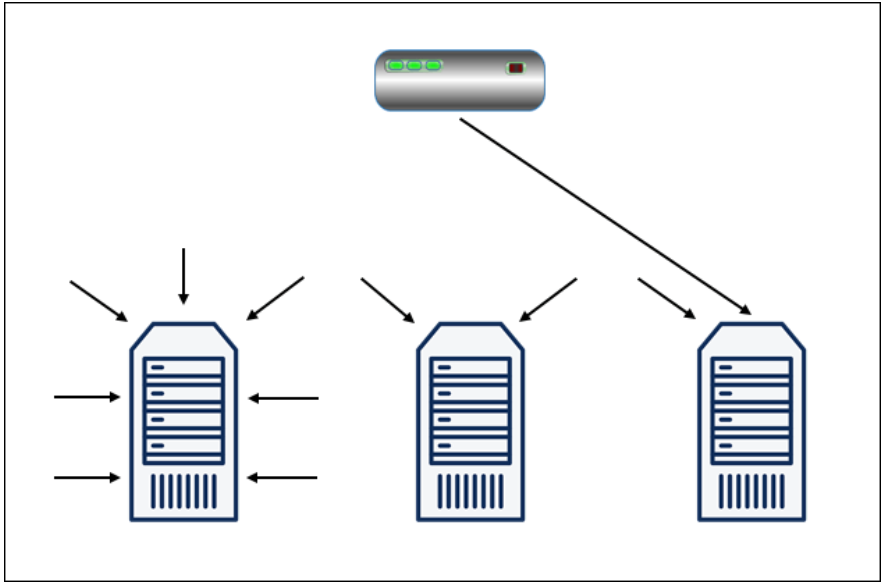


Рисунок 2 - Least connections

1. IP hash: этот алгоритм использует хеш-функцию для распределения запросов между серверами на основе IP-адреса клиента. Этот алгоритм обеспечивает постоянное направление запросов от одного и того же клиента на один и тот же сервер, что может быть полезно в случае поддержки состояний на сервере.

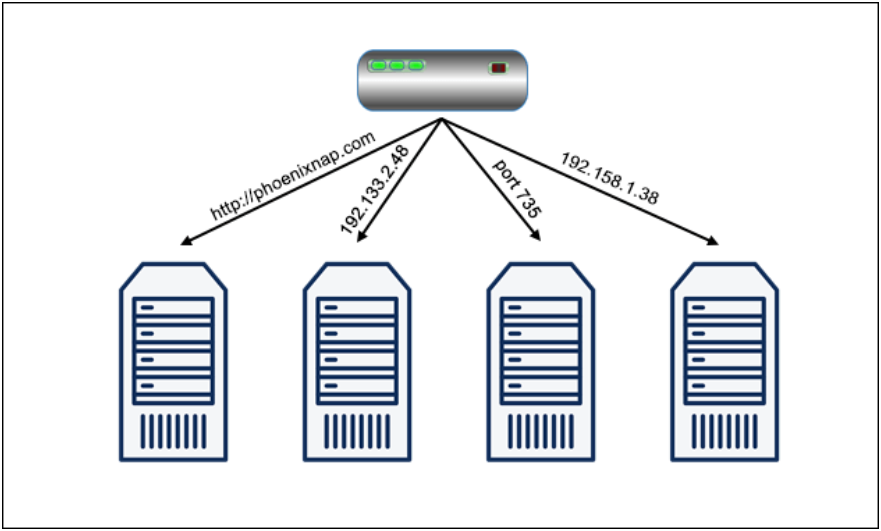


Рисунок 3 - IP hash

1. Random: этот алгоритм выбирает случайный сервер из списка доступных серверов для каждого запроса. Этот алгоритм прост в реализации, но не гарантирует равномерного распределения нагрузки между серверами.
2. Weighted round-robin: этот алгоритм распределяет запросы между серверами с учетом их весов, которые могут быть настроены вручную для каждого сервера. Этот алгоритм позволяет более точно настроить распределение нагрузки между серверами в зависимости от их производительности или других факторов.
3. Weighted least connections: этот алгоритм отправляет запросы на сервер с наименьшим количеством активных соединений, но с учетом их весов. Этот алгоритм позволяет более точно настроить распределение нагрузки между серверами в зависимости от их производительности и текущей нагрузки.

## **3.4. Рассмотрение применения балансировщика нагрузки в российских фирмах из реестра Роскомнадзора**

Рассмотрение применения балансировщика нагрузки в российских фирмах из реестра Роскомнадзора может быть полезным для анализа того, как российские компании используют эту технологию для обеспечения стабильной и эффективной работы своих веб-сервисов и приложений.

### **3.4.1. Компания NGINX**

NGINX — это крупнейшая компания, специализирующаяся на разработке программного обеспечения для веб-серверов, балансировки нагрузки, ускорения и оптимизации веб-сайтов и приложений. Компания была основана в 2002 году и на данный момент является одной из самых популярных компаний в этой области.

Одним из наиболее популярных продуктов NGINX является балансировщик нагрузки NGINX Plus. Этот продукт обеспечивает распределение трафика между несколькими серверами для обеспечения высокой доступности и производительности веб-приложений. Балансировщик нагрузки NGINX Plus имеет широкий спектр возможностей, включая:

1. Алгоритмы балансировки: NGINX Plus поддерживает различные алгоритмы балансировки, включая round-robin, least-connected, IP hash и другие.
2. Кластеризация: NGINX Plus поддерживает кластеризацию для обеспечения высокой доступности и отказоустойчивости.
3. Мониторинг: NGINX Plus включает инструменты мониторинга, которые позволяют отслеживать состояние балансировщика нагрузки и серверов, к которым он обращается.
4. Управление: NGINX Plus включает web-интерфейс управления, который позволяет настраивать и контролировать работу балансировщика нагрузки.

Балансировщик нагрузки NGINX Plus применяется в таких крупных компаниях, как Dropbox, Airbnb, Netflix, WordPress и многих других, а также используются в таких крупных российских компаниях, как Яндекс, Mail.Ru Group, Ozon и другие. Компания NGINX также предоставляет бесплатную версию своего балансировщика нагрузки - NGINX Open Source, которая также широко используется в различных веб-приложениях.

### **3.4.2. Компания Ядро (Core)**

Компания Ядро (Core) — это российская компания, которая занимается разработкой программного обеспечения для защиты данных и управления сетевой инфраструктурой. Одним из продуктов компании является балансировщик нагрузки CoreBalancer.

CoreBalancer — это программное обеспечение, которое позволяет распределять нагрузку между несколькими серверами и обеспечивать высокую доступность и производительность веб-приложений. Он имеет следующие особенности:

1. Алгоритмы балансировки: CoreBalancer поддерживает различные алгоритмы балансировки, такие как round-robin, least-connections, IP-hash и др.
2. Кластеризация: CoreBalancer поддерживает кластеризацию, которая обеспечивает отказоустойчивость и высокую доступность.
3. Мониторинг: CoreBalancer включает инструменты мониторинга, которые позволяют отслеживать состояние балансировщика нагрузки и серверов, к которым он обращается.
4. Безопасность: CoreBalancer включает функции защиты от DDoS-атак и других угроз безопасности.

CoreBalancer используется в таких крупных компаниях, как Яндекс, Росбанк, Ростелеком и другие. Компания Ядро также предоставляет услуги по настройке и поддержке балансировщика нагрузки для своих клиентов.

### **3.4.3. Компания СКАТ**

Компания СКАТ (SKAT) - это российская компания, которая занимается разработкой и поставкой инфраструктурных решений для комплексной автоматизации предприятий и организаций. Одним из продуктов компании является балансировщик нагрузки SKAT Load Balancer.

SKAT Load Balancer - это программное обеспечение, которое позволяет распределять нагрузку между несколькими серверами и обеспечивать высокую доступность и производительность веб-приложений. Он имеет следующие особенности:

1. Алгоритмы балансировки: SKAT Load Balancer поддерживает различные алгоритмы балансировки, такие как round-robin, least-connections, IP-hash и др.
2. Кластеризация: SKAT Load Balancer поддерживает кластеризацию, которая обеспечивает отказоустойчивость и высокую доступность.
3. Мониторинг: SKAT Load Balancer включает инструменты мониторинга, которые позволяют отслеживать состояние балансировщика нагрузки и серверов, к которым он обращается.
4. Безопасность: SKAT Load Balancer включает функции защиты от DDoS-атак и других угроз безопасности.

SKAT Load Balancer используется в таких компаниях, как МТС, Роснефть, Россети и другие. Компания СКАТ также предоставляет услуги по настройке и поддержке балансировщика нагрузки для своих клиентов.

# **4. Блок схема балансировщика нагрузки**

В качестве алгоритма балансировки нагрузки был выбран метод Round Robin, для реализации программной части балансировщика нагрузки необходимо составить блок схему алгоритма балансировщика нагрузки работающего на прикладном уровне (рисунок 4).

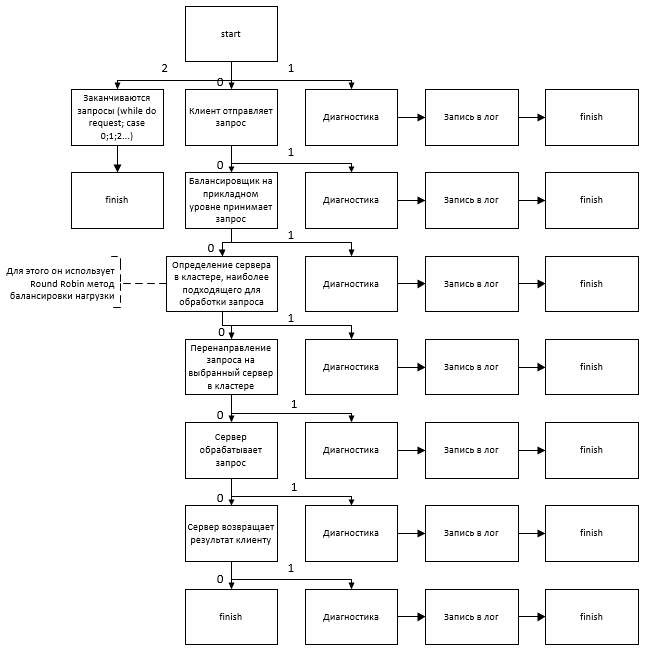


Рисунок 4 – Блок схема алгоритма работы балансировщика

Балансировщик нагрузки на прикладном уровне работает следующим образом:

1) Клиент отправляет запрос на серверный кластер.

2) Балансировщик нагрузки на прикладном уровне принимает запрос и определяет, какой сервер в кластере, наиболее подходящий для обработки запроса. Для этого он использует различные методы балансировки нагрузки, такие как круговой, случайный, взвешенный и др.

3) Балансировщик нагрузки перенаправляет запрос на выбранный сервер в кластере.

4) Сервер обрабатывает запрос и возвращает результат клиенту через балансировщик нагрузки на прикладном уровне.

Если один из серверов в кластере не отвечает или работает медленно, балансировщик нагрузки на прикладном уровне может автоматически перенаправлять запросы на другие серверы, чтобы обеспечить высокую доступность и быстродействие приложения.

Балансировщик нагрузки на прикладном уровне может также выполнять дополнительные функции, такие как шифрование и дешифрование трафика, кеширование данных и мониторинг работы серверов в кластере.

Таким образом, балансировщик нагрузки на прикладном уровне является важным инструментом для обеспечения высокой доступности и быстродействия приложений, особенно в случае, когда несколько серверов используются для обработки запросов от клиентов.

Коды возврата (также известные как HTTP-статусы) являются частью протокола HTTP и используются веб-сервером для указания результата обработки запроса. Балансировщик нагрузки использует эти коды возврата для принятия решения о перенаправлении запроса на определенный сервер:

* 200 OK: Запрос успешно выполнен, и сервер возвращает запрошенные данные.
* 301 Moved Permanently: Ресурс был перемещен на другой URL. Балансировщик нагрузки может использовать новый URL для перенаправления запроса на соответствующий сервер.
* 302 Found: Ресурс временно перемещен на другой URL. Балансировщик нагрузки может использовать новый URL для перенаправления запроса на соответствующий сервер.
* 404 Not Found: Запрошенный ресурс не найден на сервере.
* 500 Internal Server Error: Произошла ошибка на стороне сервера.
* 503 Service Unavailable: Сервер временно не может обработать запрос из-за высокой нагрузки или технических проблем. Балансировщик нагрузки может перенаправить запрос на другой сервер.
* 504 Gateway Timeout: Сервер не получил ответ от другого сервера, который должен обрабатывать запрос. Балансировщик нагрузки может попробовать перенаправить запрос на другой сервер.

# **5. Разработка кода для балансировщика нагрузки**

Код для балансировщика нагрузки на прикладном уровне с использованием алгоритма Round Robin на языке программирования C# представлен далее:

Глобальные переменные:

//ip-адрес

string ipaddr;

//порт

int port;

//количество запросов

int valreq;

//содержимое запроса

string contrequest;

//сообщение клиента

string mesclient;

//сообщение сервера

string messerver;

//номер сервера

int numserver;

Функция main():

static void Main(string[] args)

{

//запуск серверов

StartServer("127.0.0.1", 8000);

StartServer("127.0.0.1", 8001);

StartServer("127.0.0.1", 8002);

//отправка запроса клиентом к серверу

int valreq = SendClientRequest("127.0.0.1", 8000);

int zapros = 2;

switch (zapros)

{

case 0:

Console.WriteLine("Ошибка отправки запроса! Диагностика...");

break;

case 1:

Console.WriteLine("Запрос получен");

break;

case 2:

Console.WriteLine("Количество запросов: ", valreq);

break;

}

//Отправка запроса на балансировщик

zapros = 1;

switch (zapros)

{

case 0:

Console.WriteLine("Ошибка на этапе балансировки! Диагностика...");

break;

case 1:

Console.WriteLine("Балансировка нагрузки");

BalanceRequest(mesclient);

break;

}

//Отправка ответа сервера клиенту

zapros = 1;

switch (zapros)

{

case 0:

Console.WriteLine("Ошибка обработки запроса! Диагностика...");

break;

case 1:

Console.WriteLine("Возвращение результата клиенту");

SendResponse(messerver);

break;

}

}

Функция запуска сервера:

public static void StartServer(string ipAddress, int port)

{

IPAddress ip = IPAddress.Parse(ipAddress);

IPEndPoint endPoint = new IPEndPoint(ip, port);

TcpListener listener = new TcpListener(endPoint);

try

{

listener.Start();

Console.WriteLine($"Сервер запущен на {ip}:{port}");

while (true)

{

TcpClient client = listener.AcceptTcpClient();

Console.WriteLine($"Новое подключение от {client.Client.RemoteEndPoint}");

// обработка запросов от клиента

}

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка при запуске сервера: {ex.Message}");

}

finally

{

listener.Stop();

}

}

Функция отправки запроса клиентом к серверу:

public static int SendClientRequest(string ipAddress, int port)

{

// Создаем новый TCP клиент и подключаемся к серверу

TcpClient client = new TcpClient(ipAddress, port);

// Отправляем запрос серверу

byte[] data = Encoding.ASCII.GetBytes("Client request");

NetworkStream stream = client.GetStream();

stream.Write(data, 0, data.Length);

// Читаем ответ от сервера

data = new byte[256];

int bytes = stream.Read(data, 0, data.Length);

string responseData = Encoding.ASCII.GetString(data, 0, bytes);

// Получаем количество запросов от сервера

int requestCount = int.Parse(responseData);

// Закрываем соединение

stream.Close();

client.Close();

// Возвращаем количество запросов

return requestCount;

}

Функция балансировки нагрузки:

public static void BalanceRequest(TcpClient client)

{

Console.WriteLine("New client connected!");

// Выбор сервера

int selectedServer = RoundRobin();

// Подключение к выбранному серверу

TcpClient server = serverList[selectedServer];

// Отправка запроса на сервер

NetworkStream serverStream = server.GetStream();

byte[] buffer = new byte[client.ReceiveBufferSize];

int bytesRead = client.GetStream().Read(buffer, 0, client.ReceiveBufferSize);

serverStream.Write(buffer, 0, bytesRead);

serverStream.Flush();

// Увеличение счетчика запросов от клиента к выбранному серверу

clientRequestCounts[selectedServer]++;

// Отправка запроса серверу о количестве запросов от клиента

byte[] countBuffer = BitConverter.GetBytes(clientRequestCounts[selectedServer]);

serverStream.Write(countBuffer, 0, countBuffer.Length);

serverStream.Flush();

}

Функция возвращения результата клиенту:

public static void SendResponse(HttpListenerContext context, string responseString)

{

byte[] buffer = System.Text.Encoding.UTF8.GetBytes(responseString);

context.Response.ContentLength64 = buffer.Length;

System.IO.Stream output = context.Response.OutputStream;

output.Write(buffer, 0, buffer.Length);

output.Close();

}

# **6. Вывод**

В данной работе были изучены различные алгоритмы балансировки нагрузки и их применение в информационных системах, разработано приложение балансировщика на языке программирования C# в котором реализован алгоритм round robin, а также были рассмотрены примеры применения балансировки нагрузки в российских фирмах из реестра Роскомнадзора.

# **Список литературы**

1. Пересадов В.В., Пересадова Е.В. Балансировка нагрузки в компьютерных сетях. М.: БИНОМ, 2013.
2. Kozlov S., Vasyutin E., Shnayderman I. Load balancing in computer networks // 2016 International Conference on Engineering and Telecommunication (EnT). IEEE, 2016. P. 312-315.
3. Nayar R., Pani S.K. A comparative study of load balancing algorithms in cloud computing // 2017 IEEE 7th International Advance Computing Conference (IACC). IEEE, 2017. P. 424-428.
4. Чжао Ж. Разработка программного обеспечения для балансировки нагрузки на примере Apache Tomcat // Молодой ученый. — 2018. — № 12. — С. 118-121.
5. Кирюшкин А.А., Мухина Т.А. Анализ и выбор алгоритмов балансировки нагрузки // Материалы III Международной научно-технической конференции «Наука. Техника. Образование» (НТО-2018). — 2018. — С. 283-285.
6. Реджепова Д., Каримов Х. Анализ и выбор алгоритмов балансировки нагрузки в компьютерных сетях // Вестник КазНУ. Серия информатика. — 2018. — Т. 13. — № 4. — С. 9-14.
7. Официальный сайт NGINX. URL: https://nginx.org/en/ (дата обращения: 10.04.2023).
8. Официальный сайт Ядро. URL: https://j-core.org/ (дата обращения: 10.04.2023).
9. Официальный сайт СКАТ. URL: https://skat.pro/ (дата обращения: 10.04.2023).