УДК 004.45+004.75

# МОНИТОРИНГ РЕСУРСОВ ВЫЧИСЛИТЕЛЬНОЙ СЕТИ ДЛЯ ЭФФЕКТИВНОГО РАСПРЕДЕЛЕНИЯ НАГРУЗКИ

*Северин К.М., студент гр. 951004*

*Белорусский государственный университет информатики и радиоэлектроники, г. Минск, Республика Беларусь*

*Парамонов А.И. – канд. техн. наук, доцент*

**Аннотация.** Выполнен анализ проблемы эффективности распределения нагрузки в вычислительной сети. Рассмотрены алгоритмы балансировки. Предложена модификация алгоритма балансировки нагрузки на основе информации о текущей загруженности вычислительных узлов. Предложено программно-алгоритмическое обеспечение для мониторинга ресурсов доступных узлов. Описана архитектура программного комплекса и особенности программной реализации.

**Ключевые слова.** Мониторинг, ресурсы компьютера, сетевые ресурсы, вычислительная сеть, сервер, распределение нагрузки, алгоритм, балансировка, счетчик производительности, PDH, программное обеспечение, стресс-тест, Round Robin, Windows, Win32 API, C++11.

***Введение.*** В связи с массовым распространением распределенных вычислительных систем стала актуальной проблема их эффективного использования. Одним из аспектов данной проблемы является эффективное планирование и распределение задач внутри распределенных вычислительных систем с целью оптимизации использования ресурсов и сокращения времени вычисления. Весьма часто возникает ситуация, при которой часть вычислительных ресурсов простаивает, в то время как другая часть ресурсов перегружена и в очереди имеется большое количество ожидающих своего исполнения задач. Для оптимизации использования ресурсов, сокращения времени обслуживания запросов, горизонтального масштабирования (динамическое добавление или удаление устройств), а также обеспечения отказоустойчивости (резервирования) [1] применяется метод равномерного распределения заданий между несколькими сетевыми устройствами (например, серверами) называемый балансировка нагрузки, или выравнивание нагрузки (Load Balancing).

***Основная часть.*** Проблема балансировки вычислительной нагрузки распределённой системы возникает по различным причинам, в числе которых:

— структура вычислительного комплекса (например, кластера) неоднородна, т.е. разные вычислительные узлы обладают разной производительностью;

— структура межузлового взаимодействия неоднородна, т.к. линии связи, соединяющие узлы, могут иметь различные характеристики пропускной способности;

— структура распределенного приложения неоднородна, различные логические процессы требуют различных вычислительных мощностей.

При появлении новых заданий программное обеспечение, которое отвечает за балансировку, должно принять решение о том, на каком вычислительном узле следует выполнять вычисления, связанные с этим новым заданием. Кроме того, балансировка предполагает перенос части вычислений с наиболее загруженных вычислительных узлов на менее загруженные узлы. При выполнении задач процессоры обмениваются между собой коммуникационными сообщениями. В случае низких затрат на коммуникацию, некоторые процессоры могут простаивать, в то время как остальные будут перегружены. Также будут нецелесообразны большие затраты на коммуникацию. Следовательно, стратегия балансировки должна быть такой, чтобы вычислительные узлы были загружены достаточно равномерно, но и коммуникационная среда не должна быть перегружена.

В большинстве случаев балансировка используется для достижения следующих целей [2]:

— справедливость – нужно гарантировать, чтобы на обработку каждого запроса выделялись системные ресурсы и не допустить возникновения ситуаций, когда один запрос обрабатывается, а все остальные ждут своей очереди;

— эффективность – все серверы, которые обрабатывают запросы, должны быть заняты на 100%; желательно не допускать ситуации, когда один из серверов простаивает в ожидании запросов на обработку (в реальной практике эта цель достигается далеко не всегда);

— сокращение времени выполнения запроса – нужно обеспечить минимальное время между началом обработки запроса (или его постановкой в очередь на обработку) и его завершения;

— сокращение времени отклика – нужно минимизировать время ответа на запрос.

Важно чётко понимать, в каких ситуациях и при каких нагрузках алгоритм будет эффективным для решения поставленных задач, и при увеличении нагрузки алгоритм должен сохранять работоспособность. Поэтому очень желательно, чтобы алгоритм балансировки обладал свойствами: предсказуемость; равномерная загрузка ресурсов системы и масштабируемость.

На сегодня известно и применяется достаточно много алгоритмов балансировки. Одним из самых популярных и простых в реализации является Round Robin. Вместе с тем этот алгоритм имеет и целый ряд существенных недостатков. Например, чтобы распределение нагрузки по этому алгоритму отвечало упомянутым выше критериями справедливости и эффективности, нужно, чтобы у каждого сервера был в наличии условно одинаковый набор ресурсов. При выполнении всех операций также должно быть задействовано одинаковое количество ресурсов. В реальной практике эти условия в большинстве случаев оказываются практически невыполнимыми. Его модернизацией является Weighted Round Robin. Суть усовершенствований заключается в том, что каждому серверу присваивается весовой коэффициент в соответствии с его производительностью и мощностью. Это помогает распределять нагрузку более гибко: серверы с большим весом обрабатывают больше запросов. Однако всех проблем с отказоустойчивостью это отнюдь не решает.

При проектировании решения задачи балансировки будем предполагать, что все задачи одинаковы, то есть каждая задача, которая подается на один вычислительный узел, потребует от него одинаковое количество системных ресурсов.

В основе предложенного подхода к балансировке положена следующая гипотеза: задачу должен получать вычислительный узел с наименьшей загруженностью на центральный процессор данный момент. Таким образом, предполагается, что нагрузка будет распределена равномерно по всем узлам. Общую схему алгоритма можно увидеть на рисунке 1.

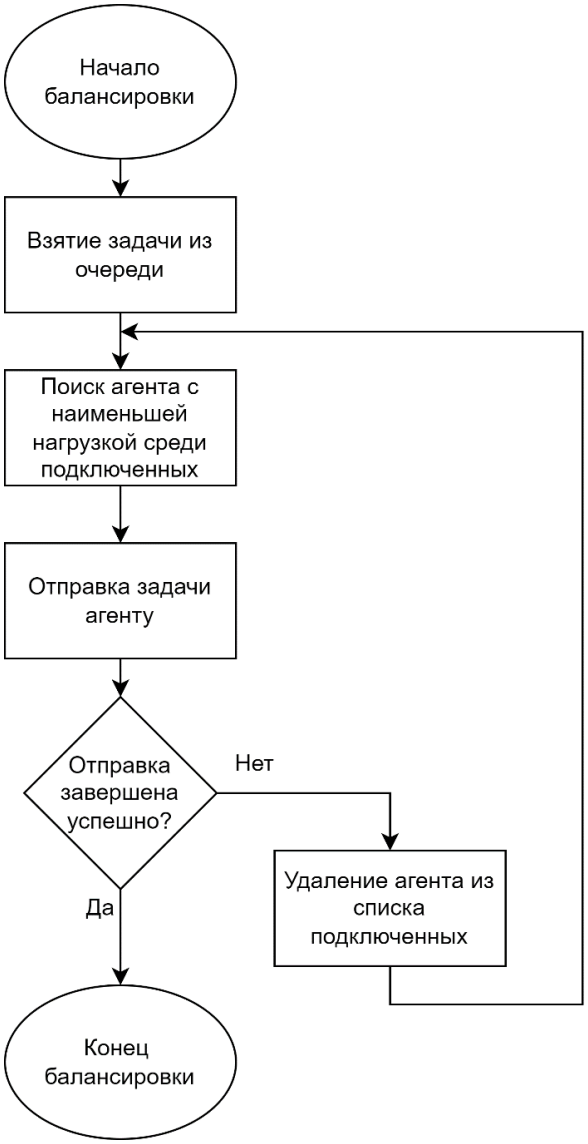


Рисунок 1 – Общая схема алгоритма балансировки

Для того чтобы минимизировать затраты на общение между узлами и при этом получить достаточно удобный для взаимодействия уровень абстракции были использованы стандартные сокеты операционной системы Windows. В качестве протокола транспортного уровня был взят TCP, так как он осуществляет повторный запрос данных в случае потери данных и устраняет дублирование при получении двух копий одного пакета, гарантируя тем самым целостность передаваемых данных и уведомление отправителя о результатах передачи [3].

В операционной системе Windows существует 3 способа получить сведений об используемых ресурсах:

1. Основанный на применении API, который становится доступен после подключения заголовочного файла windows.h [4];
2. Счетчики производительности PDH (performance data helpers) [5];
3. Инструментарий управления Windows (WMI).

В качестве способа получения данных о ресурсах выбраны счетчики производительности (использование библиотеки pdh.h). Этот API предоставляет единый интерфейс для получения основных счетчиков производительности, что дает возможность быстрого расширения количества поддерживаемых метрик и усложнения алгоритма балансировки.

Чтобы на «засорять» канал связи между клиентом и сервером, клиент отправляет информацию о нагрузке с периодом в 1 секунду. Счетчики производительности также обновляются раз в секунду.

При тестировании полученной системы одинаковыми задачами наблюдалось равномерное распределение нагрузки между всеми подключенными узлами. При тестировании различными по сложности задачами наблюдались различия в нагрузке на вычислительные узлы, обусловленные тем, что задача считается неделимой. При стрессовом тестировании был выявлен главный недостаток балансировки на основе нагрузки на узлы: когда задач становиться слишком много балансировщик может успеть отправить несколько задач одному вычислительному узлу из-за того, что информация об актуальной нагрузке на вычислительный узел еще не пришла. Такую ситуацию наглядно можно видеть на рисунке 2.

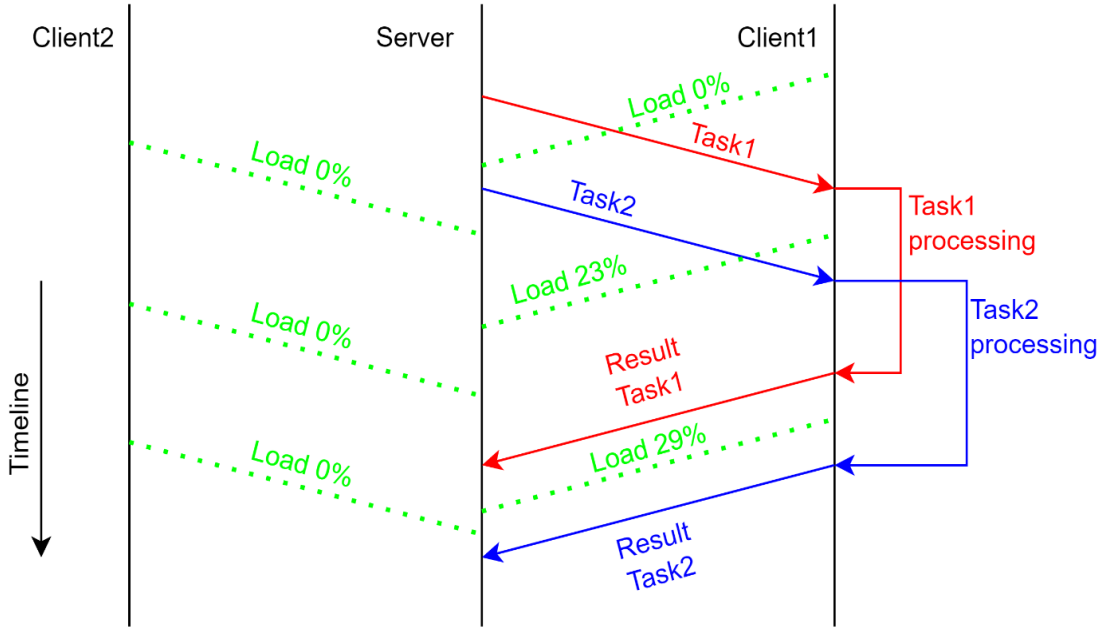


Рисунок 2 - Схема обмена сообщениями сервера и двух клиентов

Программный комплекс для мониторинга ресурсов вычислительной сети разработан на языке программирования C++, с применением стандарта языка C++11. Использовалась стандартная библиотека шаблонов, как удобные контейнеры для хранения данных, а также такие классы, как std::thread и std::mutex для организации многопоточности.

Пользовательский интерфейс программных модулей построен с использование средств Win32 API. Данное API было выбрано потому, как оно предоставляет возможности разработки первого уровня, не зависящей от управляемой среды выполнения, такой как .NET или WinRT. Благодаря этому Win32 является оптимальной платформой для приложений, которым требуется самый высокий уровень производительности и прямой доступ к системному оборудованию. Также были использованы такие системные объекты как сокеты.

Программное средство сервера состоит из трех модулей:

1. Балансировщик, который распределяет задачи между клиентами;
2. Менеджер ресурсов, который собирает сведения от клиентов (узлов) об их загруженности и обновляет интерфейс;
3. Сервер, слушающий и обрабатывающий входящие соединения.

Программное средство клиента также состоит из трех модулей:

1. «Работник», который выполняет полезную работу (запускает процесс для выполнение поставленной задачи, следит за ее выполнением);
2. «Сборщик информации», который собирает показания с датчиков производительности pdh.h;
3. «Клиент», слушающий канал связи и добавляющий задачу в очередь работнику.

Все модули работают параллельно, каждый в своем потоке. Для синхронизации доступа к общим данным используется мьютекс. Коммуникации между клиентом и сервером происходит с использованием сокетов, которые предоставляет Win API. В процессе работы приложения устанавливается постоянное соединение, по которому передаются сообщения между сервером и клиентом. Общая структура сообщения состоит из двух полей: размер текстового сообщения (4 байта) и текстовое сообщение (n байт). Текстовое сообщение бывает трех типов: задача, результат выполнения задачи и данные датчиков. Тип текстового сообщения определяется по первому ключевым словам «TASK», «RESULT», «COUNTER». Далее идут необходимые данные в виде текста.

На рисунке 3 представлена структура программного комплекса.

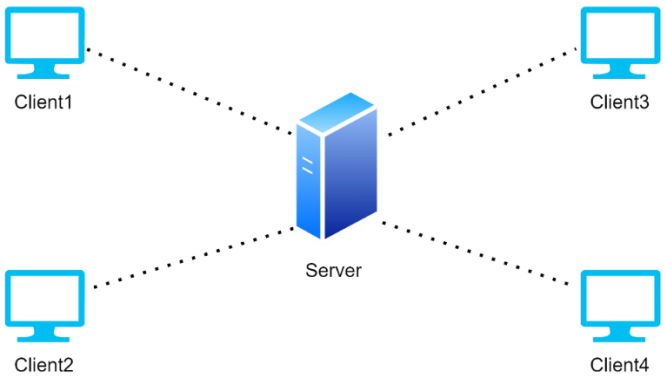


Рисунок 3 – Архитектура построенной модели взаимодействия программных компонент

В ходе проведения тестирования и экспериментов использовалось 5 компьютеров разных мощностей под управлением Windows 10, один из которых является сервером, остальные – клиенты. Для нагрузки использовалась рекурсивная реализация нахождения члена последовательности Фибоначчи ввиду простоты реализации и легкого контролирования сложности задачи заданием номера члена.

***Заключение.*** В ходе эксперимента было установлено, что разработанный программный комплекс справляется с поставленной задачей и при стабильной работе всех участков повышает производительность вычислительной сети. Однако в ходе эксплуатации обнаружилось, что возникают случаи несвоевременности подачи данных о текущей нагрузке от клиентов, по причине чего очередная задача может быть распределена с учетом устаревших данных.

Данный проект может применяться в различных системах распределенного выполнения задач, таких как обработка серверных запросов, компьютерные вычисления, создание сложных математических моделей. Кроме того, проектное решение может использоваться для проверки производительности различных алгоритмов балансировки.

***Список использованных источников:***

*1. High Availability [Электронный ресурс] – режим доступа: http://www.linuxvirtualserver.org/HighAvailability.html*

*2. Dynamic Load Balancing and Scheduling [Электронный ресурс] – режим доступа: https://web.archive.org/web/20051103023103/http://wwwcs.uni-paderborn.de/cs/ag-monien/RESEARCH/LOADBAL/*

*3. Программная спецификация протоколов коммуникаций [Электронный ресурс] – режим доступа: https://rfc.com.ru/*

*4. psapi.h header [Электронный ресурс] - Режим доступа: https://docs.microsoft.com/en-us/windows/win32/api/psapi/*

*5. Using the PDH Functions to Consume Counter Data [Электронный ресурс] - Режим доступа: https://docs.microsoft.com/en-us/windows/win32/perfctrs/using-the-pdh-functions-to-consume-counter-data*

UDC 004.45+004.75

# MONITORING OF COMPUTER NETWORK RESOURCES

*Severin K.M.*

*Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus*

*Paramonov A.I. – PhD in Computer Science*

**Annotation.** An analysis of the efficiency of load distribution problem in a computer network is carried out. Balancing algorithms are considered. A load balancing algorithm modification based on information about the current workload of computing nodes is proposed. Software and algorithmic solution for monitoring the resources of available nodes is proposed. The architecture of the software package and software implementation features are described.

**Keywords.** Monitoring, computer resources, network resources, computer network, server, load distribution, algorithm, balancing, performance counters, performance data helpers, software, stress test, Round Robin, Windows, Win32 API, C++11.