Министерство образования Республики Беларусь
Учреждение образования
Белорусский государственный университет
информатики и радиоэлектроники

УДК 004.428

Лагута
Алексей Сергеевич

Высокопроизводительный веб-сервер для получения, хранения и эффективной обработки больших объемов данных

АВТОРЕФЕРАТ

на соискание степени магистра

по специальности 1-40 80 04 Информатика и технологии программирования

Научный руководитель
Хмелев А. Г.
д. э. н., профессор

Минск 2021

КРАТКОЕ ВВЕДЕНИЕ

В связи с огромным скачком в сфере информационных технологий и популяризации использования портативных устройств в повседневной жизни, с каждым днем все больше различных сервисов и услуг переносится во всемирную сеть – Интернет. Согласно статистике, на 2019 год пользователями сети Интернет являются более 50 % всего населения нашей планеты, приблизительно 4 миллиарда человек, в Европе данный показатель доходит до 80 процентов. Любой пользователь всемирной сети является потребителем определенных услуг, то есть так называемым «клиентом». Например, когда проверяет электронную почту, общается посредством систем мгновенного обмена сообщениями, скачивает или загружает необходимую информацию или заходит на свой любимый сайт. Для всех этих действий пользователю необходимо получить информацию по сети, а значит запросить её с другой вычислительной машины (или иногда с той же самой) посредством сетевых протоколов. Здесь и вступает в дело так называемый «сервер».

Сервер в широком смысле – это любое программное обеспечение, которое ожидает запросы от клиентских программ, а затем предоставляет им свои ресурсы в виде данных или сервисных функций.

Поскольку число клиентов с каждым годом резко увеличивается и одна сервер-программа может выполнять запросы сразу от многих программ-клиентов, то очень часто данную сервер-программу размещают на специально выделенной для этой цели вычислительной машине, которую настраивают специальным образом для эффективного взаимодействия с другими программами-серверами. Из-за особой роли и специфики данной машины, её тоже часто называют сервером.

В связи с постоянным развитием сети Интернет, производительность сервера является одной из главных его характеристик. Пользователи не любят ждать, поэтому скорость ответа сервера, особенно при высоких нагрузках, является критичной во многих сферах.

Соответственно, высокопроизводительные серверы являются востребованным продуктом на рынке, а проблема увеличения производительности серверов была и остается одной из главных проблем, связанных с постоянным развитием сети Интернет.

Диссертационная работа посвящена проектированию и разработке высокопроизводительного сервера для получения, хранения и обработки информации. Разработанное решение может быть использовано в различных сферах, особенно тех, где время обработки данных и ответа сервера на запрос играет важную роль (к примеру, торговые платформы).

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Цель и задачи исследования

Целью диссертационной работы является проектирование архитектуры и разработка высокопроизводительного сервера для получения, хранения и эффективной обработки информации.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Проанализировать веб-протоколы прикладного уровня, выявить их преимущества и недостатки, выбрать наиболее подходящие для реализации цели.

2. Для каждого из выбранных протоколов разработать собственную реализацию с применением оптимизаций при разработке для увеличения производительности системы.

3. Предложить и реализовать архитектуру высокопроизводительного веб-сервера для получения, обработки и хранения информации.

4. Разработать сценарии тестирования и экспериментально протестировать производительность полученных решений. Сравнить результаты с аналогами, сделать соответствующие выводы об успешности реализации проекта.

Объектом исследования являются высоконагруженные системы клиент-серверной архитектуры.

Предметом исследования являются характеристики производительности (пропускная способность, задержка сообщения) высоконагруженных серверов при обработке информации.

Основной *гипотезой*, положенной в основу диссертационной работы, является возможность применения эффективной архитектуры и оптимизаций для достижения более высоких показателей производительности.

Личный вклад соискателя

Результаты, приведенные в диссертации, получены соискателем лично. Предложена и реализована архитектура высокопроизводительного сервера, а также собственные реализации четырех веб-протоколов.

Публикации результатов диссертации

По теме диссертации опубликовано 2 печатных работы, из них 2 статьи в научном журнале.

Структура и объем диссертации

Диссертация состоит из общей характеристики работы, введения, четырех глав, заключения, списка использованных источников, списка публикаций автора и приложений. В первой главе представлен анализ протоколов прикладного уровня, в частности веб-протоколов, выявлены их преимущества, недостатки, различия. Выбраны веб-протоколы для собственной реализации. Вторая глава посвящена обзору используемых технологий для проектирования и разработки проекта. Дано обоснование выбора данных технологий, а также приведены показатели производительности платформы, на которой было разработано решение. В третьей главе предложена архитектура системы, а также подробно описана разработка всех протоколов и сущностей и их оптимизаций. Также описана и показана разработка тестовых примеров. В четвертой главе предложены сценарии тестирования как функциональности, так и производительности. Приведены результаты тестирования, выполнено сравнение показателей с аналогами.

Общий объем работы составляет 115 страниц, из которых основного текста – 75 страниц, 27 рисунков на 22 страницах, 7 таблиц на 6 страницах, список использованных источников из 36 наименований на 3 страницах и 2 приложения на 30 страницах.

ОСНОВНОЕ СОДЕРЖАНИЕ

В **первой главе** проведен анализ протоколов прикладного уровня, в частности веб-протоколов. Выявлены преимущества и недостатки приведенных протоколов, а также различия между ними. На основе анализа выбраны протоколы для собственной реализации.

Задача протоколов прикладного уровня – предоставить данные пользователю в понятном для него виде, при этом не требуется обеспечивать маршрутизацию или гарантировать доставку данных (это задача протоколов более нижних уровней). Данные протоколы могут выполнять разные функции: предоставление удаленного доступа, передача текста и файлов, поддержка сети, и др.

Наиболее популярными веб-протоколами передачи данных являются HTTP и HTTPS. Данные протоколы схожи, а главное их отличие – протокол HTTPS защищен, а HTTP – нет.

Протокол WebSocket позволяет взаимодействовать между клиентом и веб-сервером с меньшими издержками, чем полудуплексные альтернативы, облегчая передачу данных в режиме реального времени с сервера и на сервер. Это стало возможным благодаря предоставлению стандартизированным способом отправки контента с сервера клиенту без предварительного запроса со стороны

клиента и разрешению передачи сообщений назад и вперед при сохранении открытого соединения. Таким образом, между клиентом и сервером может происходить двусторонний непрерывный разговор.

Поскольку в рамках данного проекта поставлена задача разработки высокопроизводительного сервера, то протокол WebSocket является предпочтительным. Однако, он использует протокол HTTP в качестве транспортного протокола для достижения максимальной эффективности. Вследствие этого было принято решение реализовать четыре протокола в рамках данной работы: HTTP, HTTPS, WebSocket, WebSocket Secure.

Во **второй главе** приведен обзор используемых технологий для проектирования и разработки проекта. Дано обоснование выбора данных технологий, а также приведены показатели производительности платформы, на которой было разработано решение.

Язык программирования C# и платформа .NET широко используются для разработки высоконагруженных приложений и сервисов. Благодаря поддержке компании Microsoft, данные технологии постоянно развиваются, увеличивая показатели производительности. Новая версия платформы .NET 5.0, согласно исследованию, приведенному в работе, является наилучшим решением по производительности (времени исполнения команд и использованной памяти) среди всех платформ, написанных на языке программирования C#.

Поскольку веб-сервер, который необходимо реализовать, не имеет конкретной информации о формате и структуре тех данных, которые необходимо будет хранить (то есть это фактически абстрактное решение), то может возникнуть проблема хранения данных. Суть её в том, что по ряду причин может понадобиться смена базы данных. Среди причин можно выделить следующие:

1 Неверное определения необходимого типа базы данных на ранних этапах проектирования и разработки.

2 Изменение типов данных, необходимых для хранения приложением или сервисом.

3 Необходимость разных типов данных или схем баз данных для тестирования функциональности и развертывания приложения или сервиса.

4 При проектировании и разработке унифицированного сервиса – на данном этапе неизвестно, какие данные будут храниться конкретной реализацией, вследствие чего невозможно однозначно выбрать определенный тип базы данных.

5 Необходимость поддерживать несколько типов баз данных одновременно.

В связи с этим, при проектировании и реализации проекта необходимо использовать представленное решение с помощью технологии ADO .NET, чтобы сократить затраты на более поздних этапах разработки, а также затраты при использовании данного проекта компанией, использующей его, то есть

фактически при реализации конкретной бизнес-логики. Подробнее об этом описано во второй главе диссертации, а также в авторской статье [2].

Спецификация OpenAPI и пользовательский интерфейс Swagger помогают быстро и эффективно отобразить реализованную функциональность, создавая интерактивный веб-сайт (поддерживает протоколы HTTP и HTTPS). Данные технологии удобны в использовании как для конечного пользователя, так и для разработчиков.

Для реализации данной работы были выбраны вышеописанные технологии, поскольку они являются лидерами по производительности, что является критически важным для проекта. Кроме того, данные технологии удобны в использовании, что помогает сократить время на разработку будущей функциональности. Это дает преимущество данному проекту при его сравнении с другими аналогами.

В **третьей главе** предложена архитектура системы, а также подробно описана разработка всех протоколов и сущностей и их оптимизаций. Кроме того, описана и показана разработка тестовых примеров.

Общая модель взаимодействия сервера и клиентов представлена на рисунке 1.

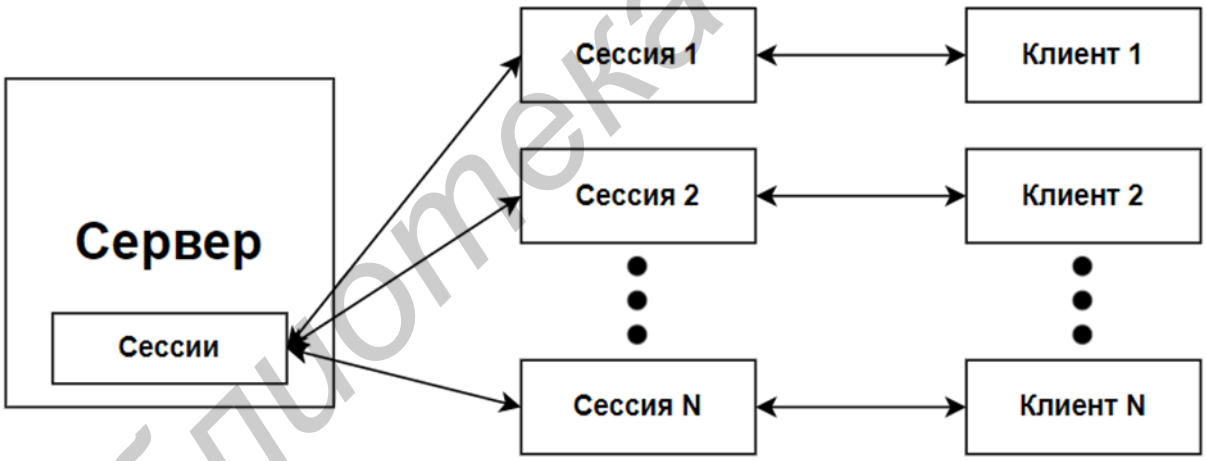

Рисунок 1 – Модель взаимодействия сервера и клиентов

Было введено понятие сессии, которая отвечает по соединение сервера с конкретным клиентом. С помощью данной концепции сервер может поддерживать большее число клиентов, а также сразу же идентифицировать их.

Предлагаемая архитектура веб-сервера для получения, хранения и обработки информации представлена на рисунке 2.

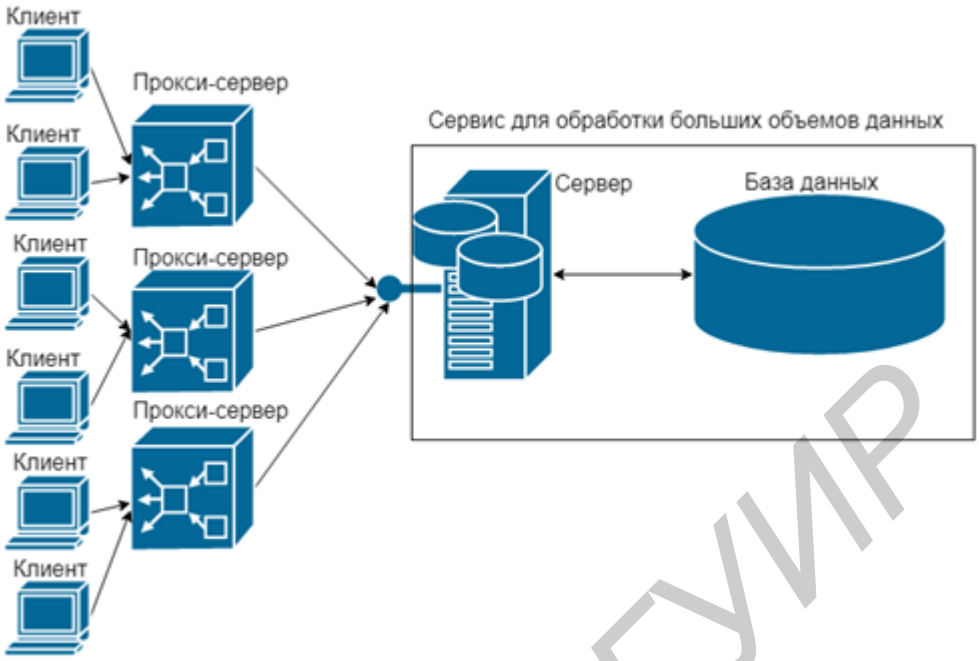

Рисунок 2 – Предлагаемая архитектура веб-сервера для получения, хранения и обработки данных

Данная архитектура учитывает все поставленные задачи и является эффективной при их решении. Каждый компонент имеет собственное назначение: клиенты подключаются к прокси-серверу (может быть один или несколько, в зависимости от масштаба и нагрузки конкретной реализации), который выполняет защитную функцию всей системы. При наличии нескольких прокси-серверов при выводе из строя одного из них по различным причинам система продолжает функционировать, клиенты смогут подключиться к любому другому из них. Это повышает стабильность работы всей системы. Среди задач прокси-сервера можно выделить аутентификацию клиентов, проверку правильности запросов клиента. После проверок данного сервера информация считается корректной и отправляется на основной сервер для обработки. Этот сервер отвечает за бизнес-логику, проверяет соответствие бизнес-информации (например, хватает ли средств на счету клиента для заказа), составляет ответ сервера на запрос клиента и отправляет его обратно прокси-серверу, который в свою очередь отправить его обратно клиенту. Сервер так же работает с подключенной к нему базой данных – читает и записывает информацию. Таким образом, каждый компонент имеет собственные задачи, а вместе они составляют сбалансированную систему.

Стоит отметить, что прокси-сервер не обязательно должен быть выражен отдельной машиной или процессом, его можно совместить с основным сервером, разделив логику на разные методы, что сделано в данной работе. Однако, переход к прокси-серверам как к отдельным сервисам не затруднителен и может быть реализован позднее.

Были написаны собственные реализации всех четырех протоколов: HTTP, HTTPS, WebSocket, WebSocket Secure. При их реализации были применены некоторые оптимизации, описанные в работе. К примеру, были реализованы дополнительные сущности HTTP-запроса и HTTP-ответа, которые используются HTTP и HTTPS протоколами. Разработанные классы представляют собой удобный способ создания запросов и ответов, необходимых для реализации HTTP протокола. Также благодаря использованию кэша внутри данных сущностей процесс их создания и обработки занимает меньше времени, что увеличивает производительность всего решения. Для WebSocket и WebSocket Secure протоколов были реализованы вспомогательные методы – переходы между протоколами и работа с фреймами.

Также было реализовано хранилище с помощью технологии ADO.NET – были подключены в качестве примера базы данных SQLite, MySQL, PostgreSQL, SQLServer, переход между которыми осуществляется лишь изменением строки подключения.

Для тестирования функциональности, а также в качества примера создания специфичных собственных серверов и клиентов были разработаны примеры для каждого из поддерживаемых протоколов. Пользовательский интерфейс примера HTTP-сервера представлен на рисунке 3.

Рисунок 3 – Пользовательский интерфейс примера HTTP-сервера

В **четвертой главе** предложены сценарии тестирования как функциональности, так и производительности. Приведены результаты тестирования, выполнено сравнение показателей с аналогами.

Для HTTP и HTTPS серверов был предложен и реализован следующий сценарий (рисунок 4). Для тестирования задаются следующие параметры: количество клиентов, количество сообщений и количество секунд для

тестирования. Все клиенты подключаются к тестируемому серверу и начинают слать TRACE-запросы (сначала каждый шлет столько, сколько указано в параметре, а затем на каждый ответ новый запрос), сервер отвечает на них. После получения ответа клиент проверяет статус ответа. Если он 200, то считается, что сообщение успешно доставлено, иначе – произошла ошибка. Тестирование происходит в течение определенного времени, указанном как параметр. После этого высчитываются показатели производительности.

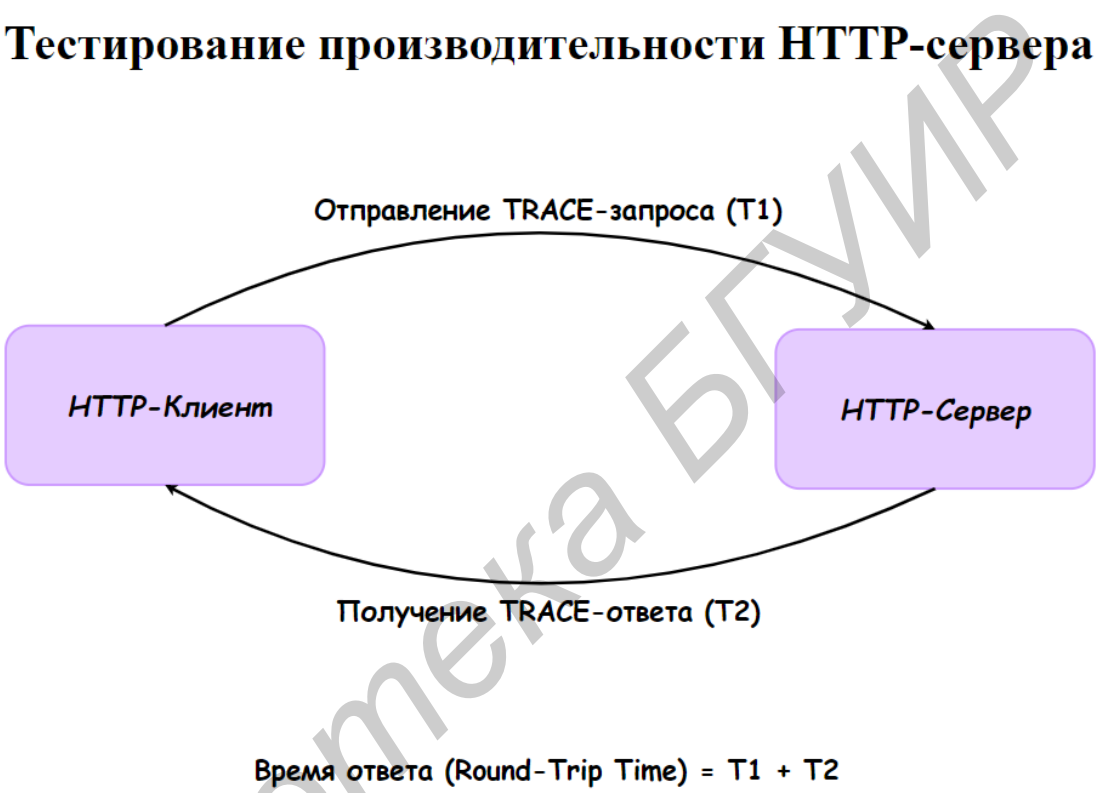

Рисунок 4 – Сценарий для тестирования производительности HTTP-сервера

Для WebSocket и WebSocket Secure серверов были предложены два варианте тестирования. Первый, эхо-сценарий, похож на сценарий для протокола HTTP, за исключением того, что вместо TRACE-запроса пересылаются сообщения по 32 байт каждое.

Второй сценарий представляет собой отправление сервером оповещений всем подключенным к нему клиентам. Схема данного подхода представление на рисунке 5. В результате подсчитываются все сообщения, полученные всеми клиентами, а затем высчитываются характеристики производительности.

Рассылочный сценарий тестирования производительности

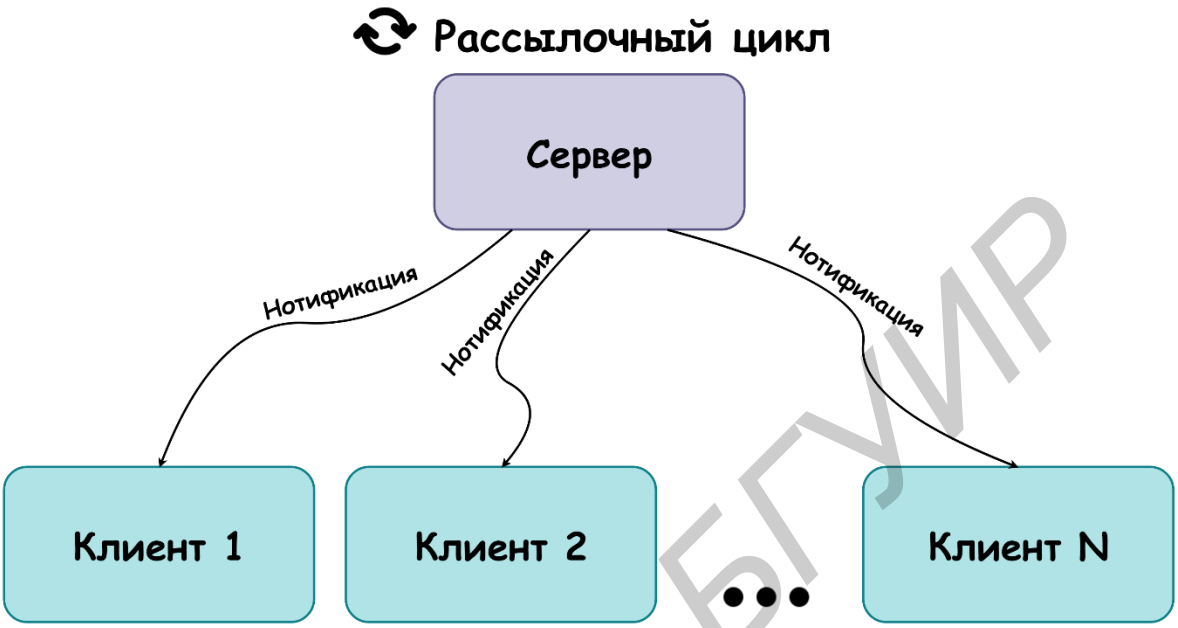

Рисунок 5 – Рассылочный сценарий для тестирования производительности WebSocket-сервера

Полные результаты тестирования представлены в 6 таблицах в данной работе. После по тем же самым сценариям было произведено тестирование производительности аналогов – gRPC от компании Google и Apache Thrift от компании Facebook. Сравнение результатов производительности с аналогами показал, что предложенное решение на 15-20 % эффективнее.

ЗАКЛЮЧЕНИЕ

В результате работы над магистерской диссертацией были выполнены следующие задачи.

Был произведен подробный анализ веб-протоколов прикладного уровня, выявлены их достоинства, недостатки и различия между друг другом, а также проанализированы возможные оптимизации при их реализации.

Для каждого из выбранных протоколов (HTTP, HTTPS, WebSocket, WebSocket Secure) была разработана и написана собственная эффективная реализация, которая включает в себя как сущности серверов, так и клиентов и сессий.

Для тестирования были разработаны примеры для каждого протокола с удобным пользовательским интерфейсом (веб-сайт), которые также служат и как примеры реализации специализированных сущностей на основе данной работы.

Была предложена и реализована оптимальная архитектура высокопроизводительного веб-сервера для получения, обработки и хранения информации. А также решена проблема при хранении данных, путем создания хранилища с быстрой сменой базы данных.

Для тестирования производительности полученного решения были предложены и разработаны три различных сценария, а результаты оформлены с помощью сводных таблиц.

Сравнение результатов производительности с аналогами показал, что предложенное решение на 15-20 % эффективнее. Это было достигнуто благодаря предложенной архитектуре и многочисленным оптимизациям как ранее разработанного TCP/UDP сервера, так и оптимизациям, описанным в главе 3 данной работы (создание отдельных классов HTTP-запроса и ответа, использование кэшей, наследование транспортных протоколов, создание вспомогательных методов и др.)

Таким образом, магистерская диссертация является полностью завершенной, а поставленная цель и задачи решены в полной мере. Данное решение может быть использовано во многих сферах, особенно в тех, где необходима высокая скорость обработки запросов (например, в области биржевой торговли, игровой индустрии и других). А предложенная архитектура и оптимизации могут быть использованы для разработки других проектов с целью увеличения их производительности.

СПИСОК ОПУБЛИКОВАННЫХ РАБОТ

1 Лагута, А.С. Архитектура высокопроизводительного сервиса для эффективной обработки больших объемов данных // Студенческий форум: электрон. научн. журн. – 2019. – № 35(86). – стр. 76-78

2 Лагута, А.С. Решение для эффективного хранения больших объемов данных // Студенческий форум: электрон. научн. журн. – 2021. – № 13(149). – стр. 32-34