ИНФОРМАТИКА И ИНФОРМАЦИОННЫЕ ПРОЦЕССЫ

УДК 004.75 Научная статья
https://doi.org/10.35330/1991-6639-2026-28-2-34-50
ISSQXZ

Архитектура распределенной системы хранения и обработки больших данных на основе Apache Ozone и Argo Workflows

К. А. Полянцева✉, А. В. Комлев, М. Г. Городничев

Московский технический университет связи и информатики
111024, Россия, Москва, ул. Авиамоторная, 8А

Аннотация. В статье рассматривается архитектура распределенной системы хранения и обработки больших данных, построенная на основе интеграции объектного хранилища Apache Ozone и системы оркестрации вычислительных процессов Argo Workflows.

Цель исследования. Разработка и исследование архитектуры распределенной системы хранения и обработки больших данных, основанной на интеграции Apache Ozone и Argo Workflows, реализующей принцип разделения функций хранения и вычислений, а также оценка эффективности предложенного решения по сравнению с традиционной архитектурой Apache Hadoop.

Методы исследования. Использованы методы системного анализа архитектур больших данных, сравнительного экспериментального тестирования распределенных систем хранения и обработки информации, а также методы математического моделирования для формализации процессов масштабирования ресурсов, времени выполнения вычислений и эффективности хранения данных. Экспериментальная оценка проводилась на кластерах Apache Ozone и Apache Hadoop с использованием Apache Spark для выполнения вычислительных задач.

Результаты. Разработана архитектура распределенной системы, обеспечивающая независимое масштабирование подсистем хранения и вычислений за счет использования объектного хранилища Apache Ozone и оркестрации вычислительных процессов на базе Argo Workflows в контейнерной среде Kubernetes. Предложена методика интеграции компонентов без использования промежуточного S3-шлюза, позволяющая снизить накладные расходы взаимодействия. Проведенные экспериментальные исследования показали сопоставимую производительность предложенного решения с Hadoop-кластером при операциях чтения, записи и обработки данных, а также преимущества в гибкости масштабирования и эффективности использования дискового пространства при применении erasure coding.

Выводы. Результаты исследования подтверждают перспективность использования архитектуры на основе Apache Ozone и Argo Workflows в качестве альтернативы традиционным платформам обработки больших данных. Раздельная архитектура хранения и вычислений позволяет повысить гибкость инфраструктуры, оптимизировать использование ресурсов и снизить затраты на хранение данных при сохранении сопоставимого уровня производительности. Предложенный подход может быть применен при построении корпоративных аналитических платформ, систем обработки больших данных и инфраструктур машинного обучения.

Ключевые слова: распределенные системы хранения данных, большие данные, Apache Ozone, Argo Workflows, Kubernetes, Apache Spark, объектные хранилища, разделение хранения и вычислений, масштабируемость, обработка данных, контейнерные вычисления, отказоустойчивость

Поступила 25.02.2026, одобрена после рецензирования 11.03.2026, принята к публикации 25.03.2026

Для цитирования. Полянцева К. А., Комлев А. В., Городничев М. Г. Архитектура распределенной системы хранения и обработки больших данных на основе Apache Ozone и Argo Workflows // Известия Кабардино-Балкарского научного центра РАН. 2026. Т. 28. № 2. С. 34–50. DOI: 10.35330/1991-6639-2026-28-2-34-50

MSC: 68M14 Original article

Architecture of a distributed storage and big data processing system based on Apache Ozone and Argo Workflows

K.A. Polyantseva✉, A.V. Komlev, M.G. Gorodnichev

Moscow Technical University of Communications and Informatics
8A, Aviamotornaya street, Moscow, 111024, Russia

Abstract. The article discusses the architecture of a distributed big data storage and processing system based on the integration of the Apache Ozone object storage and the Argo Workflows computing process orchestration system.

Aim. Development and research of the architecture of a distributed big data storage and processing system based on the integration of Apache Ozone and Argo Workflows, implementing the principle of separation of storage and computing functions, as well as evaluating the effectiveness of the proposed solution compared to the traditional Apache Hadoop architecture.

Methods. Methods of system analysis of big data architectures, comparative experimental testing of distributed information storage and processing systems, as well as mathematical modeling methods are used to formalize the processes of scaling resources, computing time, and data storage efficiency. The experimental evaluation is carried out on Apache Ozone and Apache Hadoop clusters using Apache Spark to perform computational tasks.

Results. A distributed system architecture has been developed that provides independent scaling of storage and computing subsystems through the use of Apache Ozone object storage and orchestration of computing processes based on Argo Workflows in the Kubernetes container environment. A method for integrating components without using an intermediate S3 gateway is proposed, which reduces the overhead costs of interaction. Experimental studies have shown comparable performance of the proposed solution with a Hadoop cluster for data reading, writing, and processing, as well as advantages in scaling flexibility and disk space efficiency when using erasure coding.

Conclusions. The results of the study confirm the prospects of using architecture based on Apache Ozone and Argo Workflows as an alternative to traditional big data platforms. The separate storage and computing architecture allow for increased infrastructure flexibility, optimized resource usage, and lower data storage costs while maintaining comparable performance levels. The proposed approach can be applied in the construction of corporate analytical platforms, big data processing systems and machine learning infrastructures.

Keywords: distributed storage systems, big data, Apache Ozone, Argo Workflows, Kubernetes, Apache Spark, object storage, separation of storage and computing, scalability, data processing, container computing, fault tolerance

Submitted 25.02.2026, approved after reviewing 11.03.2026, accepted for publication 25.03.2026

For citation. Polyantseva K.A., Komlev A.V., Gorodnichev M.G. Architecture of a distributed storage and big data processing system based on Apache Ozone and Argo Workflows. *News of the Kabardino-Balkarian Scientific Center of RAS.* 2026. Vol. 28. No. 2. Pp. 34–50. DOI: 10.35330/1991-6639-2026-28-2-34-50

Введение

В условиях стремительного роста объемов данных, а также растущей популярности систем на основе искусственного интеллекта и алгоритмов машинного обучения задача эффективного хранения и обработки больших массивов информации становится наиболее важной. Популярные решения, такие как Apache Hadoop, успели хорошо зарекомендовать

себя в этой сфере, однако с развитием технологий и изменением требований к инфраструктуре начинают показывать худшие результаты на фоне конкурентов [1, 2].

Современные тенденции в развитии систем больших данных указывают на необходимость разделения функций хранения и обработки данных, что позволяет оптимизировать использование ресурсов, повысить гибкость инфраструктуры и снизить общую стоимость владения.

В настоящее время разделение функций хранения и произведения вычислений позволяет ощутимо оптимизировать использование ресурсов, а также повысить гибкость инфраструктуры. В данном контексте интеграция Apache Ozone как объектного хранилища и Argo Workflows как платформы оркестрации вычислительных процессов представляет собой перспективное решение, способное преодолеть ограничения традиционных подходов.

Актуальность данного исследования обусловлена потребностью потребителей в более гибких и универсальных подходах, не уступающих привычным системам в производительности.

Целью данной работы является разработка и оценка эффективности комплексного решения на основе интеграции Apache Ozone и Argo Workflows.

Apache Hadoop: архитектура, принципы работы и применение

Несмотря на наличие разнообразия технологий в экосистеме больших данных, Apache Hadoop по-прежнему остается одним из главных инструментов в корпоративном секторе. С момента своего появления в 2006 году он произвел революцию в области обработки больших данных, предоставив открытую платформу, способную хорошо масштабироваться. Многие современные решения для работы с большими данными либо основаны на Hadoop, либо разработаны под влиянием его основных принципов.

Экосистема Apache Hadoop представляет собой комплекс взаимосвязанных технологий, но в ее основе лежат три ключевых компонента [3, 4]:

1. HDFS (Hadoop Distributed File System) – распределенная файловая система.

2. YARN (Yet Another Resource Negotiator) – система управления ресурсами и планирования заданий в кластере, отделяющая эти функции от модели программирования.

3. MapReduce – программная модель и фреймворк для распределенной обработки больших наборов данных на кластерах.

Вокруг базовой платформы Hadoop выросла обширная экосистема инструментов и технологий, каждая из которых решает определенные задачи: Apache Hive, Apache HBase, Apache Spark, Apache Sqoop и т.д.

Ключевым принципом работы Hadoop является концепция «перенести вычисления к данным, а не данные к вычислениям». Он распределяет задачи обработки на узлы, где уже хранятся соответствующие данные. Это позволяет минимизировать сетевой трафик и более эффективно использовать локальность данных.

Также Hadoop спроектирован с учетом высокой вероятности аппаратных сбоев при работе с большими кластерами. Он обеспечивает надежность через репликацию данных и перезапускает задачи на других узлах кластера в случае отказа.

Кроме того, система позволяет линейно масштабировать производительность и объем хранения путем добавления новых узлов в кластер.

Несмотря на свою значительную роль и широкое распространение, Apache Hadoop сталкивается с рядом ограничений и недостатков. Они в значительной степени обусловлены архитектурными решениями и изменением требований к Big Data-системам.

Архитектурные ограничения HDFS [5]:

• Нагрузка на NameNode. Каждый файл и каждый блок в HDFS требует отдельной записи в памяти NameNode. При наличии большого количества мелких файлов NameNode испытывает значительную нагрузку.

• Неэффективное использование дискового пространства. Даже маленький файл в HDFS занимает как минимум один блок (по умолчанию 128 МБ), что приводит к неэффективному использованию дискового пространства при хранении малых файлов.

• Накладные расходы на операции ввода-вывода. Открытие и закрытие множества мелких файлов создает значительные накладные расходы из-за множественных сетевых запросов к NameNode.

• Единая точка отказа в NameNode. Повреждение метаданных на NameNode может привести к потере доступа к данным во всей файловой системе.

• Ограниченная масштабируемость. Поскольку NameNode хранит все метаданные файловой системы в оперативной памяти, размер кластера HDFS ограничен объемом RAM, доступным на сервере NameNode.

Ограничения модели программирования MapReduce:

• Избыточное чтение/запись данных. Каждая итерация MapReduce требует полного цикла чтения данных из HDFS и записи результатов обратно в HDFS.

• Отсутствие механизмов кэширования. MapReduce не предоставляет встроенные механизмы для кэширования промежуточных результатов между итерациями.

• Сложность реализации некоторых алгоритмов. Ограничение обработки этапами Map и Reduce усложняет реализацию сложных многоступенчатых алгоритмов.

Неэффективность для интерактивной аналитики и потоковой обработки:

• Высокая задержка. Значительное время запуска задач делает MapReduce непригодным для интерактивных запросов и обработки высокочастотных потоковых данных.

• Ограниченное разделение ресурсов. Hadoop не оптимизирован для смешанных рабочих нагрузок, сочетающих пакетную обработку и интерактивные запросы.

• Сложности интеграции. Интеграция потоковых и пакетных компонентов в единый конвейер обработки данных часто требует сложных настроек и дополнительных инструментов.

Операционные проблемы и сложность:

• Сложная конфигурация. Настройка кластера Hadoop включает множество конфигурационных параметров, требующих глубокого понимания системы.

• Сложное управление кластером. Управление кластером Hadoop, особенно крупным, требует специализированных навыков и инструментов.

• Непрозрачность процессов. Внутренние процессы Hadoop часто непрозрачны для пользователей и разработчиков, что затрудняет понимание причин проблем с производительностью.

• Ограниченные инструменты отладки. Встроенные инструменты отладки и мониторинга Hadoop часто недостаточны для сложных сценариев использования.

Проблемы с производительностью:

• Избыточное копирование данных. Промежуточные данные в процессе MapReduce часто копируются несколько раз, создавая значительные накладные расходы.

• Зависимость от дисковых операций. MapReduce в значительной степени зависит от дисковых операций, что снижает производительность по сравнению с системами, работающими в памяти.

• Сетевые накладные расходы. Передача данных между узлами в кластере Hadoop может создавать значительные сетевые накладные расходы, особенно для задач с интенсивным перемещением данных.

Подход с разделением хранения и обработки данных

Одним из ключевых направлений развития систем обработки больших данных в последние годы стал подход с разделением хранения и вычислений. В рамках данной концепции компоненты системы, отвечающие за хранение данных, физически и логически отделяются от компонентов, выполняющих их обработку. Такой подход отличается от монолитной архитектуры, представленной в Apache Hadoop, где вычисления и хранение сосуществуют на одних и тех же узлах.

Принцип раздельной архитектуры предполагает, что:

• хранилище представлено как отдельный сервис (например, объектное хранилище);

• обработка данных осуществляется внешними вычислительными системами, которые подключаются к хранилищу по API или через другие интерфейсы;

• обе части могут масштабироваться независимо в зависимости от текущих требований нагрузки.

На практике это означает, что для хранения можно использовать объектные хранилища (например, Amazon S3, Apache Ozone, MinIO), которые обеспечивают высокую доступность, отказоустойчивость и горизонтальное масштабирование. Для обработки можно использовать контейнерные платформы и пайплайны (например, Flink, Argo Workflows), которые могут быть развернуты динамически в Kubernetes.

Преимущества раздельного подхода:

1. Гибкое масштабирование. Хранение и вычисления масштабируются независимо, что позволяет более точно подстраиваться под текущие нужды и избегать избыточных затрат.

2. Оптимизация затрат. Организации могут выбирать более дешевые решения для хранения и более производительные для обработки. В случае облачных решений это позволяет оплачивать только реально используемые ресурсы.

3. Повышенная отказоустойчивость. Отказ вычислительных узлов не влияет на доступность хранилища, и наоборот.

4. Упрощенное администрирование и обновление. Обновления компонентов могут проводиться независимо друг от друга. Также появляется возможность выбирать наиболее подходящие инструменты обработки данных без изменения хранилища.

5. Универсальность хранилища. Один и тот же пул данных может использоваться различными инструментами анализа, не требуя дублирования.

Apache Ozone как система хранения данных

Apache Ozone – это распределенное объектное хранилище с открытым исходным кодом, разработанное в рамках экосистемы Apache Hadoop. Оно было создано как ответ на архитектурные ограничения HDFS (Hadoop Distributed File System) при работе с большими объемами мелких файлов и современной облачной инфраструктурой. Ozone обеспечивает высокую масштабируемость, отказоустойчивость и совместимость с современными системами обработки данных, включая контейнерные и микросервисные архитектуры [6]. Архитектура Ozone представлена на рисунке 1.

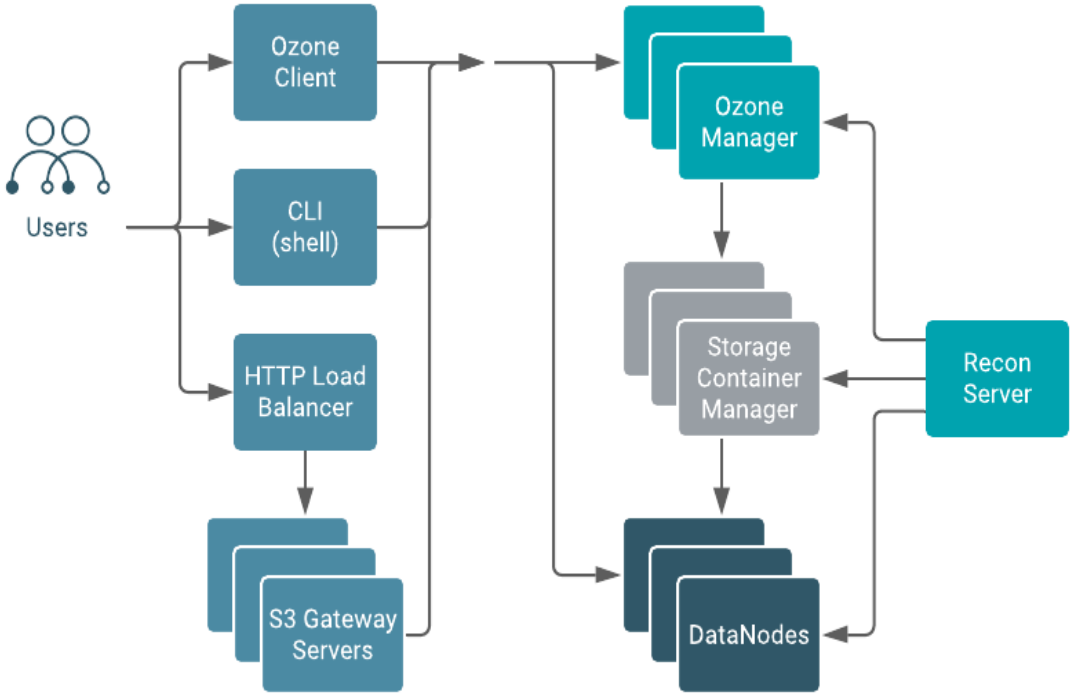

Рис. 1. *Архитектура Apache Ozone*

Fig. 1. *Apache Ozone Architecture*

Apache Ozone основан на микросервисной архитектуре и логически разделяется на несколько ключевых компонентов:

OM (Ozone Manager) – сервис, управляющий метаданными: пространствами имен, bucket'ами и ключами объектов. Он аналогичен NameNode в HDFS, но масштабируется горизонтально.

SCM (Storage Container Manager) – управляет контейнерами хранения и координирует узлы хранения (DataNodes). Отвечает за репликацию, размещение данных и контроль целостности.

DataNodes – узлы хранения, содержащие блоки данных в виде контейнеров. Обеспечивают хранение объектов, репликацию и восстановление.

Recon – компонент мониторинга и визуализации состояния кластера.

Ozone FS API – позволяет использовать Ozone как файловую систему, совместимую с HDFS.

Argo Workflows как система оркестрации вычислительных процессов

Argo Workflows – это система с открытым исходным кодом для построения и оркестрации вычислительных пайплайнов (workflows) в среде Kubernetes. Она позволяет запускать, управлять и отслеживать сложные цепочки задач, описанных в виде направленных ациклических графов (DAG). Благодаря глубокой интеграции с Kubernetes Argo Workflows [7, 8] обеспечивает масштабируемость, надежность и автоматизацию обработки данных. Архитектура Argo Workflows представлена на рисунке 2.

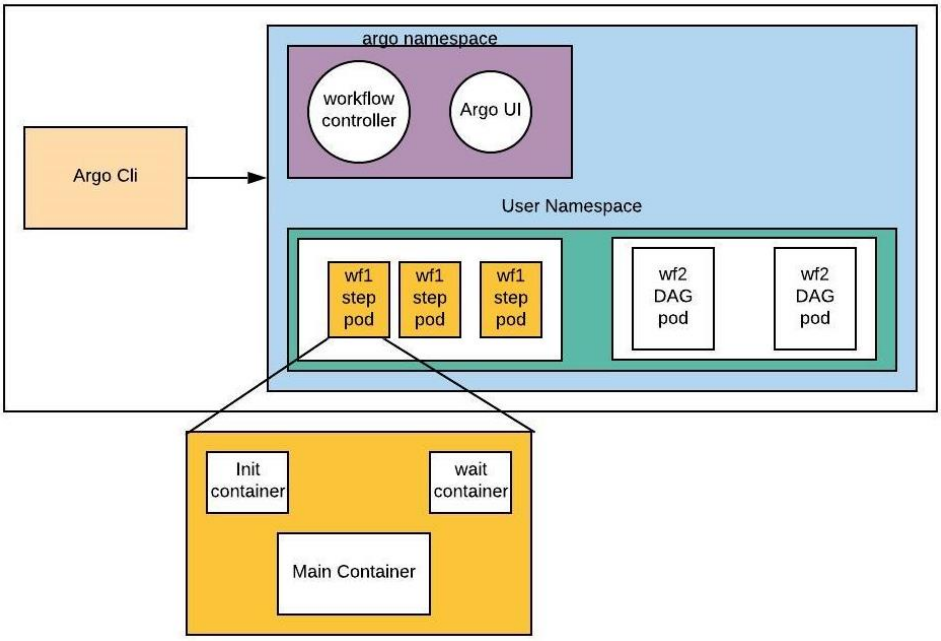

***Рис. 2**. Архитектура Argo Workflows*

***Fig. 2**. Argo Workflows architecture*

Argo Workflows развертывается как набор Kubernetes-подобных ресурсов и контроллеров. Основные компоненты:

Workflow Controller – основной контроллер, который отслеживает Workflow и управляет созданием и выполнением подов (Pods) в соответствии с определением пайплайна.

Workflow CRD (Custom Resource Definition). Пользователь описывает workflow в YAML-файле с использованием Argo DSL, а Kubernetes управляет этим объектом как собственным ресурсом.

Pods. Каждое задание (шаг workflow) исполняется в виде отдельного POD в кластере Kubernetes.

Argo server – отдельный компонент, предоставляющий визуальный интерфейс и API для запуска задач.

Преимущества интеграции Apache Ozone и Argo Workflows

Интеграция Apache Ozone и Argo Workflows представляет собой полноценное решение, сочетающее в себе преимущества масштабируемого распределенного хранилища и гибкой системы оркестрации вычислений. Такое объединение позволяет реализовать раздельную архитектуру хранения и обработки данных, обеспечивающую эффективность, отказоустойчивость и оптимальное использование ресурсов, описанные ранее.

Apache Ozone и Argo Workflows выполняют строго разграниченные функции. Ozone отвечает за надежное хранение, репликацию, управление метаданными и доступ к данным через объектную модель. Argo Workflows управляет процессами обработки, запуском контейнеров, планированием задач и контролем зависимостей. Такое разделение упрощает архитектуру системы, повышает отказоустойчивость и облегчает сопровождение [9].

Одним из главных ограничений традиционных решений является необходимость совместного масштабирования хранилища и вычислительных узлов. В интегрированной архитектуре Ozone + Argo объем хранилища увеличивается за счет расширения кластера Ozone.

Производительность обработки повышается путем добавления ресурсов в Kubernetes-кластер, где исполняются workflows. Это позволяет гибко адаптироваться к текущим нагрузкам и снижать инфраструктурные издержки, особенно в облачной или гибридной среде.

В Argo Workflows каждая задача обрабатывается в изолированном окружении, что обеспечивает:

- легкое управление зависимостями;
- повышенную безопасность;
- возможность запуска гетерогенных задач (на Python, R, Spark, etc.) в рамках одного пайплайна.

Данные, необходимые для вычислений, считываются из Ozone через совместимые API, а результаты могут быть обратно записаны в хранилище.

В случае отказа Ozone сохраняет данные с репликацией или erasure coding, а Argo автоматически перезапускает упавшие шаги пайплайна, логирует ошибки и сохраняет промежуточное состояние. Это делает всю систему устойчивой к сбоям как на уровне хранения, так и на уровне вычислений.

Обе системы могут быть развернуты в Kubernetes [10, 11], что делает их полностью совместимыми с современными DevOps-практиками. Helm-чарты и манифесты позволяют автоматизировать установку. Появляется возможность интеграции с CI/CD пайплайнами. А также обеспечивается простота настройки мониторинга и логирования через Prometheus, Grafana, Loki, ELK и т.п.

Архитектура предлагаемого решения

Предлагаемое решение реализовано по принципу раздельных подсистем хранения и обработки данных. Оно включает в себя такие компоненты: хранилище Apache Ozone, развернутое на кластере из 12 серверов (сервер кластера Apache Ozone – 8 ядер ЦП, 32 Гб оперативной памяти, диск HDD 256 Гб); система обработки данных Argo Workflows на кластере OpenShift, состоящем из 6 серверов (сервер кластера OpenShift – 32 ядра CPU, 512 Гб ОЗУ, диск HDD 50 Гб), а также вспомогательные сервисы для интеграции и эксплуатации.

Как мы видим из конфигурации серверов, для обработки данных используются высокопроизводительные серверы, в разы превосходящие хосты Ozone. В свою очередь для хранения используются более бюджетные серверы с малым количеством вычислительных мощностей, но с большим объемом дисков.

Далее рассмотрим компоненты систем, которые будут развернуты.

Кластер Ozone собран в стандартной конфигурации, включающей в себя Ozone Manager, Storage Container Manager, DataNodes; Ozone S3 Gateway. Для управления установлен Ambari сервер (аналогичный Hadoop), кроме того для его работы будет использоваться СУБД PostgreSQL.

Для Argo используется стандартная конфигурация OpenShift.

Вспомогательные сервисы, необходимые для работы проекта: HashiCorp Vault, Prometheus и Grafana.

Методология интеграции Apache Ozone и Argo Workflows

Интеграция подсистем хранения и обработки данных происходит без использования промежуточных S3-Gateway. В качестве фреймворка и механизма обработки данных используется образ Apache Spark, размещенный в POD нашего Workflow, а для взаимодействия с Ozone используется нативный клиент, аналогично встроенный в исполняемое окружение рабочего контейнера. Схема интеграции отражена на рисунке 3.

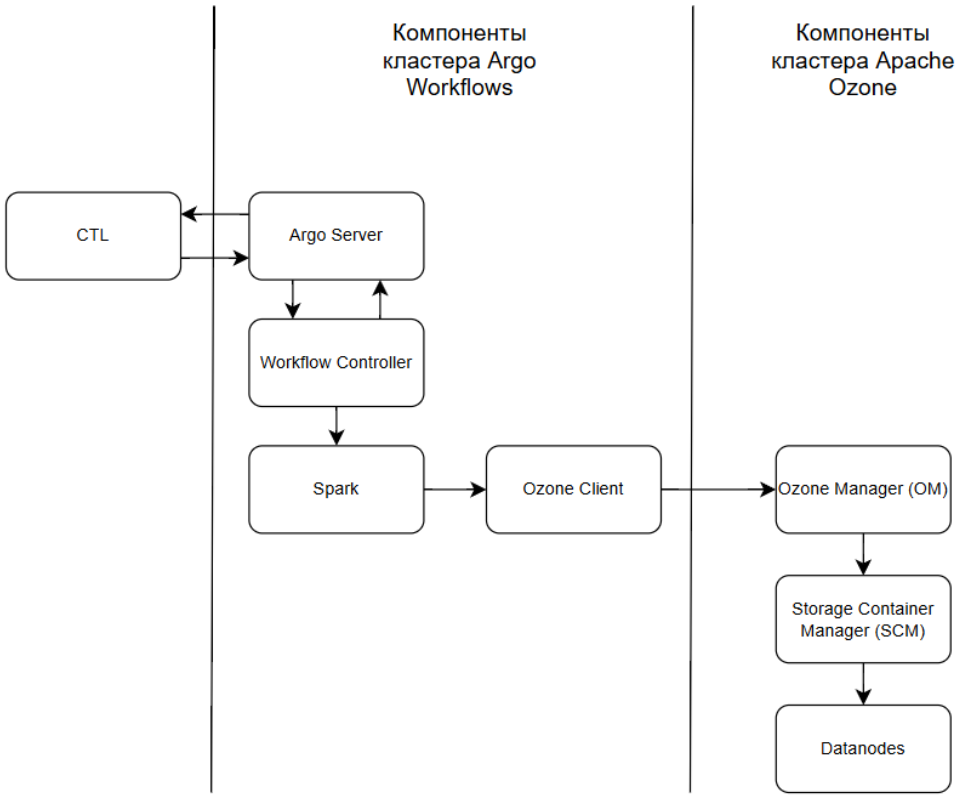

Рис. 3. *Схема интеграции Apache Ozone и Argo Workflows*

Fig. 3. *Apache Ozone and Argo Workflows integration diagram*

Компоненты интеграции:

CTL – универсальный менеджер запуска потоков. Сервис редактирует стандартный Workflow template с параметрами, указанными пользователем, и передает готовый манифест в Argo Server. Argo Server предоставляет API для запуска Workflow. Workflow Controller, получив команду от Argo Server, непосредственно запускает POD с Workflow и отслеживает его. Spark выполняет указанный ему код, обращаясь к Ozone через Ozone client. Ozone client выполняет команды (аналогичные командам для работы с HDFS в Hadoop) с помощью взаимодействия с конкретным компонентом кластера Ozone. Ozone Manager в свою очередь производит изменения в хранилище через Storage Container Manager. Storage Container Manager определяет, на какие Datanode распределяются объекты, и поддерживает консистентность и доступность данных. Datanode непосредственно хранят физические данные объектов. Они получают команды от SCM и обрабатывают запросы на чтение/запись с рабочих узлов, используя контейнерную модель хранения.

Преимуществом такой модели интеграции является отказ от использования S3 Gateway, что повышает производительность за счет отсутствия дополнительной прослойки HTTP. Взаимодействие с хранилищем происходит напрямую через файловый API (ozone fs) в Ozone client. Ozone client реализует POSIX-подобную модель работы с данными, как в Hadoop. Это упрощает работу с данными и миграцию пользователей с Hadoop на Ozone.

Методика оценки эффективности системы хранения и обработки данных

Для объективной оценки эффективности предлагаемого решения разработаем методику сравнения с традиционной системой в виде Apache Hadoop. Такое сравнение позволит

определить, насколько описанный подход может конкурировать с наиболее активно используемой на рынке системой и, возможно, выявить его преимущества и недостатки. Сравнительный анализ проведем по совокупности показателей, характеризующих ключевые процессы при работе с данными. Среди показателей можно привести такие, как производительность операций чтения и записи, эффективность выполнения вычислений, а также отказоустойчивость и экономическая целесообразность.

Для обеспечения корректности тестирование будет производиться в сопоставимых программно-аппаратных условиях, что позволит устранить возможное влияние различий в инфраструктуре:

- Кластер Apache Ozone с Argo Workflows, развернутый в предыдущем разделе, использующий для обработки Apache Spark.
- Кластер Apache Hadoop в похожей конфигурации, 12 серверов, практически не отличающихся по CPU и RAM от Argo кластера, а по дискам от Ozone. Все вычисления будем запускать не через MapReduce, а исключительно через Spark, чтобы исключить влияние недостатков устаревшего фреймворка.

Определим два ключевых класса операций:

1. Операции хранения (I/O): запись и чтение большого количества файлов малого (1 КБ) и среднего (20 МБ) размера. Малые файлы представляют собой сложный тип нагрузки, с которым плохо справляются распределенные файловые системы. В свою очередь средние файлы представляют наиболее типичные сценарии для задачи хранения.

2. Операции обработки (compute): выполнение Spark-задач с доступом к данным из хранилища, включая простые преобразования (например, кодирование, агрегация).

Сценарий 1. Запись маленьких файлов в Ozone и Hadoop.

Цель: оценить производительность записи большого количества мелких файлов (размером 1 КБ) в обе системы хранения данных. Такой сценарий моделирует распространенную практику журналирования, хранения телеметрии и IoT-данных.

Условия проведения: размер файла – 1 Кб; количество файлов – 14000 шт.; формат – бинарный.

Метод тестирования:

- В Hadoop запись выполняется напрямую через hdfs dfs -put на клиентском узле.
- В Ozone – запуск через Argo Workflow, но без Spark: в POD передается список файлов, каждый файл записывается командой ozone fs -put.

Ожидаемые результаты: Ozone в силу своей объектной модели хранения тратит больше времени на каждый запрос из-за дополнительного оборачивания метаданных и создания отдельных объектов. В связи с этим при записи Hadoop должен показать лучший результат, однако не стоит забывать, что он для каждого файла выделит по одному блоку в HDFS, минимальный размер которого составляет 128 Мб. Соответственно оптимальность такого хранения является очень сомнительной.

Сценарий 2. Чтение маленьких файлов из Ozone и Hadoop.

Цель: оценить производительность чтения большого количества мелких файлов, как продолжение сценария 1.

Условия проведения: аналогичны сценарию 1.

Метод тестирования:

- В Hadoop чтение осуществляется через hdfs dfs -cat с перенаправлением в /dev/null.
- В Ozone – выполнение в Argo-поде, содержащем Ozone client с использованием ozone fs -cat на каждый объект.

Ожидаемые результаты: из-за большего количества метаданных и сетевых обращений Ozone может демонстрировать небольшую задержку по сравнению с HDFS.

Сценарий 3. Запись средних файлов в Ozone и Hadoop.

Цель: оценить производительность записи большого количества средних файлов (размером 20 МБ) в обе системы хранения данных приближенно к реальной загрузке данных.

Условия проведения: размер файла – 20 МБ; количество файлов – 14000 шт.; формат – бинарный.

Метод тестирования: аналогичен сценарию 1.

Ожидаемые результаты: Ozone может продемонстрировать более высокую пропускную способность, особенно при включенном Erasure Coding, благодаря эффективной работе с большими блоками.

Сценарий 4. Чтение средних файлов из Ozone и Hadoop.

Цель: оценить производительность чтения большого количества средних файлов, как продолжение сценария 3.

Условия проведения: аналогичны сценарию 3.

Метод тестирования: аналогичен сценарию 2.

Ожидаемые результаты: при последовательном доступе и хорошо настроенной сети Ozone может показывать лучшую или сравнимую производительность чтения.

Сценарий 5. Обработка данных.

Цель: оценить производительность выполнения вычислений при реальной нагрузке, включающей чтение, фильтрацию, агрегацию и запись результатов.

Описание задачи: в рамках тестирования используется CSV-файл объемом ~1 ГБ, содержащий около 5 миллионов строк. Файл моделирует типичную структурированную выгрузку из реляционной СУБД (например, журнал заказов). В рамках задачи необходимо прочитать таблицу из распределенного хранилища, отфильтровать строки по значению одного из полей, выполнить агрегацию, подсчитав количество записей по значению одного из полей, а после записать результат обратно в файловую систему.

Метод тестирования:

- Для Hadoop обработка выполняется с помощью Apache Spark, запущенного через spark-submit, ресурсы выделяются компонентом Yarn.
- Для Ozone процесс запускается в поде Argo кластера через spark-submit, данные читаются и записываются с помощью ozone-client.

Анализ полученных результатов и выявление преимуществ предлагаемого решения

Проанализируем результаты тестирования, описанного в предыдущем разделе. В рамках первого и третьего сценариев на запись файлов были получены следующие результаты (табл. 1).

Таблица 1. Результаты тестирования систем на запись

Table 1. Results of system testing

Система	**Среднее время записи 1 объекта 1 Кб, мс**	**Среднее время записи 1 объекта размером 20 Мб, мс**
Hadoop	**450**	2948
Ozone + Argo	461	**2919**

Как и ожидалось, при работе с маленькими файлами Hadoop демонстрирует более высокую производительность. Хотя разница и не критична, лишь ~2.4%, это подтверждает недостаток объектных хранилищ, заключающийся в дополнительной нагрузке при создании каждого объекта.

При работе с файлами большего объема Ozone показывает себя лучше, что предположительно связано с использованием механизма Erasure Coding [12–14].

В рамках второго и четвертого сценариев на чтение файлов были получены следующие результаты (табл. 2).

Таблица 2. Результаты тестирования систем на чтение

Table 2. Results of reading system testing

Система	Среднее время чтения 1 объекта 1 Кб, мс	Среднее время чтения 1 объекта размером 20 Мб, мс
Hadoop	**20**	458
Ozone + Argo	26	**381**

По результатам тестирования на чтение и запись можно прийти к выводу, что общее поведение систем вполне сопоставимо, это уже является значимым достижением для предлагаемой архитектуры.

В рамках пятого сценария тестирования были получены следующие результаты (табл. 3).

Таблица 3. Результаты обработки данных по сценарию 5

Table 3. Results of data processing for scenario 5

Метрика	Hadoop	Ozone + Argo
Объем файла, ГБ	1	1
Кол-во строк	~ 5 000 000	~ 5 000 000
Общее время выполнения, сек.	**48.2**	50.6
Время запуска Spark job, сек.	**3.2**	6.7
Время чтения, сек.	9.5	**9.3**
Время обработки, сек.	26.7	**25.5**
Время записи, сек.	**8.8**	9.1

Общее время выполнения у Ozone и Argo оказалось немного выше, что говорит о почти идентичной производительности систем при запуске полноценного пайплайна обработки.

Время запуска Spark Job у Argo также выше, это объясняется тем, что запуск производится внутри контейнеризированного окружения, а это в свою очередь влечет за собой дополнительные шаги подготовки, такие как выделение пода и инициализация окружения.

Время чтения и записи демонстрирует результаты, аналогичные первым сценариям тестирования.

Время обработки оказалось ниже у Argo. Это может свидетельствовать о лучшем распределении ресурсов Kubernetes по сравнению с Yarn.

По результатам всех тестов можно сделать главный вывод, что связка Ozone и Argo не уступает традиционному решению в виде Hadoop, в записи и чтении она немного хуже за счет особенностей объектных хранилищ, однако в задачах обработки превосходит Hadoop, так как контейнерная архитектура Kubernetes эффективнее Yarn, распределяющего ресурсы там.

Далее необходимо провести аналитическое исследование систем по метрикам масштабируемости, отказоустойчивости и экономической эффективности.

Ozone и Argo функционируют независимо. Ozone – система хранения, Argo – оркестратор вычислений, что позволяет масштабировать их раздельно. Ozone способен обрабатывать миллиарды объектов в одном кластере, потому что он не ограничен оперативной памятью Namenode как Hadoop. Argo и Kubernetes в целом поддерживают тысячи одновременных PODов.

Рассмотрим расход дискового пространства для обеспечения одинакового показателя отказоустойчивости (табл. 4).

Таблица 4. Сравнение требований к дисковому пространству для обеспечения отказоустойчивости

Table 4. Comparison of disk space requirements for fault tolerance

Система	**Механизм**	**Фактор избыточности**	**Требуемое пространство для 100 Тб данных**
Hadoop	Простая репликация	3x	300 Тб
Ozone	Erasure Coding	1.5x	150 Тб

Итого механизм отказоустойчивости Ozone в два раза позволяет сократить используемое дисковое пространство, что является ощутимым при рассмотрении финансовой стороны подобных систем.

Математическое описание предлагаемой системы

Модель распределения ресурсов и масштабируемости. Предлагаемая архитектура основана на принципе разделения функций хранения и обработки данных. Для формального анализа преимуществ данного подхода введем модель распределения ресурсов системы.

Рассмотрим два основных типа ресурсов:

- вычислительные ресурсы (CPU, RAM),
- ресурсы хранения (дисковое пространство).

В традиционной архитектуре Hadoop узлы выполняют обе функции одновременно, поэтому увеличение объема хранения автоматически приводит к увеличению вычислительных ресурсов, даже если это не требуется. Общий объем ресурсов можно представить векторной функцией

$$\mathbf{R}_{hadoop}(n) = \sum_{i=1}^{n} (\mathbf{C}_i + \mathbf{S}_i)$$

при предположении однородности узлов:

$$\mathbf{R}_{hadoop}(n) = n(\mathbf{C} + \mathbf{S}).$$

В предлагаемой архитектуре функции хранения и обработки разделены между различными типами узлов. Тогда общий объем ресурсов определяется выражением

$$\mathbf{R}_{split}(n_s, n_c) = \sum_{i=1}^{n_s} \mathbf{S}_i + \sum_{j=1}^{n_c} \mathbf{C}_j .$$

Для однородных узлов:

$$\mathbf{R}_{split}(n_s, n_c) = n_s \mathbf{S} + n_c \mathbf{C} .$$

Данная модель демонстрирует ключевое преимущество архитектуры – возможность независимого масштабирования хранения и вычислений.

Модель времени выполнения вычислительного процесса. Время выполнения задачи обработки данных состоит из нескольких этапов, каждый из которых может вносить вклад в общую задержку выполнения.

Общее время выполнения вычислительного процесса представим как сумму времен отдельных фаз:

$$T_{total} = T_{sched} + T_{start} + T_{read} + T_{proc} + T_{write} + T_{over},$$

где T_{sched} – время планирования задачи, T_{start} – время запуска вычислительной среды, T_{read} – время чтения данных, T_{proc} – время вычислений, T_{write} – время записи результатов, T_{over} – накладные расходы.

Контейнерная архитектура может увеличивать время запуска, но одновременно снижать время вычислений за счет более эффективного управления ресурсами.

Модель доступа к данным и влияние промежуточных слоев. В архитектуре с использованием S3-шлюза взаимодействие с системой хранения включает дополнительный уровень сетевых и протокольных накладных расходов. Время операции можно представить как

$$T_{s3} = T_{net} + T_{http} + T_{storage}.$$

При использовании нативного клиента

$$T_{native} = T_{net} + T_{storage}.$$

Тогда выигрыш по времени определяется выражением

$$\Delta T = T_{s3} - T_{native} = T_{http}.$$

Данная модель показывает, что отказ от промежуточного шлюза позволяет уменьшить задержки за счет устранения протокольных накладных расходов.

Метрики производительности операций хранения. Для анализа результатов экспериментов используются стандартные показатели производительности. Среднее время операции определяется как

$$T_{avg} = \frac{1}{N}\sum_{i=1}^{N} T_i.$$

Пропускная способность системы:

$$Throughput = \frac{N \cdot B}{T},$$

где N – число операций, B – размер объекта, T – общее время выполнения.

Модель сравнительной эффективности систем. Для количественного сравнения систем используется коэффициент ускорения

$$S = \frac{T_{hadoop}}{T_{ozone}}.$$

Разницу во времени выполнения можно представить как сумму вкладов отдельных этапов:

$$\Delta T = T_{ozone} - T_{hadoop}$$
$$\Delta T = \Delta T_{start} + \Delta T_{read} + \Delta T_{proc} + \Delta T_{write}$$

Модель надежности хранения и избыточности данных. Рассмотрим объем дискового пространства, необходимый для хранения данных с учетом отказоустойчивости. Для Hadoop с коэффициентом репликации r:

$$S_{hadoop} = r \cdot D.$$

Для Apache Ozone с erasure coding:

$$S_{ozone} = k \cdot D.$$

Коэффициент экономии дискового пространства:

$$E_{space} = \frac{S_{hadoop}}{S_{ozone}} = \frac{r}{k}.$$

Заключение

В работе предложена архитектура распределенной системы хранения и обработки больших данных, основанная на интеграции Apache Ozone и Argo Workflows и реализующая принцип разделения функций хранения и вычислений. Проведенный анализ существующих решений показал, что традиционные монолитные архитектуры, такие как Apache Hadoop, обладают рядом ограничений, связанных с необходимостью совместного масштабирования ресурсов, сложностью интеграции с контейнерными платформами и неэффективным использованием инфраструктуры при изменяющихся нагрузках.

Разработанная архитектура обеспечивает независимое масштабирование подсистем хранения и обработки данных, что позволяет более эффективно использовать вычислительные и дисковые ресурсы. В рамках исследования была предложена методология интеграции компонентов системы без использования промежуточного S3-шлюза, что позволило снизить накладные расходы взаимодействия и упростить доступ к данным для вычислительных приложений.

Для количественной оценки эффективности предложенного подхода была разработана математическая модель системы, описывающая распределение ресурсов, время выполнения вычислительных процессов, показатели производительности операций хранения и экономические характеристики инфраструктуры. Модель позволила формально обосновать преимущества раздельной архитектуры по сравнению с традиционными решениями.

Проведенные экспериментальные исследования показали, что предложенная система обеспечивает сопоставимую производительность с Hadoop-кластером при операциях хранения и обработки данных. При этом наблюдается преимущество в гибкости управления ресурсами и масштабируемости вычислительной инфраструктуры. Дополнительные задержки, связанные с запуском контейнерной среды, компенсируются более эффективным распределением вычислительных нагрузок.

Таким образом, результаты исследования подтверждают перспективность использования архитектуры на основе Apache Ozone и Argo Workflows в качестве альтернативы классическим платформам обработки больших данных. Предложенный подход может быть применен при построении корпоративных аналитических платформ, систем обработки больших данных и инфраструктур машинного обучения.

В качестве направлений дальнейших исследований можно выделить разработку методов автоматического масштабирования вычислительных ресурсов, оптимизацию алгоритмов размещения данных в распределенном хранилище, а также расширение интеграции с потоковыми системами обработки данных.

СПИСОК ЛИТЕРАТУРЫ / REFERENCES

1. *Полянцева К. А.* Высоконагруженная платформа для агрегации и анализа неструктурированных данных о состоянии дорожного полотна // Автоматизация в промышленности. 2022. № 5. С. 32–37. DOI: 10.25728/avtprom.2022.05.09

Polyantseva K. A. High-load platform for aggregation and analysis of unstructured data on road surface condition. *Avtomatizatsiya v promyshlennosti* [Automation in Industry]. 2022. No. 5. Pp. 32–37. DOI: 10.25728/avtprom.2022.05.09. (In Russian)

2. *Городничев М. Г., Титов Д. В., Липатова А. Д.* О задаче построение независимых архитектур обработки данных в интеллектуальных транспортных системах // Инженерный вестник Дона. 2025. № 11(131). С. 62–92.

Gorodnichev M.G., Titov D.V., Lipatova A.D. On problem of constructing independent data processing architectures in intelligent transport systems. *Inzhenernyy vestnik Dona* [Engineering Bulletin of the Don]. 2025. No. 11(131). Pp. 62–92. (In Russian)

3. Malik V. Hadoop Distributed file system (HDFS) with its architecture. *International Journal for Research in Applied Science and Engineering Technology*. 2025. Vol. 13. Pp. 6031–6034. DOI: 10.22214/ijraset.2025.71584

4. Kala Karun A., Chitharanjan K. A review on Hadoop – HDFS infrastructure extensions. *2013 IEEE Conference on Information & Communication Technologies*, Thuckalay, India. 2013. Pp. 132–137. DOI: 10.1109/CICT.2013.6558077

5. Zhu Z., Tan L., Li Y., Ji C. PHDFS: Optimizing I/O performance of HDFS in deep learning cloud computing platform. *Journal of Systems Architecture.* 2020. Vol. 109. Article 101810. DOI: 10.1016/j.sysarc.2020.101810

6. *Иевлев К. О., Городничев М. Г.* Сравнительный анализ систем хранения данных HDFS и Apache Ozone // Computational Nanotechnology. 2025. Т. 12. № 1. С. 26–33. DOI: 10.33693/2313-223X-2025-12-1-26-33

Ievlev K.O., Gorodnichev M.G. Comparative analysis of HDFS and Apache Ozone data storage systems. *Computational Nanotechnology.* 2025. Vol. 12. No. 1. Pp. 26–33. DOI: 10.33693/2313-223X-2025-12-1-26-33. (In Russian)

7. Wilkinson S. R., Aloqalaa M., Belhajjame K. et al. Applying the FAIR principles to computational workflows. *Scientific Data.* 2025. Vol. 12. Article 328. DOI: 10.1038/s41597-025-04451-9

8. Gustafsson O.J.R., Wilkinson S.R., Bacall F. et al. WorkflowHub: a registry for computational workflows. *Scientific Data*. 2025. Vol. 12. Article 837. DOI: 10.1038/s41597-025-04786-3

9. Tourouta E., Gorodnichev M., Polyantseva K., Moseva M. Providing fault tolerance of cluster computing systems based on fault-tolerant dynamic computation planning. *Lecture Notes in Information Systems and Organisation: 3rd*. Virtual, Online, 2022. Pp. 143–150. DOI: 10.1007/978-3-030-94252-6_10

10. Kumar B., Verma A., Verma P. Introduction of kubernetes. Modern kubernetes: From core concepts to intelligent autoscaling for cloud applications. Cham: Springer, 2026. Pp. 1–15. (Studies in Autonomic, Data-driven and Industrial Computing). DOI: 10.1007/978-3-032-12972-7_1

11. Aqasizade H., Ataie E., Bastam M. Kubernetes in action: Exploring the performance of Kubernetes distributions in the cloud. *Software: Practice and Experience*. 2025. Vol. 55. Pp. 1711–1725. DOI: 10.1002/spe.70000

12. Lucani D., Feher M. HyRES: A hybrid replication and erasure coding approach to data storage. 2025. 14 p. arXiv: 2511.00896. URL: https://arxiv.org/abs/2511.00896 (accessed: 22/02/2026)

13. Shen Z., Cai Y., Cheng K., Lee P. P. C., Li X., Hu Y., Shu J. A survey of the past, present, and future of erasure coding for storage systems. *ACM Transactions on Storage.* 2025. Vol. 21. No. 1. Article 4. 39 p. DOI: 10.1145/3708994

14. Ibrahim S., Darrous J. Erasure coding aware block placement for data-intensive applications. *ACM SIGOPS Operating Systems Review.* 2025. Vol. 59. No. 1. Pp. 62–69. DOI: 10.1145/3759441.3759451

Вклад авторов: все авторы сделали эквивалентный вклад в подготовку публикации. Авторы заявляют об отсутствии конфликта интересов.

Contribution of the authors: the authors contributed equally to this article. The authors declare no conflict of interest.

Финансирование. Исследование проведено без спонсорской поддержки.

Funding. The study was performed without external funding.

Информация об авторах

Полянцева Ксения Андреевна, канд. техн. наук, доцент кафедры «Интеллектуальный анализ данных», Московский технический университет связи и информатики;
111024, Россия, Москва, ул. Авиамоторная, 8А;
k.a.poliantseva@mtuci.ru, ORCID: https://orcid.org/0000-0002-7102-4208, SPIN-код: 8112-8560
Комлев Артем Владимирович, студент, Московский технический университет связи и информатики;
111024, Россия, Москва, ул. Авиамоторная, 8А;
komlev1257@gmail.com
Городничев Михаил Геннадьевич, канд. техн. наук, доцент, декан факультета «Информационные технологии», Московский технический университет связи и информатики;
111024, Россия, Москва, ул. Авиамоторная, 8А;
m.g.gorodnichev@mtuci.ru, ORCID: https://orcid.org/0000-0003-1739-9831, SPIN-код: 4576-9642

Information about the authors

Ksenia A. Polyantseva, Candidate of Technical Sciences, Associate Professor of the Department of Data Mining, Moscow Technical University of Communications and Informatics;
8A, Aviamotornaya street, Moscow, 111024, Russia;
k.a.poliantseva@mtuci.ru, ORCID: https://orcid.org/0000-0002-7102-4208, SPIN-code: 8112-8560
Artem V. Komlev, Student, Moscow Technical University of Communications and Informatics;
8A, Aviamotornaya street, Moscow, 111024, Russia;
komlev1257@gmail.com
Mikhail G. Gorodnichev, Candidate of Technical Sciences, Associate Professor, Dean of the Faculty of Information Technology, Moscow Technical University of Communications and Informatics;
8A, Aviamotornaya street, Moscow, 111024, Russia;
m.g.gorodnichev@mtuci.ru, ORCID: https://orcid.org/0000-0003-1739-9831, SPIN-code: 4576-9642