# Переход от хранилища данных университета к озеру: модели и методы обработки больших данных

*С.Г. Ермаков[1], М.М. Халил[31] , А.Д. Хомоненко[12], В.А. Гончаренко[21], В.А. Ходаковский[1], Р. Абу Хасан[1]*

*[1]Петербургский государственный университет путей сообщения императора Александра I, Санкт-Петербург*
*[2]Военно-космическая академия им. А.Ф. Можайского, Санкт-Петербург*
*[3]Диалский университет, Диала, Ирак*

**Аннотация**: Университеты обладают фантастическим потенциалом для получения важнейших знаний, благодаря наличию огромных объемов данных. Статья посвящена переходу университетов от хранилищ данных к более гибким и масштабируемым озёрам данных для обработки больших данных. Рассматриваются ключевые различия и сходства хранилищ данных и озер данных, где хранилища ориентированы на структурированные данные и традиционную аналитику, а озера данных позволяют работать с сырыми и полуструктурированными данными, поддерживая гибкий ELT-подход (извлечение, преобразование, загрузка). Акцентируется внимание на вызовах, связанных с миграцией данных между этими архитектурами, включая вопросы безопасности, масштабируемости и затрат. Использование облачных решений и технологий, таких как Apache Hadoop и Spark, помогает снижать затраты и эффективно управлять большими данными. Приводятся примеры способов обработки данных с помощью машинного обучения и специализированных инструментов, что способствует лучшему пониманию процесса управления и анализа данных в университетских структурах.

**Ключевые слова:** хранилище данных, озеро данных, большие данные, облачное хранение, неструктурированные данные, полуструктурированные данные.

## Введение

Хранилища (Data Warehouse, DW) и озера данных (Data Lake, DL) не являются взаимоисключающими: хранилища могут использовать озера как источники данных. Хранилище хранит структурированные данные, тогда как озеро содержит сырые данные без определенной цели. Хранилище часто представляет собой реляционную базу для бизнеса, запущенную на

специальном оборудовании или в облаке, в то время как озеро предлагает хранение сырых данных с метаданными и стандартным интерфейсом доступа. Исследований по озёрам данных проведено немного. Хотя хранилище и озеро имеют сходства, их философия управления данными и условия использования различны [1,2]. Мы рассматриваем методы преобразования данных из хранилища в озеро, освещаем проектные вопросы и разницу между двумя архитектурами, показывая, как учебные заведения могут использовать их преимущества.

## Источники данных, используемые университетом

Университеты обрабатывают большие изолированные массивы данных, собранных в разных департаментах [3]. Так, приёмное отделение собирает данные о заявках и профилях студентов, тогда как департамент по делам студентов фокусируется на студенческой деятельности и прогрессе. Архитектура хранилища данных университета показана на рис. 1.


Рис. 1. Архитектура хранилища данных университета

Виртуальная учебная среда и система управления обучением собирают данные об участии студентов, поведении и достижениях в университете (например, [4]). Отдел бухгалтерского учета отвечает за ведение финансовой отчетности учреждения, которая включает информацию о стипендиях, предоставляемых студентам. Дополнительные центры обработки данных в университете включают в себя офис регистрации, отдел исследований и грантов, отдел для персонала, библиотеки, жилищное отделение и другие.

Разделенные данные сохраняются в различных источниках, таких как ERP (Enterprise Resource Planning – планирование ресурсов предприятия) и финансовые системы, программное обеспечение для виртуального обучения, облачные базы данных, файлы JSON, листы Excel и так далее.

## Озера данных и хранилища данных

Хранилища данных интегрируют базы данных, поддерживая OLAP-приложения и бизнес-аналитику [5]. Их основа – реляционная структура данных, реализующая вариант «schema-on-write» (схема при записи), которая требует точного моделирования и длительной разработки. Это затрудняет адаптацию к новым данным и актуальной аналитике, сложны и требовательны ETL-процессы (Extract, Transform, Load – извлечение, преобразование, загрузка). Архитектура озера данных приведена рис. 2.

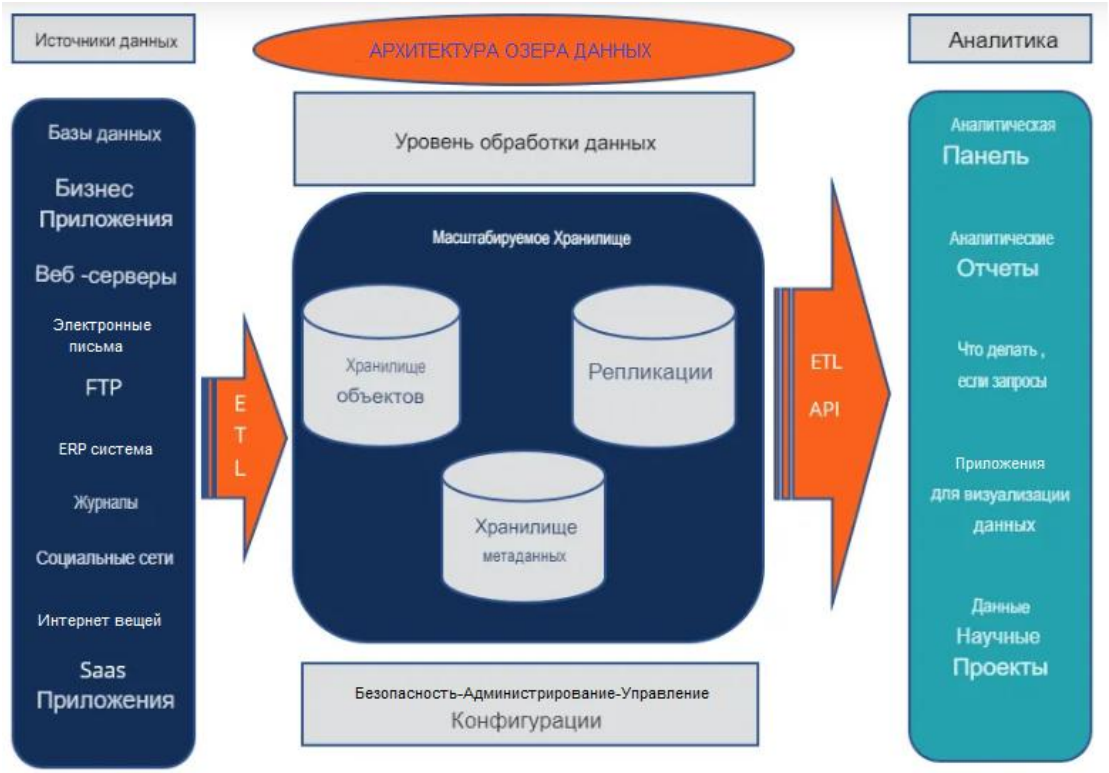


Рис. 2. Архитектура озера данных

Озера данных централизуют хранение данных в естественном формате и подходят как для OLAP, так и для оперативного анализа с технологиями

больших данных. Они применяют вариант «schema-on-read» (схема при чтении) — данные анализируются в их исходном формате, что более гибко. Озера поддерживают разнообразные нагрузки с помощью разных фрейворков и предлагают экономически эффективную архитектуру благодаря открытым программным решениям.

Хранилища данных и озера данных служат для хранения и управления данными, но имеют ключевые различия: хранилища данных работают со структурированными данными и предварительно обрабатывают их с помощью ETL, чтобы обеспечить быструю аналитику и отчетность, в то время как озера данных могут хранить данные в их сыром виде, включая неструктурированные форматы, используя ELT-подход для гибкого анализа и обработки больших объемов данных [6,7]. Хранилища данных более дорогостоящи из-за необходимости структурирования, тогда как озера данных обладают большей гибкостью и масштабируемостью, что делает их подходящими для исследований и анализа больших данных.

## Проблемы перехода в озеро данных

Основные проблемы переноса данных из хранилищ в озера данных включают разнообразие форматов и неоднородность данных, сложность в обеспечении безопасности и масштабируемости, а также затраты на внедрение и обслуживание [8]. Облачные решения и платформы с открытым исходным кодом, такие как Hadoop, могут снизить затраты, но требуют квалифицированной рабочей силы. Управление озерами данных также связано с обеспечением масштабируемости, безопасности и эффективного использования инфраструктуры. В контексте DLS (Data Lake Storage – хранилище озера данных) безопасность и нехватка вычислительных ресурсов остаются серьёзными вызовами.

Для повышения эффективности рекомендуется использовать облачные технологии, унифицировать хранилища данных и применять технологии

извлечения и очистки данных, такие, как CLAMS (Clustered Learning Algorithm for Multi-task and Sequential tasks – кластеризованный алгоритм обучения для многозадачных и последовательных задач) [9]. Интеграция методов анализа и применение машинного обучения, например, KNN и логистической регрессии, важны для оптимизации обработки данных и управления метаданными. Система управления версиями данных, такая как DataHub, помогает в управлении эволюцией схем данных.

## Архитектурные модели озёр данных

Озёра данных разрабатываются с учётом различных моделей для удовлетворения специфических требований к хранению, управлению и анализу данных. Основные модели [10]:

*Raw Data Lake:* Хранение данных в необработанном виде для гибкости анализа.

*Staging Data Lake:* Предварительная обработка и структурирование данных для последующего анализа.

*Curated Data Lake:* Обработанные и классифицированные данные для конкретных аналитических задач.

*Multi-Zone Data Lake:* Разделение данных на необработанные, обработанные и аналитические зоны для управления жизненным циклом и доступа.

*Data Lakehouse:* Гибридная модель, объединяющая элементы хранилищ данных и озёр данных с поддержкой транзакций и SQL-запросов.

*Lambda:* Параллельная обработка в реальном времени и пакетах [11].

*Kappa:* Полностью потоковая обработка данных, минимизирующая пакетную обработку.

Выбор модели зависит от задач, типов данных, их объёмов и частоты обновления в организации. Эти модели можно комбинировать и адаптировать по мере изменения бизнес-целей и технологий.

## Форматы обработки больших данных в озерах

Обычно озера данных создаются на базе платформы Apache Hadoop с файловой системой HDFS, которая экономичнее по сравнению с коммерческими БД. Среди популярных форматов в Hadoop выделяют [12]:

Apache Avro широко используется для сериализации данных и поддерживает эволюцию схемы, что делает его гибким в потоковых системах.

JSON – это текстовый формат с пар ключ-значение, часто используемый в веб-сервисах и документных NoSQL базах.

CSV – простой текстовый формат для табличных данных, популярный в научных и деловых приложениях.

Apache Parquet – бинарный колоночно-ориентированный формат, который эффективен для аналитических задач благодаря сложной структуре хранения.

Apache ORC – оптимизированный формат, схожий с Parquet, используемый для больших потоков данных.

Каждому формату присущи свои подходы и ограничения, определяющие его применение в системах больших данных, аналитике и других сценариях.

## Методы и технологии обработки больших данных

Методы и технологии обработки больших данных включают использование разных инструментов для хранения, управления и анализа больших данных [13]. Ключевые технологии хранения данных включают Apache Hadoop для распределенного хранения и облачные решения, такие как Amazon S3 и Azure Data Lake Storage. Для обработки данных

используются Apache Spark и Apache Flink, а Presto/Trino обеспечивает интерактивный анализ.

***Управление данными*** осуществляется через системы, такие как Apache Hive и Apache Hudi, обеспечивающие транзакционность. Для подготовки и интеграции данных применяются Apache NiFi и Apache Kafka. Аналитика и машинное обучение возможны с Apache Mahout, TensorFlow и PyTorch. Безопасность и контроль доступа поддерживаются Apache Ranger и Apache Sentry, а оркестрация рабочих процессов – Apache Airflow и Luigi. Эти технологии обеспечивают масштабируемое, гибкое и функциональное управление данными.

## Примеры обработки больших данных

Рассмотрим примеры обработки данных, машинного обучения и детализируем исходные данные, результаты и характеристику алгоритмов.

***Пример обработки данных с использованием Apache Spark***

Apache Spark – популярная платформа для распределённой обработки больших данных [14] в кластере, поддерживающая языки программирования, включая Python, Java и Scala. В примере используем PySpark (Python API для Apache Spark) для обработки CSV-файла с данными о пользователях.

Убедимся, что установлены Apache Spark и PySpark. Затем выполним следующий код:

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
# Создаем SparkSession
spark = SparkSession.builder \
    .appName("User Data Processing") \
    .getOrCreate()
# Путь к CSV файлу
csv_file_path = "path/to/users.csv"
# Убедимся, что изменили на путь к файлу
```

```
    # Загружаем CSV данные в DataFrame
    df = spark.read.csv(csv_file_path, header=True,
inferSchema=True)
    # Фильтруем пользователей старше 18 лет
    filtered_df = df.filter(col("age") > 18)
    # Группируем данные по странам и считаем число пользователей
для каждой страны
    result_df = filtered_df.groupBy("country").count()
    # Показываем результат
    result_df.show()
    # Завершаем SparkSession
    spark.stop()
```

Код позволяет обрабатывать данные о пользователях, фильтровать их по возрасту и агрегировать данные по странам проживания.

Пусть есть список пользователей с возрастами и странами (`users.csv`):

```
name,age,country
Alice,25,Greece
Ivan,17,Russia
Judy,40,Turkey
      …
```

Результат – список стран и число пользователей в каждой из них.

Достоинства: Apache Spark обеспечивает высокую производительность благодаря распределенной обработке данных. Поддерживает большой набор языков программирования и может обрабатывать данные разных форматов. Обладает возможностями для интеграции с другими инструментами Apache.

Ограничения: Требуется настройка и управление кластером, что увеличивает сложность использования. Высокий объем ресурсов может быть необходим для обработки действительно больших данных. Может быть избыточным для простых задач.

***Пример машинного обучения с Apache Spark***

Используем Spark Mllib для логистической регрессии. Предположим, есть набор данных с именами столбцов feature1, feature2, и label. Подготовим данные, обучим модель, сделаем предсказания и оценим точность.

Пример кода:

```
# Импорт библиотек
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
# Создание SparkSession
spark = SparkSession.builder \
    .appName("Logistic Regression Example") \
    .getOrCreate()
# Пример данных
data = [
    (0.0, 0.0, 0.0),
    (1.0, 1.0, 1.0),
    (2.0, 2.0, 0.0),
    (3.0, 3.0, 1.0),
    (4.0, 4.0, 0.0),
    (5.0, 5.0, 1.0)
]
# Создание DataFrame
columns = ["feature1", "feature2", "label"]
df = spark.createDataFrame(data, columns)
# Векторизация признаков
assembler = VectorAssembler(inputCols=["feature1", "feature2"], outputCol="features")
df = assembler.transform(df)
# Разделение данных на обучающую и тестовую выборки
```

```
    train_data, test_data = df.randomSplit([0.7, 0.3])
    # Создание и обучение модели логистической регрессии
    lr = LogisticRegression(featuresCol="features",
labelCol="label", maxIter=10)
    lr_model = lr.fit(train_data)
    # Предсказание на тестовой выборке
    predictions = lr_model.transform(test_data)
    # Оценка модели
    evaluator =
MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="accuracy")
    accuracy = evaluator.evaluate(predictions)
    print(f"Точность модели: {accuracy}")
    # Завершение сессии Spark
    spark.stop()
```

Предсказание и оценка: Используем тестовые данные для получения предсказаний и оцениваем точность с помощью метрики точности.

Вывод может выглядеть следующим образом:

```
+---------+-----+----------+
| features|label|prediction|
+---------+-----+----------+
|[2.0, 2.0]|  0.0 |  0.0    |
|[3.0, 3.0]|  1.0 |  1.0    |
|[4.0, 4.0]|  0.0 |  1.0    |  # Плохое предсказание
+---------+-----+----------+
```

Оценка модели на тестовых данных: Точность модели: 0.66, например, если модель правильно классифицировала 2 из 3 образцов.

Фактические значения зависят от случайного разделения данных на обучающие и тестовые наборы каждый раз при запуске кода.

Достоинства: Spark MLlib позволяет обрабатывать большие объемы данных для обучения моделей благодаря распределенной архитектуре. Подходит для масштабируемых вычислений и обработки больших наборов данных. Поддерживает широкий набор алгоритмов машинного обучения.

Ограничения: Ограниченное количество алгоритмов по сравнению с более специализированными библиотеками, такими как scikit-learn. Может уступать в производительности специализированным библиотекам на малых наборах данных. Высокие требования к ресурсам при больших данных.

## Предобработка неструктурированных и полуструктурированных данных

Предобработка таких данных в озёрах требует специализированных подходов, поскольку они не так легко поддаются организации и анализу, как структурированные данные. Полуструктурированные данные, такие как XML, JSON и HTML, обладают некоторыми организующими элементами, что делает их более подходящими для анализа.

***Подходы для полуструктурированных данных:***

NoSQL базы данных: Эффективны в хранении и обработке полуструктурированных данных. Системы потоковой обработки: Например, Apache Kafka для реального времени. Аналитические сервисы озера данных: Amazon Athena и Google BigQuery позволяют выполнять запросы SQL над полуструктурированными данными.

***Неструктурированные данные*** требуют более сложных методов, таких как машинное обучение и специализированные системы для обработки текста и изображений. Hadoop и Spark предоставляют возможности для сложной обработки в распределенных системах.

Ключевые технологии: Data Lake as a Service (DaaS): Управление данными и аналитикой в облаке [15]. Каталогизация и безопасность: Метаданные помогают организовать и обеспечить доступ к данным.

Примеры применения: Hadoop для анализа логов веб-серверов, использующего MapReduce для парсинга и агрегации данных. Spark для анализа тональности текстов с использованием MLlib для предварительной

обработки и классификации данных. Эти инструменты позволяют эффективно извлекать ценную информацию из большого объёма данных.

***Пример кода на Python***, иллюстрирует, как ***удалить шум и выполнить токенизацию*** текста.

Воспользуемся библиотекой NLTK [16] для токенизации и регулярными выражениями для удаления шума. Определим функцию для очистки текста от шума и токенизации:

```
import re
from nltk.tokenize import word_tokenize
import nltk
nltk.download('punkt')  # Необходимо для первого использования word_tokenize
def preprocess_text(text):
    # Удаление HTML тегов
    text = re.sub(r'<.*?>', '', text)
        # Удаление email
    text = re.sub(r'\S*@\S*\s?', '', text)
        # Удаление URL
    text = re.sub(r'http\S+', '', text)
    text = re.sub(r'www\S+', '', text)
        # Удаление чисел и символов, не являющихся словами
    text = re.sub(r'[^a-яА-Яа-zA-Z\s]', '', text)
        # Приведение к нижнему регистру
    text = text.lower()
        # Токенизация
    tokens = word_tokenize(text)
        return tokens
# Пример текста
text = "Привет! Этот пример поможет вам понять, как удалить шум из текста \
и разбить его на слова. Посещайте наш сайт: https://example.com или напишите \
```

```
нам на contact@example.com."
# Обработка текста
processed_text = preprocess_text(text)
# Вывод обработанного текста
print(processed_text)
```

В примере удаляются HTML-теги, email-адреса, URL, числа и символы, не являющиеся буквами. Текст приводится к нижнему регистру для унификации. Применяется токенизация с использованием функции word_tokenize из NLTK, разбивает текст на отдельные слова (токены).

## Заключение

Переход университетов от хранилищ к озерам данных становится актуальным из-за роста объемов информации и необходимости ее эффективного использования. Озера данных более гибкие и масштабируемые, поддерживают хранение данных в исходном формате и упрощают интеграцию различных источников для аналитики. Использование технологий, таких как Apache Hadoop и Apache Spark, позволяет справляться с большими объемами данных и применять инновационные аналитические техники. Это помогает в научных исследованиях и принятии решений.

Внедрение озер данных сопряжено с вызовами в области безопасности, управления метаданными и обеспечения производительности. Облачные технологии снижают инфраструктурные затраты и обеспечивают гибкость масштабирования. Правильный выбор архитектуры и инструментов повышает продуктивность в управлении данными и аналитике.

## Литература

1. Harby A. A., Zulkernine F. From data warehouse to lakehouse: A comparative review // 2022 IEEE International Conference on Big Data (BigData). 2022. С. 834-843.

2. Nambiar A., Mundra D. An overview of data warehouse and data lake in modern enterprise data management // Big Data and Cognitive Computing. 2022. Т. 6. № 4. С. 132.

3. Рахеб А. Х., Кириенко А. Б., Хомоненко А. Д. Метод перехода от хранилищ данных к озерам данных геоинформационных систем на основе Лямбда-архитектуры // Интеллектуальные технологии на транспорте. 2024. № 1 (37). С. 45-55.

4. Neamah A. F. Adoption of data warehouse in university management: Wasit University case study // Journal of Physics: Conference Series. 2021. Т. 1860. № 1. С. 012027.

5. Пучков Е. В., Пономарева Е. И. Разработка информационно-аналитической системы на основе многомерного хранилища данных // Инженерный вестник Дона. 2012. № 4 (часть 1). URL: ivdon.ru/ru/magazine/archive/n4p1y2012/1123.

6. Смирнов В. А., Андреева А. Р., Севанян А. В. Особенности концепции хранения данных-озеро данных // Проблемы эффективного использования научного потенциала общества. 2019. С. 55-57.

7. Farid M., Hassan S., Alhazmi H., Shamsuddin S. CLAMS: Bringing quality to data lakes // Proceedings of the 2016 International Conference on Management of Data. 2016. С. 2089-2092.

8. Hlupić Т., Aydın S., Kapetanović М., Adedayo Т., Peršić А., Zovko D., Očić S. An overview of current data lake architecture models // 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO). 2022. С. 1082-1087.

9. Белов В. А., Никульчев Е. В. Инструменты анализа форматов хранения больших данных для построения озер данных // Управление развитием крупномасштабных систем (MLSD'2021). 2021. С. 1501-1504.

10. Giebler C., Gröger C., Hoos E., Schwarz H., Mitschang B., Sadowski B., Schäfer B., Decker A., Wenzel M. Leveraging the data lake: Current state and challenges // Big Data Analytics and Knowledge Discovery: 21st International Conference, DaWaK, Linz, Austria, August 26–29, 2019, Proceedings 21. Springer International Publishing. 2019. С. 179-188.

11. Khine P. P., Wang Z. S. Data lake: A new ideology in big data era // ITM Web of Conferences. 2018. Т. 17. С. 03025.

12. Aundhkar A. A review on enterprise data lake solutions // Journal of Science & Technology (JST). 2021. Т. 6. Special Issue 1. С. 11-14.

13. Zagan E., Danubianu M. Cloud DATA LAKE: The new trend of data storage // 2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA). 2021. С. 1-4.

14. Yang C. T., Zhang Y., Kim H. The implementation of data storage and analytics platform for big data lake of electricity usage with spark // The Journal of Supercomputing. 2021. Т. 77. № 6. С. 5934-5959.

15. Воробьев С. П., Горобец В. В. Исследование модели транзакционной системы с репликацией фрагментов базы данных, построенной по принципам облачной среды // Инженерный вестник Дона. 2012. № 4 (часть 1). URL: ivdon.ru/ru/magazine/archive/n4p1y2012/1149.

16. Cherradi M., El Haddadi A. Enhancing Data Lake Management Systems with LDA Approach // Journal of Data Science and Intelligent Systems. 2024. Т. 00. Выпуск 00. С. 1-9.

## References

1. Harby A. A., Zulkernine F. 2022 IEEE International Conference on Big Data (BigData). 2022. Pp. 834-843.

2. Nambiar A., Mundra D. Big data and cognitive computing. 2022. Vol. 6. No. 4. P. 132.

3. Rakheb A. Kh., Kirienko A. B., Khomonenko A. D. Intellektual`ny`e texnologii na transporte. 2024. No. 1 (37). Pp. 45-55. DOI: 10.20295/2413-2527-2024-137-45-55.

4. Neamah A. F. Journal of Physics: Conference Series. 2021. Vol. 1860. No. 1. P. 012027.

5. Puchkov E. V., Ponomareva E. I. Inzhenernyj vestnik Dona. 2012. No. 4 (part 1). URL: ivdon.ru/ru/magazine/archive/n4p1y2012/1123.

6. Smirnov V. A., Andreeva A. R., Sevanyan A. V. Problemy` e`ffektivnogo ispol`zovaniya nauchnogo potenciala obshhestva. 2019. Pp. 55-57.

7. Farid M., Hassan S., Alhazmi H., Shamsuddin S. Proceedings of the 2016 International Conference on Management of Data. 2016. Pp. 2089-2092.

8. Hlupić T., Aydın S., Kapetanović M., Adedayo T., Peršić A., Zovko D., Očić S. 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO). 2022. Pp. 1082-1087.

9. Belov V. A., Nikul'chev E. V. Upravlenie razvitiem krupnomasshtabny`x sistem (MLSD'2021). 2021. Pp. 1501-1504.

10. Giebler C., Gröger C., Hoos E., Schwarz H., Mitschang B., Sadowski B., Schäfer B., Decker A., Wenzel M. Springer International Publishing. 2019. Pp. 179-188.

11. Khine P. P., Wang Z. S. ITM Web of Conferences. 2018. Vol. 17. P. 03025.

12. Aundhkar A. Journal of Science & Technology (JST). 2021. Vol. 6. Special Issue 1. Pp. 11-14.

13. Zagan E., Danubianu M. 2021 3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA). 2021. Pp. 1-4.

14. Yang C. T., Zhang Y., Kim H. The Journal of Supercomputing. 2021. Vol. 77. No. 6. Pp. 5934-5959.

15. Vorobyev S. P., Gorobets V. V. Inzhenernyj vestnik Dona. 2012. №4 (part 1). URL: ivdon.ru/ru/magazine/archive/n4p1y2012/1149.

16. Cherradi M., El Haddadi A. Journal of Data Science and Intelligent Systems. 2024. Vol. 00. Issue 00. Pp. 1-9.

**Дата поступления: 19.10.2024**

**Дата публикации: 5.12.2024**