Russian Technological Journal.
2026;14(1):7–18

ISSN 2782-3210 (Print)
ISSN 2500-316X (Online)

УДК 004.042
https://doi.org/10.32362/2500-316X-2026-14-1-7-18
EDN TAUPKU

НАУЧНАЯ СТАТЬЯ

Методы приоритизации процессов переноса данных в центральное хранилище

Д.А. Пушкарев [1, @], В.А. Богатырев [1, 2]

[1] *Национальный исследовательский университет ИТМО, Санкт-Петербург, 197101 Россия*
[2] *Санкт-Петербургский государственный университет аэрокосмического приборостроения, Санкт-Петербург, 190000 Россия*
[@] *Автор для переписки, e-mail: dpushkarev@itmo.ru*

• **Поступила:** 10.06.2025 • **Доработана:** 28.07.2025 • **Принята к опубликованию:** 12.11.2025

Резюме

Цели. Эффективное управление параллельным выполнением ETL[1]-процессов в центральных хранилищах данных критически влияет на общее время обработки. Существующие инструменты оркестрации Apache Airflow, NiFi, Luigi используют упрощенные алгоритмы приоритизации, игнорирующие топологию графов зависимостей и динамику ресурсов, что приводит к субоптимальному планированию. Целью данной работы являются разработка и валидация нового метода приоритизации задач в рамках ETL-конвейеров, направленного на минимизацию их общей длительности за счет глубокого анализа структурных особенностей направленных ациклических графов и использования имитационного моделирования для оценки различных стратегий планирования в условиях конкуренции за ограниченные слоты параллелизма.

Методы. Предложена имитационная модель на языке Python, воспроизводящая выполнение ETL-процессов в среде с ограниченными слотами параллелизма. Модель генерирует направленный ациклический граф, отражающий структуру связей процессов для формирования центрального хранилища данных, и сравнивает 9 алгоритмов приоритизации, включая базовые (приоритизация минимального/максимального среднего времени выполнения), топологические (приоритизация минимального/максимального уровня слоя, максимизация числа зависимостей) и гибридные (разделение слотов на очереди для минимального и максимального времени выполнения). Эксперименты проведены на графах различных топологий на основе полученной имитационной модели.

Результаты. Гибридный алгоритм (разделение слотов: 50% для задач с максимальным временем выполнения, 50% – с минимальным) показал наилучшую эффективность: снижение общего времени выполнения на 15–17% по сравнению с базовыми алгоритмами, минимизацию времени простоя задач на 20–25% и устойчивость к вариациям топологии графов. Линейная комбинация с оптимизированными коэффициентами (время выполнения – наиболее значимый фактор) заняла второе место по эффективности.

[1] Извлечение (extract), преобразование (transform) и загрузка (load) данных.

Выводы. Приоритизация на основе анализа DAG[2]-топологии и гибридных стратегий существенно сокращает время выполнения ETL-конвейеров. Гибридный алгоритм рекомендуется для внедрения в оркестраторы как балансирующий минимизацию длительности конвейера и времени простоя задач. Перспективное направление – адаптивные алгоритмы, учитывающие динамическую загрузку ресурсов в реальном времени.

Ключевые слова: оркестрация ETL-процессов, имитационное моделирование, графы зависимостей, хранилище данных, ациклический направленный граф

Для цитирования: Пушкарев Д.А., Богатырев В.А. Методы приоритизации процессов переноса данных в центральное хранилище. *Russian Technological Journal*. 2026;14(1):7–18. https://doi.org/10.32362/2500-316X-2026-14-1-7-18, https://www.elibrary.ru/TAUPKU

Прозрачность финансовой деятельности: Авторы не имеют финансовой заинтересованности в представленных материалах или методах.

Авторы заявляют об отсутствии конфликта интересов.

RESEARCH ARTICLE

Methods for prioritizing the processes of transferring data to central storage

Daniil A. Pushkarev [1, @], Vladimir A. Bogatyrev [1, 2]

[1] *ITMO University, Saint Petersburg, 197101 Russia*
[2] *Saint Petersburg State University of Aerospace Instrumentation (SUAI), Saint Petersburg, 190000 Russia*
[@] *Corresponding author, e-mail: dpushkarev@itmo.ru*

• **Submitted:** 10.06.2025 • **Revised:** 28.07.2025 • **Accepted:** 12.11.2025

Abstract

Objectives. The efficient management of parallel ETL (Extract, Transform, Load) process execution in central data warehouses critically impacts overall processing time. Existing orchestration tools such as Apache Airflow, NiFi, Luigi employ simplified prioritization algorithms which ignore dependency graph topology and resource dynamics, leading to suboptimal scheduling. The objective of this work is to develop and validate a novel task prioritization method for ETL pipelines, aimed at minimizing their total duration through deep analysis of structural features of Directed Acyclic Graphs (DAGs), as well as the use of simulation modeling to evaluate various scheduling strategies under conditions of competition for limited concurrency slots.

Methods. The study proposed a Python simulation model, replicating ETL process execution in an environment with limited concurrency slots. The model generates a DAG which reflects the dependency structure of processes for building a central data warehouse and compares 9 prioritization algorithms. These include basic algorithms (prioritization by minimum/maximum average execution time), topological algorithms (prioritization by minimum/maximum layer level, maximization of dependency count), and hybrid algorithms (splitting slots into queues for minimum and maximum execution time). Experiments were conducted on graphs of a variety of topologies using the developed simulation model.

Results. The hybrid algorithm (slot allocation: 50% for tasks with maximum execution time, 50% for tasks with minimum execution time) demonstrated the highest level of efficiency. It reduced total execution time by 15–17%, when compared to basic algorithms, minimized task idle time by 20–25%, and showed resilience to graph topology variations. A linear combination of optimized coefficients (execution time being the most significant factor) ranked second in terms of efficiency.

[2] Directed acyclic graph – ориентированный ациклический граф.

Conclusions. Prioritization based on DAG topology analysis and hybrid strategies significantly reduces ETL pipeline execution time. The hybrid algorithm is recommended for implementation in orchestrators, since it balances minimizing pipeline duration and task idle time. A promising area for further study is the development of adaptive algorithms that account for real-time dynamic resource load.

Keywords: ETL orchestration, simulation modeling, dependency graphs, data warehouse, directed acyclic graph

For citation: Pushkarev D.A., Bogatyrev V.A. Methods for prioritizing the processes of transferring data to central storage. *Russian Technological Journal.* 2026;14(1):7–18. https://doi.org/10.32362/2500-316X-2026-14-1-7-18, https://www.elibrary.ru/TAUPKU

Financial disclosure: The authors have no financial or proprietary interest in any material or method mentioned.

The authors declare no conflicts of interest.

ВВЕДЕНИЕ

С каждым годом количество данных, проходящих через различные приложения, растет. Сбор и хранение этих данных стали одной из главных задач, стоящих перед разработчиками хранилищ данных. Для решения подобной задачи были предложены центральные хранилища данных (ЦХД) [1, 2], представляющие собой распределенные хранилища, куда попадают данные из всех сервисов с целью дальнейшего анализа и получения дополнительной информации, из которой бизнес может извлечь выгоду. Таким образом ЦХД становятся центром принятия всех бизнес-решений [3].

Наполнение хранилища осуществляется посредством процессов извлечения (extract), преобразования (transform) и загрузки (load) данных – ETL [2, 4–7]. Эти процессы представляют собой комплексные операции, направленные на интеграцию данных из разнообразных источников, включая реляционные базы данных, файловые системы и веб-серверы. На первом этапе происходит извлечение данных, что подразумевает сбор информации из указанных источников. Далее данные подвергаются процессу преобразования, в ходе которого осуществляется их структуризация, нормализация и очистка с целью повышения качества и согласованности информации. Завершающим этапом является загрузка подготовленных данных в целевое хранилище, что обеспечивает их доступность для последующего анализа и использования.

В современных системах обработки данных ETL-процессы играют ключевую роль в подготовке информации. Однако при масштабировании инфраструктуры и увеличении числа таких процессов возникает проблема эффективного управления их выполнением. Когда десятки или сотни ETL-задач взаимодействуют в рамках сложной архитектуры, ручное определение порядка их запуска становится не только трудоемким, но и рискует привести к субоптимальным результатам. Последовательность выполнения процессов напрямую влияет на общее время обработки данных: конкуренция за вычислительные ресурсы, каскадные задержки из-за зависимостей между задачами и неоптимальное распределение нагрузки могут многократно увеличить длительность выполнения всего конвейера процессов. Например, задержка в процессе, предоставляющем данные для нескольких последующих этапов, способна вызвать цепную реакцию простоев. Для решения этой задачи необходимо не только внедрение методов приоритизации, но и четкое моделирование архитектуры ETL-системы.

Сложные взаимосвязи между процессами, включающие как явные зависимости данных, когда выход одного задания служит входом другого, так и скрытые ограничения (ресурсы памяти, пропускная способность сети), могут быть визуализированы через направленные ациклические графы (directed acyclic graphs, DAG) [8].

Направленные ациклические графы зарекомендовали себя в качестве стандартного инструмента для моделирования процессов извлечения, преобразования и загрузки данных благодаря своим ключевым свойствам, обеспечивающим надежность, эффективность и прозрачность обработки данных [2, 7, 8]. Ацикличность структуры гарантирует отсутствие бесконечных циклов, что является критически важным для обеспечения завершенности задач в конвейерах обработки данных, особенно при работе с большими объемами информации. Структура данных графов позволяет контролировать выполнение зависимых друг от друга задач, что важно при построении процессов наполнения хранилища данных (например, извлечение данных всегда предшествует их преобразованию). Кроме того, направленный ациклический граф помогает эффективно работать с параллельными задачами благодаря своей структуре, которая позволяет визуализировать зависимости между различными этапами обработки данных. Таким образом, представленные в графе задачи независимы и могут выполняться одновременно.

Инструменты оркестрации, такие как Apache Airflow [9–11], предоставляют базовые механизмы для определения зависимостей, однако эффективная приоритизация требует более детального анализа – учета динамически меняющейся нагрузки, критичности бизнес-метрик и предсказания узких мест. Без системного подхода к описанию архитектуры и формализации правил упорядочивания процессы рискуют выполняться в последовательности, далекой от оптимальной, что особенно критично в условиях аналитики, приближенной к реальному времени.

Несмотря на значительную практическую важность оптимизации ETL-конвейеров, количество специализированных исследований по приоритизации задач в этой области остается ограниченным. Большинство существующих оркестраторов[3, 4] (Apache Airflow, NiFi, Luigi) [6, 10–12] используют упрощенные стратегии, такие как first-in-first-out (первым пришел – первым ушел) или статические приоритеты, проставленные пользователем, основанные исключительно на его эмпирическом опыте. Значительные достижения существуют в смежной сфере систем реального времени, где модель DAG также широко используется [13]. Данные исследования решают задачу гарантированного соблюдения жестких дедлайнов для параллельных задач на многопроцессорных системах, используя концепцию параллельного прогрессирования выбранных путей в графе с приоритизацией подзадач и сложным анализом наихудшего времени отклика.

Несмотря на общую основу – направленные ациклические графы, цели и условия ETL-оптимизации принципиально отличаются от планирования в реальном времени. Во-первых, ETL фокусируется на минимизации среднего времени выполнения конвейера и простоев задач при гибкой параллельности, тогда как системы реального времени требуют гарантий соблюдения дедлайнов при жестком выделении ресурсов. В качестве метрики для систем реального времени могут использоваться вероятность выполнения запросов за заданное время и коэффициент готовности к выполнению запросов за заданное время [14, 15]. Во-вторых, метрики эффективности различаются: для ETL критичны общая длительность и утилизация ресурсов, а не анализ наихудшего случая.

Прямое применение методов повышения эффективности систем реального времени (включая алгоритмы выбора путей) в ETL-контексте неоптимально. Их избыточная сложность и требование резервирования ведут к недоиспользованию ресурсов, игнорируя специфику слоев хранилищ данных и динамические ограничения оркестраторов. Статические подходы обеспечения реального времени плохо адаптируются к изменениям нагрузки, где гибридные стратегии показывают лучшую эффективность. Таким образом, несмотря на формальную общность моделей, различие операционных требований и метрик делает необходимым разработку специализированных методов приоритизации для ETL, что и обосновывает данное исследование.

Анализ существующих методов ETL-приоритизации [16, 17] показывает значительные ограничения современных подходов:

- предлагаемые алгоритмы не учитывают топологические характеристики графов процессов, что негативно сказывается на эффективности планирования в распределенных системах;
- преобладающим остается статический метод приоритизации, основанный на пользовательских метках для отдельных задач, который не принимает во внимание динамику выполнения и взаимное влияние процессов в конвейере.

Целью данной работы являются разработка и обоснование метода приоритизации ETL-процессов, обеспечивающего минимизацию времени выполнения всего конвейера данных за счет методов системного анализа и имитационного моделирования.

В рамках данного исследования предлагается имитационная модель, позволяющая анализировать управление ETL-процессами в условиях сложной архитектуры. Модель воссоздает среду, в которой генерируется направленный ациклический граф, отражающий цепочки выполнения ETL-задач, формирующий теоретическое хранилище данных. Каждый направленный ациклический граф моделирует зависимости между процессами и временные ограничения. Ключевая особенность подхода – возможность сравнивать различные алгоритмы приоритизации в идентичных условиях, что обеспечивает объективную оценку их эффективности. Это позволяет не только визуализировать выполнение процессов в рамках централизованного хранилища данных, но и количественно оценивать влияние выбранного алгоритма на общую длительность конвейера. Например, варьируя количество узлов, глубину зависимостей или длительность выполнения отдельной задачи, можно определить, при каких сценариях какой алгоритм демонстрирует лучший результат. Результаты работы модели формируют метрики для сравнения – среднее время выполнения, процент простоев ресурсов, что формирует основу для обоснованного выбора метода оптимизации в реальных условиях. Таким образом, имитационная модель

[3] https://nifi.apache.org/. Apache NiFi. Apache Software Foundation. Дата обращения 10.01.2025. / Accessed January 10, 2025.

[4] https://www.informatica.com/. Informatica. Informatica LLC. Дата обращения 12.01.2025. / Accessed January 12, 2025.

выступает инструментом для системного анализа ETL-оркестрации, минимизируя риски внедрения неэффективных решений в промышленные системы.

1. ФОРМАЛИЗАЦИЯ ПРОБЛЕМЫ

В рамках данной работы формализуем задачу оркестрации ETL-процессов и установим ряд ограничений, необходимых для ее решения. Задача оркестрации включает в себя несколько ключевых этапов: определение приоритетов для задач, контроль за количеством одновременно выполняемых процессов, а также управление изменением статусов задач в соответствии с их текущим состоянием.

Следует подчеркнуть, что в исследовании не будет проводиться анализ загрузки целевой вычислительной системы. Вместо этого для регулирования нагрузки на систему будет введена переменная, определяющая максимальное количество одновременно активных задач – слотов. Данный подход является распространенным в аналогичных инструментах; например, в Apache Airflow существует параметр количества доступных вычислительных слотов[5], который отвечает за общее количество задач, способных выполняться одновременно в рамках всех процессов. Кроме того, будет установлено дополнительное ограничение: при запуске конкретного процесса его выполнение не может быть приостановлено до момента завершения данного процесса. Ресурсы станут доступны только по завершении его работы.

Сформулируем задачу оптимизации для процесса оркестрации. Пусть $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ представляет собой ориентированный ациклический граф, где $\mathbf{V}$ – множество процессов, а $\mathbf{E}$ – множество зависимостей между ними. В случае, когда максимальное количество одновременно выполняемых процессов ограничено единицей, задача может быть решена с использованием классического метода – топологической сортировки. Топологическая сортировка является линейным упорядочением вершин ориентированного ациклического графа, при котором для каждого направленного ребра от вершины A к вершине B выполняется условие, что A предшествует B в данном упорядочении. Применение топологической сортировки к графу $\mathbf{G}$ позволяет получить последовательность выполнения процессов, которая удовлетворяет всем установленным зависимостям.

Для данной задачи мы можем использовать следующие обозначения: $\mathbf{P} = \{p_1, p_2, \ldots, p_n\}$ – множество процессов; $\mathbf{D}(p_i)$ – множество процессов, от которых зависит процесс p_i; t_i – время выполнения каждого процесса.

[5] https://airflow.apache.org/docs/apache-airflow/stable/configurations-ref.html. Airflow Configuration Reference. Дата обращения 20.01.2025. / Accessed January 20, 2025.

Цель состоит в минимизации общего времени выполнения всех процессов с учетом их зависимостей. Эту задачу оптимизации можно записать следующим образом:

$$\text{minimize}\, T_{\text{total}} = \sum_{i=1}^{n} t_i, \qquad (1)$$

при условии, что для каждого процесса p_i справедливо условие: p_j выполняется перед p_i, если $p_j \in \mathbf{D}(p_i)$.

Но при такой постановке задачи, где одновременно может быть запущен только один процесс, получаем единственное решение, представляющее собой сумму времен выполнения всех процессов:

$$T_{\text{total}} = \sum_{i=1}^{n} t_i.$$

В случае, если может быть запущено одновременно (параллельно) N процессов, такое решение может быть не единственным. При такой постановке новую задачу оптимизации можно описать следующим образом:

1. Определим множество активных процессов $\mathbf{A}(t)$, которые могут быть запущены в момент времени t.
2. Пусть $C(t) \le N$ – количество процессов, которые могут быть запущены в момент времени t.
3. Время выполнения всех процессов теперь будет зависеть от параллельности их выполнения.

Общая цель остается прежней – минимизация общего времени выполнения (1).

2. ИМИТАЦИОННОЕ МОДЕЛИРОВАНИЕ ОРКЕСТРИРОВАНИЯ

Сравнение различных алгоритмов приоритизации на реальных данных в условиях вычислительного кластера может представлять собой значительные трудности по нескольким причинам. Во-первых, продолжительность отдельных экспериментов может достигать нескольких часов. Во-вторых, для запуска множества параллельных процессов требуется значительная вычислительная мощность кластера. В связи с вышеизложенным, целесообразно рассмотреть имитационную модель, которая бы воспроизводила поведение аналогичной системы, минимизируя затраты времени и вычислительных ресурсов.

Моделирование параллельного выполнения задач с ограниченным числом слотов реализовано с использованием подхода дискретно-событийного моделирования с собственным планировщиком событий на базе приоритетной очереди, реализованной на языке Python. Выбор языка Python для реализации

имитационной модели обусловлен совокупностью его преимуществ, отвечающих специфике поставленной исследовательской задачи. Данный язык программирования предоставляет развитую экосистему научных библиотек, которые значительно упрощают реализацию модели. Кроме того, интеграционные возможности Python с инструментами визуализации и средствами обработки данных упрощают валидацию модели и интерпретацию экспериментальных метрик.

Алгоритм работы включает четыре ключевых этапа. Во-первых, формируется список запущенных задач: на этом этапе программа анализирует направленный ациклический граф и добавляет в очередь процессы, зависимости которых уже выполнены (например, родительские узлы в графе завершены или отсутствуют), а их текущий приоритет максимален среди доступных задач. Это реализуется через проверку статуса узлов в направленном ациклическом графе и сортировку по заданным правилам приоритизации (например, на основе значения слоя, на котором выполняется процесс). Во-вторых, определяется минимальное время выполнения среди всех активных задач в списке – это время, за которое хотя бы одна из задач завершится. В-третьих, это значение прибавляется к общему времени выполнения конвейера, таким образом двигая систему на это значение вперед по времени. Наконец, для всех запущенных задач время их оставшейся работы уменьшается на вычисленное минимальное значение, а завершившиеся процессы помечаются как выполненные, обновляя граф зависимостей для следующих итераций. Такой подход эффективно эмулирует конкурентное выполнение задач, учитывая как логические зависимости между ними, так и динамику распределения ресурсов, что обеспечивает гибкость тестирования различных алгоритмов приоритизации в рамках единой модели.

Рассмотрим данные, на которых будут проведены эксперименты. Классическое хранилище данных имеет структуру слоев [1, 7, 18, 19].

Операционное хранилище данных. Хранит операционные, часто текущие детальные данные, которые еще не агрегированы. Этот слой служит источником оперативной информации для создания отчетов и может напрямую использоваться для построения более высокоуровневых слоев.

Слой детализированных данных. Централизованное хранилище интегрированных данных, собранных из одного или нескольких разрозненных источников. Данные структурированы специально для выполнения запросов и анализа.

Витрины данных. Подмножество хранилища данных, сосредоточенное на конкретной области деятельности или команде. Карты данных разрабатываются с учетом специфических потребностей или анализа.

Существуют ситуации, в которых количество слоев может превышать три; данный параметр определяется конкретными условиями применения. Однако наиболее распространенным вариантом реализации является трехслойная архитектура: архитектура, предложенная Ральфом Кимбалллом [1], и архитектура медальона [13, 20].

В рамках данного исследования для обеспечения максимального охвата разнообразных сценариев были сгенерированы направленные ациклические графы. Генерация выполнялась по следующей методике:

- Количество слоев – 3. Такое количество наиболее распространено при формировании хранилища данных.
- Общее число узлов (процессов) для каждого графа генерировалось в соответствии с равномерным распределением в диапазоне от 250 до 400.
- Узлы распределялись по слоям в соответствии с табл. 1.
- Дополнительно внутри слоев детального слоя и витрин данных добавлялись связи, моделирующие сложные преобразования: каждый узел имел до двух случайных предшественников внутри своего слоя (вероятность установления связи равна 0.1, распределение Парето для числа связей). После генерации связей осуществлялась проверка на наличие циклов с целью исключить связи, из-за которых данный цикл возникает. Общее число связей на узел не превышает 10.
- Время выполнения каждого узла назначалось независимо, равномерно распределенным в диапазоне от 100 до 5000 с. Связи между временем выполнения, слоем или степенью узла не устанавливались.

Таблица 1. Доля таблиц относительно слоя

Слой	Доля таблиц
Хранение исходных данных	30–40%
Интегрированное хранилище структурированных данных, готовых для анализа	30–40%
Специализированное подмножество, ориентированное на конкретную предметную область	20–30%

Для оценки эффективности девяти алгоритмов приоритизации каждый алгоритм был запущен на каждом сгенерированном направленном ациклическом графе. Эксперименты проводились для различного количества слотов параллелизма с целью оценки влияния увеличение данного параметра на эффективность алгоритма.

3. АЛГОРИТМЫ ПРИОРИТИЗАЦИИ

Приоритизация задач представляет собой процесс, в рамках которого уровень приоритета каждой отдельной задачи определяется на основе ее характеристик и взаимосвязей с другими задачами. Этот уровень может быть случайным или же формироваться на основании параметров конкретной задачи, ее положения и связей с другими элементами. Рассмотрим несколько возможных алгоритмов приоритизации.

Случайный выбор. Данный алгоритм функционирует по принципу равного приоритета. Это означает, что любой процесс, который может быть запущен в данный момент, будет активирован с равной вероятностью. Такое решение не является оптимальным и скорее представляет собой базовое решение, используемое для сравнения с альтернативными подходами.

Минимальное время выполнения. Сначала выполняются процессы с наименьшим временем выполнения.

Максимальное время выполнения. Сначала выполняются процессы с наибольшим временем выполнения.

Максимизация числа зависимостей. Алгоритм основывается на количестве зависимых от каждого процесса других процессов.

Гибридный алгоритм по времени. Данный подход разделяет доступные вычислительные слоты ETL-системы на две части: первая часть выделяется для выполнения задач с наивысшим приоритетом по стратегии максимального времени выполнения, вторая – с наивысшим приоритетом по стратегии минимального времени выполнения.

Рекурсивная максимизация числа зависимостей. Алгоритм, учитывающий количество зависимых задач не только текущего процесса, но и всех нижестоящих процессов, которые зависят от данного.

Минимальный уровень слоя. Алгоритм, который основывается на уровне слоя, в рамках которого выполняется данный ETL-процесс. Оперативный слой соответствует уровню 0, детальный – уровню 1, а слой витрин данных – уровню 2. При наличии большего количества слоев они классифицируются аналогично в соответствии с топологией графа.

Максимальный уровень слоя. Алгоритм, приоритизирующий процессы с более высоким уровнем слоя.

Линейная комбинация. Данный подход обобщает все параметры в рамках одной модели. Для приоритизации строится линейная комбинация признаков и соответствующих им коэффициентов. В качестве параметров выбираются следующие критерии: время выполнения, количество зависимых процессов и уровень слоя. Полученная линейная комбинация имеет вид:

$$t_i x_1 + p_i x_2 + l_i x_3,$$

где x_1, x_2, x_3 – константы, l_i – уровень слоя i-го процесса.

При использовании данной комбинации необходимо подобрать соответствующие коэффициенты. Для этого были использованы алгоритмы оптимизации, где каждая итерация запускает имитационную модель с новым вектором параметров **X**, получая в результате работы модели общее время выполнения. Данное время необходимо минимизировать.

4. РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТОВ

На основе требований, описанных выше, было сгенерировано 50 графов, которые представляют различные архитектуры ETL-процессов, наполняющих хранилище данными.

Рассмотрим процесс подбора коэффициентов для алгоритма, основанного на линейной комбинации. Для определения оптимальных значений этих коэффициентов было выбрано N графов, которые запускаются с заранее установленными коэффициентами x_i. Задача оптимизационной модели заключается в минимизации общего времени выполнения всех N графов. Таким образом, коэффициенты подбираются не для конкретной модели, а обобщаются для различных топологий графов. Этот подход является необходимым, поскольку в условиях реального применения алгоритма подбор таких параметров может быть невозможным.

В качестве оптимизационных алгоритмов для решения данной задачи были выбраны следующие методы: алгоритм Пауэлла (Powell), метод COBYLA (от англ. Constrained Optimization BY Linear Approximations – ограниченная оптимизация с линейной аппроксимацией), алгоритм Нелдера – Мида (Nelder–Mead) и метод сопряженного градиента (conjugate gradient, CG). Реализация этих алгоритмов взята из библиотеки scipy, модуля optimize[6]. Результаты работы оптимизационных алгоритмов представлены в табл. 2.

Наилучший результат получен с использованием алгоритма Нелдера – Мида (Nelder–Mead). Исходя из полученных коэффициентов, можно сделать вывод, что данный алгоритм выделил время выполнения как наиболее влиятельный параметр для приоритизации процессов.

[6] https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html. Дата обращения 27.01.2025. / Accessed January 27, 2025.

Таблица 2. Результаты решения оптимизационной задачи для линейной комбинации

Модель	Число итераций	Время выполнения	Коэффициенты
Powell	193	32282	2.58792896, 2.66311897, −1.85081307
CG	43	32385	$2.10667215 \cdot 10^{-5}$, $4.67212040 \cdot 10^{-5}$, 0
COBYLA	26	32274	1.1580144, 0.80662082, 1.44364821
Nelder–Mead	**109**	**30193**	**−3.68735705, 0.75631002, 0.5**

Алгоритм был применен к набору графов, не входивших в обучающую выборку. В результате получена общая длительность обработки всех процессов в этих графах при различных значениях параметра числа доступных слотов (рис. 1).

Для оценки оперативности обслуживания задач в очереди введем дополнительную метрику – время простоя задачи. Она вычисляется как суммарное время, проведенное в очереди, готовых к запуску задач. Результаты полученной метрики представлены на рис. 2.

Результаты исследования демонстрируют, что алгоритм приоритизации задач с максимальной длительностью выполнения обеспечивает наилучшее общее время выполнения ETL-процессов только в условиях ограниченного параллелизма, когда число одновременно выполняемых задач (слотов) не больше двух. В таких условиях данный алгоритм минимизирует влияние длительных задач как узких мест, предотвращая их блокирующее воздействие на завершение всей очереди.

Однако при увеличении степени параллелизма гибридный алгоритм, разделяющий очередь на две равные части – для задач с минимальным и максимальным временем выполнения, – превосходит алгоритм, максимизирующий время выполнения по общей длительности выполнения. Это происходит благодаря способности гибридного подхода эффективно распределять ресурсы: выделенные слоты для коротких задач позволяют быстро очищать очередь от мелких операций, в то время как слоты для длительных задач обеспечивают их непрерывное выполнение без простоев. Кроме того, гибридный алгоритм значительно сокращает время простоя задач в очереди по сравнению с максимизацией времени, хотя и уступает в этом показателе алгоритму, приоритизирующему минимальные по времени выполнения задачи.

Что касается противоречия относительно времени простоя, при котором минимизация времени демонстрирует наименьшие значения, а максимизация – наибольшие, это объясняется принципиальным различием в их оптимизационных целях. Алгоритм, ставящий в приоритет задачи с минимальным временем выполнения, обеспечивает их быстрое выполнение, что резко сокращает время ожидания для большинства задач в очереди. Алгоритм, ставящий в приоритет максимальные по времени выполнения

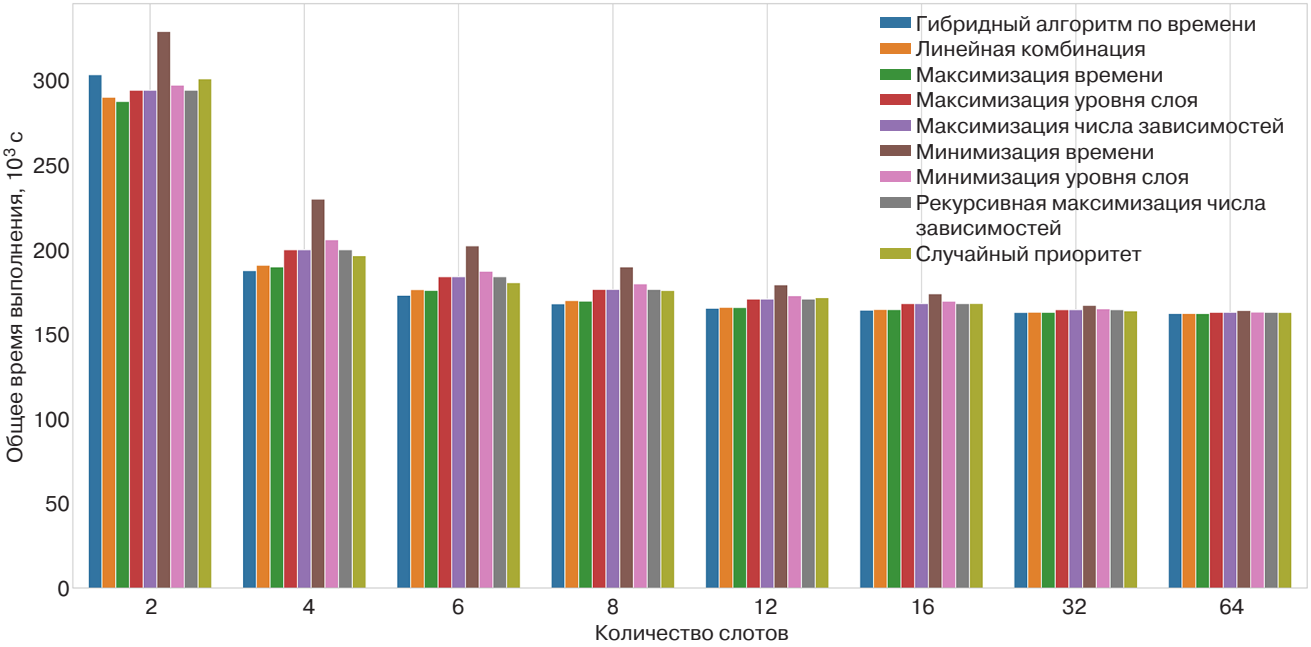

Рис. 1. Результаты оркестрации представленных алгоритмов (цвета указаны в порядке справа направо)

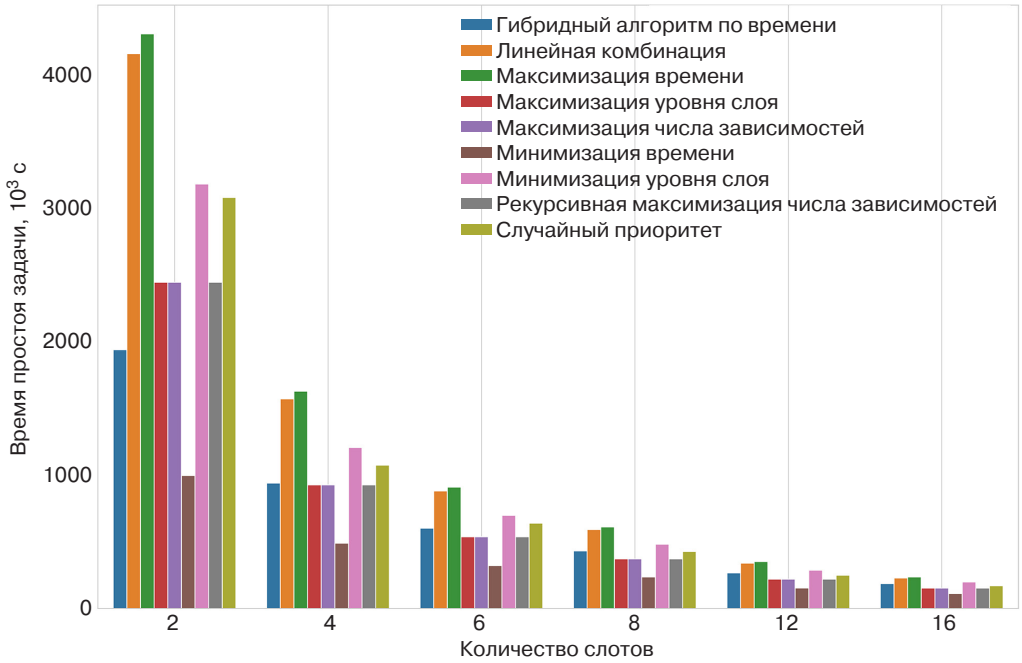

Рис. 2. Время простоя задач для различных алгоритмов

задачи, начинает обработку с самых длительных задач, вынуждая короткие операции накапливаться и простаивать в течение всего времени выполнения длительных процессов. Например, если в очереди присутствует одна 10-часовая задача и двадцать 1-минутных, при приоритизации длительных процессов все короткие задачи будут ожидать всю длительность 10-часовой операции, создавая колоссальный совокупный простой.

Таким образом, алгоритм, приоритизирующий длительные процессы, оптимизирует общее время завершения всей очереди за счет концентрации ресурсов на долгих операциях, а алгоритм, приоритизирующий минимальные по времени выполнения задачи, жертвует скоростью обработки длительных задач ради минимизации времени ожидания для большинства операций, гибридный алгоритм смягчает этот компромисс, достигая баланса между двумя метриками и при этом демонстрируя наилучший результат по длительности выполнения всего конвейера задач.

ЗАКЛЮЧЕНИЕ

В работе предложена комбинация методологического подхода и инструментальных решений для оптимизации ETL-конвейеров, адаптированных к динамическим условиям выполнения. В отличие от классических исследований, фокусирующих внимание на статической приоритизации или изолированном анализе алгоритмов, предлагаемая модель интегрирует имитацию выполнения задач с учетом ресурсных ограничений и топологических характеристик цепочки процессов, формирующих наполнение хранилища данных.

Исследование продемонстрировало, что приоритизация на основе топологии связей процессов возможна и позволяет уменьшить общее время выполнения конвейера данных. Таким образом, оптимальный выбор алгоритма приоритизации может значительно снизить общее время простоя, что повлияет на общую длительность процессов, как это продемонстрировал гибридный алгоритм, делящий очередь на очередь минимальных по времени выполнения задач и наиболее длительных.

Результаты исследования обладают значительной практической ценностью для оптимизации проектирования и эксплуатации ETL-процессов в современных хранилищах данных и озерах данных. Разработанный гибридный алгоритм приоритизации продемонстрировал в экспериментах снижение общей длительности выполнения конвейеров данных на 15–17% по сравнению с альтернативными алгоритмами. Ключевым преимуществом для его практического внедрения является архитектурная гибкость алгоритма: он может быть реализован в виде кастомного компонента для популярных систем оркестрации с открытым исходным кодом, например, Apache Airflow. Подобная интеграция позволяет повысить эффективность использования вычислительных ресурсов кластера при минимальных затратах на модификацию существующей инфраструктуры.

Перспективным направлением дальнейших исследований является разработка и оценка адаптивных алгоритмов приоритизации, способных динамически учитывать текущую загрузку вычислительных

ресурсов ЦХД в реальном времени. Учитывая, что стратегия, основанная на максимальной длительности задачи, показала свою эффективность в статических условиях, производительность ETL-конвейера в значительной степени зависит от фактической доступности таких ресурсов, как вычислительные ядра, память, пропускная способность дисковой подсистемы и сети, которые могут существенно флуктуировать в процессе выполнения. Интеграция систем мониторинга ресурсов и использование этих динамических метрик в качестве дополнительных входных параметров для алгоритма приоритизации открывает путь к созданию более гибких и отзывчивых систем. Такой подход потенциально позволит оптимизировать распределение задач не только на этапе планирования, но и в ходе исполнения, оперативно повышая приоритет задач, чье выполнение может быть ускорено при временном избытке определенного ресурса, или с помощью понижения ресурсоемких задач в периоды пиковой нагрузки, тем самым минимизируется общее время выполнения и снижая риск простоев из-за нехватки ресурсов. Исследование эффективности различных стратегий комбинирования прогнозируемой длительности задач с актуальными метриками ресурсов представляет значительный научный и практический интерес.

Вклад авторов

Д.А. Пушкарев – концептуализация исследования, разработка методологии, программное обеспечение, валидация результатов, подготовка первоначального варианта рукописи, визуализация.

В.Б. Богатырев – концептуализация исследования, разработка методологии, рецензирование и редактирование текста статьи, научное руководство.

Все авторы прочитали и одобрили опубликованную версию рукописи.

Authors' contributions

D.A. Pushkarev – conceptualization, methodology, software, validation, writing – original draft preparation, visualization.

V.B. Bogatyrev – conceptualization, methodology, writing – review and editing, supervision.

All authors have read and approved the published version of the manuscript.

СПИСОК ЛИТЕРАТУРЫ

1. Kimball R., Ross M. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling.* 3nd ed. Kimball group. Wiley; 2013, 608 p.
2. Simitsis A., Skiadopoulos S., Vassiliadis P. The History, Present, and Future of ETL Technology. In: *DOLAP, CEUR Workshop Proceedings*. 2023;3369:3–12.
3. Tian W. Enhancing Financial Decision-Making Through Automated Business Intelligence Systems. *Int. J. e-Collaboration* (*IJeC*). 2025;21(1):1–20. URL: https://www.igi-global.com/article/enhancing-financial-decision-making-through-automated-business-intelligence-systems/367575. Дата обращения 20.01.2025. / Accessed January 20, 2025.
4. El-Sappagh S.H.A., Hendawi A.M.A., El Bastawissy A.H. A proposed model for data warehouse ETL processes. *Journal of King Saud University – Computer and Information Sciences* (*J. King Saud Univ.*). 2011;23(2):91–104. https://doi.org/10.1016/j.jksuci.2011.05.005
5. Wijaya R., Pudjoatmodjo B. An overview and implementation of extraction-transformation-loading (ETL) process in data warehouse. In: *2015 3rd International Conference on Information and Communication Technology* (*ICoICT*). 2015. P. 70–74. https://doi.org/10.1109/ICoICT.2015.7231399
6. Кузьмина Ю.В., Кубанских О.В. Краткое описание процесса ETL. *Ученые записки Брянского государственного университета*. 2017;1(5):33–36. https://www.elibrary.ru/zmwlez
7. Dhaouadi A., Bousselmi K., Gammoudi M.M., Monnet S., Hammoudi S. Data Warehousing Process Modeling from Classical Approaches to New Trends: Main Features and Comparisons. *Data*. 2022;7(8):113. https://doi.org/10.3390/data7080113
8. Vassiliadis P., Simitsis A., Skiadopoulos S. Graph-Based Modeling of ETL Activities with Multi-level Transformations and Updates. In: Tjoa A.M., Trujillo J. (Eds.). *Data Warehousing and Knowledge Discovery. Part of the book series*: *DaWaK 2005*. *Lecture Notes in Computer Science*. 2005. V. 3589. P. 43–52. https://doi.org/10.1007/11546849_5
9. Yasmin J., Wang J.A., Tian Y., Adams B. An empirical study of developers' challenges in implementing Workflows as Code: A case study on Apache Airflow. *J. Syst. Software.* 2024;219(5):112248. https://doi.org/10.1016/j.jss.2024.112248
10. Михайлов А.Н. Использование Apache Airflow для оркестрации процессов обработки данных. *Вестник науки*. 2024;10(79):783–787. https://www.elibrary.ru/ijihms
11. Громов Н.Д., Платошин А.И., Панов А.В. Сравнительный анализ средств и платформ для автоматизации ETL процессов в современных хранилищах данных. *Международный журнал гуманитарных и естественных наук.* 2023;11-4(86):46–48. https://doi.org/10.24412/2500-1000-2023-11-4-46-48
12. Жданов Д.Е. Построение ETL процессов на базе Cron и оркестратора задач Luigi. *Актуальные исследования*. 2023;46-1(176):63–68. https://www.elibrary.ru/nenikj
13. Ueter N., Günzel M., Brüggen G., Chen J. Parallel Path Progression DAG Scheduling. *IEEE Transactions on Computers*. 2023;72(10):3002–3016. https://doi.org/10.1109/TC.2023.3280137

14. Богатырев В.А., Богатырев С.В., Богатырев А.В. Оценка готовности компьютерной системы к своевременному обслуживанию запросов при его совмещении с информационным восстановлением памяти после отказов. *Научно-технический вестник информационных технологий, механики и оптики.* 2023;23(3):608–617. https://doi.org/10.17586/2226-1494-2023-23-3-608-617
15. Bogatyrev V.A., Bogatyrev S.V., Bogatyrev A.V. Recovery of Real-Time Clusters with the Division of Computing Resources into the Execution of Functional Queries and the Restoration of Data Generated Since the Last Backup. In: Vishnevskiy V.M., Samouylov K.E., Kozyrev D.V. (Eds.). *Distributed Computer and Communication Networks: Control, Computation, Communications*. Book series: *Lecture Notes in Computer Science* (*including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics*). 2024. V. 14123. P. 236–250. https://doi.org/10.1007/978-3-031-50482-2_19
16. Karagiannis A., Vassiliadis P., Simitsis A. Scheduling strategies for efficient ETL execution. *Inform. Syst.* 2013;38(6): 927–945. https://doi.org/10.1016/j.is.2012.12.001
17. Topcuoglu H., Hariri S., Min-You Wu. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *IEEE Trans. Parallel Distrib. Syst*. 2007;13(3):260–274. https://doi.org/10.1109/71.993206
18. Strengholt P. *Building Medallion Architectures*. 1st ed. Sebastopol (CA): O'Reilly Media; 2024, 209 p.
19. Serra J. *Deciphering Data Architectures*. 1st ed. Sebastopol (CA): O'Reilly Media; 2024, 146 p.
20. Blažić G., Poščić P., Jakšić D. Data warehouse architecture classification. In: *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics* (*MIPRO*). 2017. P. 1491–1495. https://doi.org/10.23919/MIPRO.2017.7973657

REFERENCES

1. Kimball R., Ross M. *The Data Warehouse Toolkit: The Definitive Guide to Dimensional Modeling.* 3nd ed. Kimball group. Wiley; 2013, 608 p.
2. Simitsis A., Skiadopoulos S., Vassiliadis P. The History, Present, and Future of ETL Technology. In: *DOLAP, CEUR Workshop Proceedings*. 2023;3369:3–12.
3. Tian W. Enhancing Financial Decision-Making Through Automated Business Intelligence Systems. *Int. J. e-Collaboration* (*IJeC*). 2025;21(1):1–20. Available from URL: https://www.igi-global.com/article/enhancing-financial-decision-making-through-automated-business-intelligence-systems/367575. Accessed January 20, 2025.
4. El-Sappagh S.H.A., Hendawi A.M.A., El Bastawissy A.H. A proposed model for data warehouse ETL processes. *Journal of King Saud University – Computer and Information Sciences* (*J. King Saud Univ.*). 2011;23(2):91–104. https://doi.org/10.1016/j.jksuci.2011.05.005
5. Wijaya R., Pudjoatmodjo B. An overview and implementation of extraction-transformation-loading (ETL) process in data warehouse. In: *2015 3rd International Conference on Information and Communication Technology* (*ICoICT*). 2015. P. 70–74. https://doi.org/10.1109/ICoICT.2015.7231399
6. Kuzmina Yu.V., Kubanskikh O.V. Brief description of the ETL process. *Uchenye Zapiski Bryanskogo Gosudarstvennogo Universiteta = Scientific Notes of the Bryansk State University.* 2017;1(5):33–36 (in Russ.). https://www.elibrary.ru/zmwlez
7. Dhaouadi A., Bousselmi K., Gammoudi M.M., Monnet S., Hammoudi S. Data Warehousing Process Modeling from Classical Approaches to New Trends: Main Features and Comparisons. *Data*. 2022;7(8):113. https://doi.org/10.3390/data7080113
8. Vassiliadis P., Simitsis A., Skiadopoulos S. Graph-Based Modeling of ETL Activities with Multi-level Transformations and Updates. In: Tjoa A.M., Trujillo J. (Eds.). *Data Warehousing and Knowledge Discovery. Part of the book series*: *DaWaK 2005*. *Lecture Notes in Computer Science*. 2005. V. 3589. P. 43–52. https://doi.org/10.1007/11546849_5
9. Yasmin J., Wang J.A., Tian Y., Adams B. An empirical study of developers' challenges in implementing Workflows as Code: A case study on Apache Airflow. *J. Syst. Software.* 2024;219(5):112248. https://doi.org/10.1016/j.jss.2024.112248
10. Mikhailov A.N. Using Apache Airflow for Data Processing Orchestration. *Vestnik Nauki.* 2024;10(79):783–787 (in Russ.). https://www.elibrary.ru/ijihms
11. Gromov N.D., Platoshin A.I., Panov A.V. Comparative analysis of tools and platforms for automation of ETL processes in modern data warehouses. *Mezhdunarodnyi zhurnal gumanitarnykh i estestvennykh nauk = International Journal of Humanities and Natural Sciences.* 2023;11-4(86):46–48 (in Russ.). https://doi.org/10.24412/2500-1000-2023-11-4-46-48
12. Zhdanov D.E. Building ETL Processes Based on Cron and Luigi Task Orchestrator. *Aktual'nye issledovaniya = Current Research*. 2023;46-1(176):63–68 (in Russ.). https://www.elibrary.ru/nenikj
13. Ueter N., Günzel M., Brüggen G., Chen J. Parallel Path Progression DAG Scheduling. *IEEE Transactions on Computers*. 2023;72(10):3002–3016. https://doi.org/10.1109/TC.2023.3280137
14. Bogatyrev V.A., Bogatyrev S.V., Bogatyrev A.V. Assessment of the readiness of a computer system for timely servicing of requests when combined with informational recovery of memory after failures. *Nauchno-tekhnicheskii vestnik informatsionnykh tekhnologii, mekhaniki i optiki = Scientific and Technical Journal of Information Technologies, Mechanics and Optics.* 2023;23(3):608–617 (in Russ.). https://doi.org/10.17586/2226-1494-2023-23-3-608-617
15. Bogatyrev V.A., Bogatyrev S.V., Bogatyrev A.V. Recovery of Real-Time Clusters with the Division of Computing Resources into the Execution of Functional Queries and the Restoration of Data Generated Since the Last Backup. In: Vishnevskiy V.M., Samouylov K.E., Kozyrev D.V. (Eds.). *Distributed Computer and Communication Networks: Control, Computation, Communications*. Book series: *Lecture Notes in Computer Science* (*including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics*). 2024. V. 14123. P. 236–250. https://doi.org/10.1007/978-3-031-50482-2_19

16. Karagiannis A., Vassiliadis P., Simitsis A. Scheduling strategies for efficient ETL execution. *Inform. Syst.* 2013;38(6): 927–945. https://doi.org/10.1016/j.is.2012.12.001
17. Topcuoglu H., Hariri S., Min-You Wu. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *IEEE Trans. Parallel Distrib. Syst*. 2007;13(3):260–274. https://doi.org/10.1109/71.993206
18. Strengholt P. *Building Medallion Architectures*. 1st ed. Sebastopol (CA): O'Reilly Media; 2024, 209 p.
19. Serra J. *Deciphering Data Architectures*. 1st ed. Sebastopol (CA): O'Reilly Media; 2024, 146 p.
20. Blažić G., Poščić P., Jakšić D. Data warehouse architecture classification. In: *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics* (*MIPRO*). 2017. P. 1491–1495. https://doi.org/10.23919/MIPRO.2017.7973657

Об авторах

Пушкарев Даниил Александрович, аспирант, преподаватель, факультет программной инженерии и компьютерной техники, ФГАОУ ВО «Национальный исследовательский университет ИТМО» (Университет ИТМО) (197101, Россия, Санкт-Петербург, Кронверкский пр., д. 49, лит. А). E-mail: dpushkarev@itmo.ru. SPIN-код РИНЦ 5781-0210, https://orcid.org/0009-0003-2688-0093

Богатырев Владимир Анатольевич, д.т.н., профессор факультета, факультет программной инженерии и компьютерной техники, ФГАОУ ВО «Национальный исследовательский университет ИТМО» (Университет ИТМО) (197101, Россия, Санкт-Петербург, Кронверкский пр., д. 49, лит. А); профессор кафедры информационной безопасности, ФГАОУ ВО «Санкт-Петербургский государственный университет аэрокосмического приборостроения» (ГУАП) (190000, Россия, Санкт-Петербург, ул. Большая Морская, д. 67, лит. А). E-mail: vabogatyrev@itmo.ru. Scopus Author ID 7006571069, SPIN-код РИНЦ 3310-8044, https://orcid.org/0000-0003-0213-0223

About the Authors

Daniil A. Pushkarev, Postgraduate Student, Lecturer, Faculty of Software Engineering and Computer Systems, ITMO University (49, bldg. A, Kronverkskii pr., St. Petersburg, 197101 Russia). E-mail: dpushkarev@itmo.ru. RSCI SPIN-code 5781-0210, https://orcid.org/0009-0003-2688-0093

Vladimir A. Bogatyrev, Dr. Sci. (Eng.), Professor, Faculty of Software Engineering and Computer Systems, ITMO University (49, bldg. A, Kronverkskii pr., St. Petersburg, 197101 Russia); Professor, Department of Information Security, Saint Petersburg State University of Aerospace Instrumentation (SUAI) (67, bldg. A, Bolshaya Morskaya ul., St. Petersburg, 190000 Russia). E-mail: vabogatyrev@itmo.ru. Scopus Author ID 7006571069, RSCI SPIN-code 3310-8044, https://orcid.org/0000-0003-0213-0223