Information systems. Computer sciences. Issues of information security
Информационные системы. Информатика. Проблемы информационной безопасности

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

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.

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.

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.

INTRODUCTION

Every year sees a growth in the amount of data passing through various applications. One of the main tasks facing data storage developers is to collect and store this data Central data warehouses (CDWs) [1, 2] have been proposed as a solution to this problem. These are distributed storage facilities which collect data from all services for further analysis, and provide additional information which businesses can use to their advantage. In this way, CDWs are becoming the center for all business decisions [3].

The repository is populated through the processes of extracting (Extract), transforming (Transform), and loading (Load) data—ETL [2, 4–7]. These processes are complex operations aimed at integrating data from a variety of sources, including relational databases, file systems, and web servers. The first stage involves data extraction which means collecting information from the specified sources. Next, the data undergoes a transformation process, during which it is structured, normalized, and cleaned, in order to improve the quality and consistency of the information. The final stage is loading the prepared data into the target storage to make it available for subsequent analysis and use.

In modern data processing systems, ETL processes play a key role in preparing information. However, with the increase in infrastructure scales and the number of such processes, the problem of effectively managing their execution arises. When dozens or hundreds of ETL tasks interact within a complex architecture, manually determining the order in which they are launched becomes not only labor-intensive, but also risks suboptimal results. The sequence of process execution directly affects the overall data processing time. Competition for computing resources, cascading delays due to dependencies between tasks, and suboptimal load distribution can multiply the execution time of the entire process pipeline. For example, a delay in a process which provides data for several subsequent stages can cause a chain reaction of downtime. In order to resolve this problem, not only do prioritization methods need to be implemented, but also the architecture of the ETL system needs to be clearly modeled.

Complex interrelationships between processes, including explicit data dependencies, in which the output of one task serves as the input for another, and hidden constraints (memory resources, network bandwidth), can be visualized using directed acyclic graphs (DAGs) [8].

Directed acyclic graphs are accepted as the standard tool for modeling data extraction, transformation, and loading processes. This is due to their key properties which ensure reliability, efficiency, and transparency of data processing [2, 7, 8]. The acyclic nature of the structure guarantees the absence of infinite cycles, critical for ensuring the completion of tasks in data processing pipelines, especially when working with large amounts of information. The graph data structure allows the execution of interdependent tasks to be controlled, which is important when building data warehouse filling processes (for example, data extraction always precedes data transformation). In addition, thanks to its structure a directed acyclic graph allows parallel tasks to be worked on efficiently. This also allows the dependencies between different stages of data processing to be visualized. Thus, the tasks represented in the graph are independent and can be performed simultaneously.

Orchestration tools such as Apache Airflow [9–11] provide basic mechanisms for defining dependencies, but effective prioritization requires more detailed analysis which takes into account dynamically changing loads, the criticality of business metrics, and which predict bottlenecks. Without a systematic approach to describing the architecture and formalizing the rules for ordering processes, there is a risk that they will be executed in a sequence which is far from optimal. This is particularly critical in the context of near real-time analytics.

[2] Directed acyclic graph – ориентированный ациклический граф.

Despite the significant practical importance of optimizing ETL pipelines, the number of specialized studies on task prioritization in this area remains limited. Most existing orchestrators[3, 4] (Apache Airflow, NiFi, Luigi) [6, 10–12] use simplified strategies such as first-in-first-out or static priorities set by the user based solely on their empirical experience. Significant achievements exist in the related field of real-time systems, in which the DAG model is also widely used [13]. These studies solve the problem of guaranteed adherence to strict deadlines for parallel tasks on multiprocessor systems, using the concept of parallel progression of selected paths in a graph with sub-task prioritization and complex worst-case response time analysis.

Despite their common foundation in directed acyclic graphs, the goals and conditions of ETL optimization differ fundamentally from real-time planning. Firstly, ETL focuses on minimizing the average pipeline execution time and task downtime with flexible parallelism, while real-time systems require guarantees of deadline compliance with strict resource allocation. The probability of completing requests within a given time and the readiness coefficient for completing requests within a given time can be used as metrics for real-time systems [14, 15]. Secondly, the performance metrics differ. For ETL, the total duration and resource utilization are critical, rather than worst-case analysis.

The direct application of methods for improving the efficiency of real-time systems (including path selection algorithms) in an ETL context is suboptimal. Their excessive complexity and reservation requirements lead to underutilization of resources, ignoring at the same time the specifics of data storage layers and the dynamic constraints of orchestrators. Static approaches to real-time assurance are poorly adapted to load changes, in which hybrid strategies are more efficient. Thus, despite the formal similarity of the models, the difference in operational requirements and metrics necessitates the development of specialized prioritization methods for ETL, which justifies this study.

An analysis of existing ETL prioritization methods [16, 17] reveals significant limitations in current approaches:

- The proposed algorithms do not take into account the topological characteristics of process graphs which negatively affects the efficiency of planning in distributed systems;
- The predominant method of prioritization remains static. It is based on user labels for individual tasks and does not take into account the dynamics of execution and the mutual influence of processes in the pipeline.

The aim of this work is to develop and justify a method for prioritizing ETL processes which minimizes the execution time of the entire data pipeline using system analysis and simulation modeling methods.

This study proposes a simulation model which enables ETL process management in complex architecture environments to be analyzed. The model recreates an environment in which a directed acyclic graph is generated, reflecting the chains of ETL task execution which form a theoretical data warehouse. Each directed acyclic graph models dependencies between processes and time constraints. A key feature of the approach is the ability to compare different prioritization algorithms under identical conditions, providing an objective assessment of their effectiveness. This allows not only the execution of processes to be visualized within a centralized data warehouse, but also the impact of the selected algorithm on the overall duration of the pipeline to be quantitatively assessed. For example, by varying the number of nodes, the depth of dependencies, or the duration of a single task, it is possible to determine which algorithm demonstrates the best results in which scenarios. The results of the model generate metrics for comparison such as average execution time and percentage of resource downtime, forming the basis for an informed choice of optimization method in real-world conditions. Thus, the simulation model serves as a tool for systematic analysis of ETL orchestration, minimizing the risks of implementing ineffective solutions in industrial systems.

1. FORMALIZATION OF THE TASK

As part of this work, we will formalize the task of orchestrating ETL processes and will also establish a number of constraints necessary for its solution. The orchestration task includes several key stages: defining priorities for tasks; controlling the number of processes running simultaneously; and managing changes in task statuses according to their current status.

It should be emphasized that the study will not analyze the load on the target computing system. Instead, a variable will be introduced to regulate the load on the system, in order to determine the maximum number of simultaneously active tasks–slots. This is a common approach in similar tools. For example, Apache Airflow has a parameter for the number of available computing slots[5], responsible for the total

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

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

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

number of tasks which can be executed simultaneously across all processes. In addition, an additional restriction will be imposed. Once a specific process is started, its execution cannot be suspended until that process is completed. Resources will only become available once it has finished running.

Let us formulate the optimization problem for the orchestration process. Let **G** = (**V**, **E**) be a directed acyclic graph, in which **V** is a set of processes and **E** is a set of dependencies between them. In the case where the maximum number of simultaneously executing processes is limited to one, the problem can be solved using the classic method of topological sorting. Topological sorting is a linear ordering of the vertices of a directed acyclic graph, in which for each directed edge from vertex A to vertex B, the condition that A precedes B in this ordering is satisfied. Applying topological sorting to graph **G** allows us to obtain a sequence of process execution which satisfies all the established dependencies.

For this task, we can use the following notations: $\mathbf{P} = \{p_1, p_2, \ldots, p_n\}$ is the set of processes; $\mathbf{D}(p_i)$ set of processes upon which the process p_i depends; t_i is the execution time of each process.

The objective is to minimize the total execution time of all processes, taking their dependencies into account. This optimization problem can be written as follows:

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

provided that for each process p_i the following condition holds: p_j is executed before p_i if $p_j \in \mathbf{D}(p_i)$.

However, with such a formulation of the problem, in which only one process can be launched at a time, we obtain a single solution. This is the sum of the execution times of all processes:

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

If N processes can be run simultaneously (in parallel), this solution may not be unique. Given this formulation, the new optimization problem can be described as follows:

1. Let us define a set of active processes $\mathbf{A}(t)$ which can be launched at time t.
2. Let $C(t) \le N$ be the number of processes which can be launched at time t.
3. The execution time of all processes will now depend on the parallelism of their execution.

The general objective remains the same—minimizing the total execution time (1).

2. SIMULATION MODELING OF ORCHESTRATION

Comparing different prioritization algorithms upon real data in a computing cluster can be challenging for several reasons. Firstly, individual experiments can take several hours to complete. Secondly, running multiple parallel processes requires significant cluster computing power. Therefore, it makes sense to consider a simulation model which would reproduce the behavior of a similar system while minimizing time and computational resources.

The simulation of parallel execution of tasks with a limited number of slots was implemented using a discrete-event modeling approach with a custom event scheduler based on a priority queue, implemented in Python. Python was chosen for implementing the simulation model due to its combination of advantages, corresponding to the specifics of the research problem. This programming language provides a well-developed ecosystem of scientific libraries which significantly simplify model implementation. Furthermore, Python's integration with visualization tools and data processing capabilities simplifies model validation and interpretation of experimental metrics.

The operation algorithm includes four key stages. First, a list of running tasks is generated. At this stage, the program analyzes the directed acyclic graph and adds processes to the queue, the dependencies of which are already fulfilled (for example, parent nodes in the graph are completed or missing). The current priority of these processes is the highest among available tasks. This is implemented by checking the status of nodes in a directed acyclic graph and sorting them according to specified prioritization rules (for example, based on the layer value at which the process is running). Second, the minimum execution time among all active tasks in the list is determined: this is the time it takes for at least one of the tasks to complete. Third, this value is added to the total pipeline execution time, thus moving the system forward in time by this amount. Finally, for all running tasks, their remaining execution time is reduced by the calculated minimum value. Completed processes are marked as completed, updating the dependency graph for subsequent iterations. This approach effectively emulates the concurrent execution of tasks, taking into account both the logical dependencies between them and the dynamics of resource allocation, providing flexibility for testing various prioritization algorithms within a single model.

Let us consider the data upon which the experiments will be conducted. A classic data warehouse has a layered structure [1, 7, 18, 19].

Operational data storage. Stores operational, often current, detailed data which has not yet been aggregated. This layer serves as a source of operational information for reporting and can be directly used to build higher-level layers.

Detailed data layer. A centralized repository of integrated data collected from one or more disparate sources. The data is structured specifically for querying and analysis.

Data marts. A subset of a data storage focused on a specific area of activity or team. Data maps are developed with specific needs or analysis in mind.

There are situations in which the number of layers may exceed three. This parameter is determined by the specific application conditions. However, the most common implementation option is three-layer architecture: the architecture proposed by Ralph Kimball [1] and the medallion architecture [13, 20].

In this study, directed acyclic graphs were generated to ensure maximum coverage of diverse scenarios. The generation was performed using the following methodology:

- Number of layers is 3. This number is most common when forming a data storage.
- The total number of nodes (processes) for each graph was generated according to a uniform distribution in the range from 250 to 400.
- The nodes were distributed among the layers in accordance with Table 1.
- Additionally, links were added within the layers of the detail layer and data marts to model complex transformations. Each node had up to two random predecessors within its layer (the probability of establishing a link was 0.1, the Pareto distribution for the number of links). After generating the links, a check for cycles was performed in order to exclude links that could cause a given cycle. The total number of links per node does not exceed 10.
- The execution time of each node was assigned independently, and uniformly distributed in the range from 100 to 5000 s. No relationship was established between execution time, layer, or node degree.

Table 1. Percentage of tables relative to the layer

Layer	Share of tables
Source data storage	30–40%
Integrated repository of a structured data ready for analysis	30–40%
Specialized subset focused on a specific subject area	20–30%

In order to evaluate the performance of nine prioritization algorithms, each algorithm was run on each generated directed acyclic graph. Experiments were conducted for varying numbers of parallelism slots to assess the impact of increasing this parameter on the performance of the algorithm.

3. PRIORITIZATION ALGORITHMS

Task prioritization is a process in which the priority level of each individual task is determined based upon its characteristics and relationships with other tasks. This priority level can be random or determined based on the parameters of a specific task, its position, and its relationships with other elements. Let us consider several possible prioritization algorithms.

Random selection. This algorithm operates on the principle of equal priority. This means that any process that can be launched at a given moment will be activated with equal probability. This solution is not optimal and is more of a baseline solution used for comparison with alternative approaches.

Minimum execution time. Processes with the shortest execution time are executed first.

Maximum execution time. Processes with the longest execution time are executed first.

Maximizing the number of dependencies. The algorithm is based on the number of processes dependent on each process.

Hybrid time algorithm. This approach divides the available computing slots of the ETL system into two parts. The first part is allocated to execute the tasks with the highest priority according to the maximum execution time strategy. The second part is allocated to execute the tasks with the highest priority according to the minimum execution time strategy.

Recursive maximization of the number of dependencies. An algorithm which takes into account the number of dependent tasks not only of the current process, but also of all downstream processes that depend on this one.

Minimum layer level. An algorithm based on the layer level within which a given ETL process is executed. The operational layer corresponds to level 0, the detailed layer to level 1, and the data mart layer to level 2. If there are more layers, they are classified similarly according to the graph topology.

Maximum layer level. An algorithm which prioritizes processes at a higher layer level.

Linear combination. This approach integrates all parameters into a single model. For prioritization, a linear combination of features and their corresponding coefficients is constructed. The following criteria are selected as parameters: execution time, number

of dependent processes, and layer level. The resulting linear combination has the following form:

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

wherein x_1, x_2, x_3 are constants, l_i is the layer level of the ith process.

When using this combination, the appropriate coefficients need to be selected. For this purpose, optimization algorithms were used, in which each iteration runs the simulation model with a new parameter vector **X**, resulting in the overall execution time of the model. This time must be minimized.

4. EXPERIMENTAL RESULTS

Based on the requirements described above, 50 graphs were generated representing the different architectures of ETL processes which fill the data storage.

Let us consider the process of selecting coefficients for an algorithm based on a linear combination. In order to determine the optimal values for these coefficients, N graphs were selected, which were run with predetermined coefficients x_i. The objective of the optimization model is to minimize the total execution time of all N graphs. Thus, the coefficients are selected not for a specific model, but rather generalized for various graph topologies. This approach is necessary because in real-world applications of the algorithm it may be impossible to select such parameters.

The following methods were selected as optimization algorithms for solving this problem: Powell's algorithm; the COBYLA method (from Constrained Optimization BY Linear Approximations); the Nelder–Mead algorithm; and the Conjugate Gradient (CG) method. The implementation of these algorithms is taken from the SciPy library, optimize module[6]. The results of the optimization algorithms are presented in Table 2.

Table 2. Results of solving the optimization problem for a linear combination

Model	Iterations	Execution time	Coefficients
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**

The best result was obtained using the Nelder–Mead algorithm. Based on the coefficients obtained, it can be concluded that this algorithm identified execution time as the most influential parameter for process prioritization.

The algorithm was applied to a set of graphs not included in the training set. The result was the total processing time of all processes in these graphs for various values of the number of available slots parameter (Fig. 1).

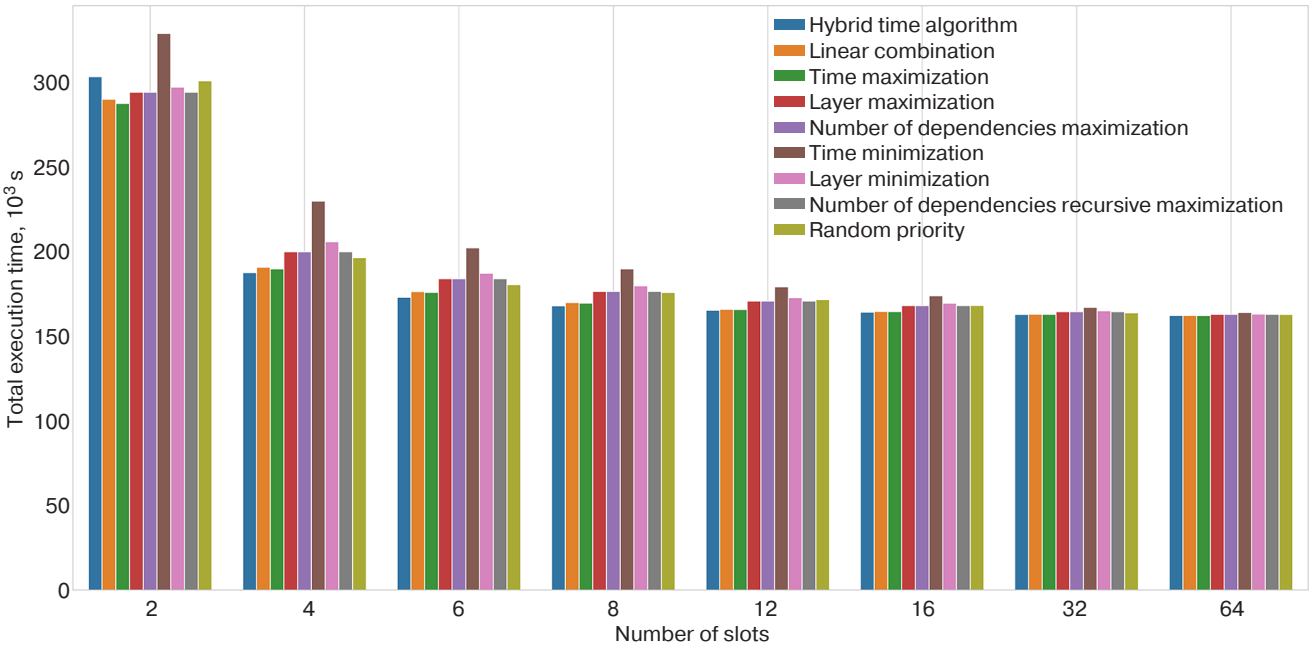

Fig. 1. Orchestration results of the presented algorithms (colors are indicated in order from left to right)

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

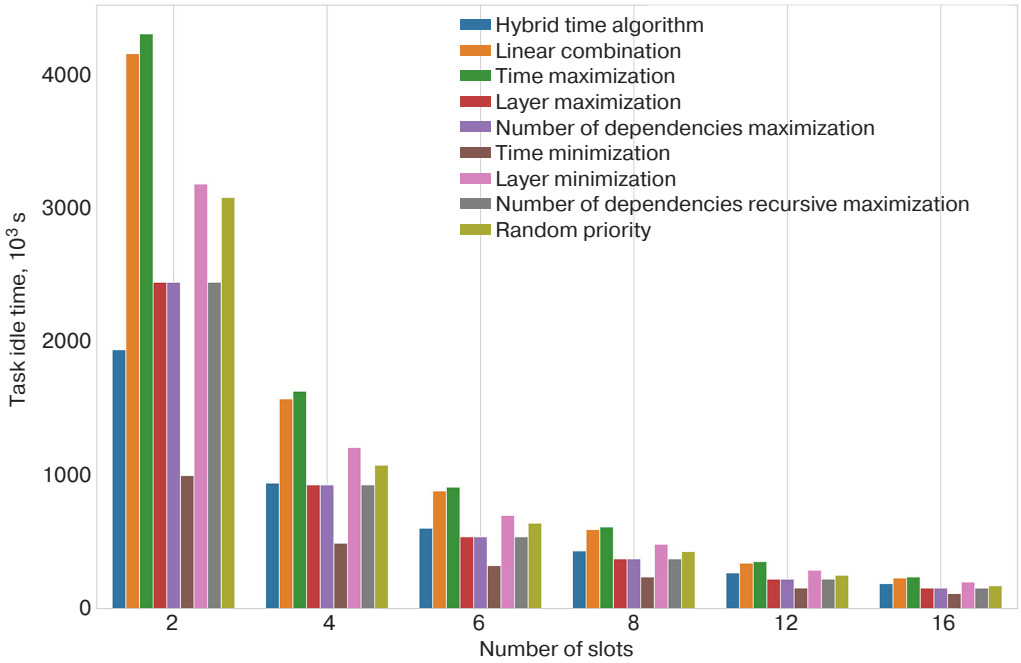

Fig. 2. Tasks idle time for different algorithms

To assess the efficiency of task servicing in the queue, we introduce an additional metric—task idle time. It is calculated as the total time spent in the queue by tasks ready to run. The results of this metric are presented in Fig. 2.

The results of the study demonstrate that the algorithm for prioritizing tasks with maximum execution time provides the best overall execution time for ETL processes only under conditions of limited parallelism. This is when the number of concurrently executing tasks (slots) is no more than two. Under such conditions, this algorithm minimizes the impact of long-running tasks as bottlenecks, preventing their blocking effect on the completion of the entire queue.

However, as the degree of parallelism increases, the hybrid algorithm, which divides the queue into two equal parts—one for tasks with minimum and one for maximum execution time—outperforms the algorithm which maximizes execution time based on total execution time. This is due to the ability of the hybrid approach to efficiently allocate resources. Dedicated slots for short tasks enable the queue to be quickly cleared of small operations, while slots for long-running tasks ensure their continuous execution without downtime. Furthermore, the hybrid algorithm significantly reduces the idle time of tasks in the queue, when compared to the time-maximizing algorithm. However, it is inferior in this respect to the algorithm that prioritizes tasks with minimum execution time.

The discrepancy regarding idle time, where minimizing time yields the lowest values and maximizing time yields the highest, can be explained by the fundamental difference in their optimization goals. An algorithm which prioritizes tasks with the shortest execution times ensures their rapid completion. This dramatically reduces the waiting time for most tasks in the queue. An algorithm which prioritizes tasks with the longest execution times starts processing the longest tasks, forcing short operations to accumulate and remain idle for the entire duration of long processes. For example, if a queue contains one 10-h task and twenty 1-min tasks, prioritizing long processes will cause all short tasks to wait for the entire duration of the 10-h operation, creating colossal cumulative idle time.

Thus, an algorithm which prioritizes long-running processes optimizes the overall queue completion time by concentrating resources on long-running operations, while an algorithm which prioritizes tasks with minimal execution time sacrifices the processing speed of long-running tasks in order to minimize the waiting time for most operations. A hybrid algorithm mitigates this tradeoff, achieving a balance between the two metrics while demonstrating the best result in terms of the duration of the entire pipeline of tasks.

CONCLUSIONS

This article proposes a combination of a methodological approach and tool solutions for optimizing ETL pipelines adapted to dynamic execution conditions. Unlike classical studies which focus on static prioritization or isolated algorithm analysis, the proposed model integrates task execution simulation, while taking into account resource constraints and the topological

characteristics of the process chain which generates the data warehouse's content.

The study demonstrated that prioritization based on the topology of process links is feasible and can reduce the overall execution time of a data pipeline. Thus, the optimal choice of prioritization algorithm can significantly reduce overall idle time. This will impact the overall duration of processes, as demonstrated by a hybrid algorithm that divides the queue into a queue of tasks with the shortest execution time and those with the longest execution time.

The results of the study have significant practical value for optimizing the design and operation of ETL processes in modern data warehouses and data lakes. The developed hybrid prioritization algorithm demonstrated experimental results which reduced the overall execution time of data pipelines by 15–17% compared to alternative algorithms. A key advantage for its practical implementation is the architectural flexibility of the algorithm. It can be implemented as a custom component for popular open-source orchestration systems, such as Apache Airflow. This integration enables increased efficiency in the utilization of cluster computing resources with minimal modifications to the existing infrastructure.

A promising area for further research is the development and evaluation of adaptive prioritization algorithms capable of dynamically accounting for the current load of centralized data storage computing resources in real time. While a strategy based on maximum task duration has proven effective under static conditions, ETL pipeline performance largely depends on the actual availability of resources such as computer cores, memory, disk subsystem bandwidth, and network capacity, which can fluctuate significantly during execution. Integrating resource monitoring systems and using these dynamic metrics as additional input parameters for the prioritization algorithm will pave the way for more flexible and responsive systems. This approach will potentially enable the optimization of task allocation not only at the planning stage but also during execution. This will also promptly increase the priority of tasks, the execution of which can be accelerated by a temporary surplus of a particular resource, or by downgrading resource-intensive tasks during peak load periods. This will thereby minimize overall execution time and reduce the risk of downtime due to resource shortages. The study of the effectiveness of various strategies for combining predicted task durations with actual resource metrics is of significant scientific and practical interest.

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.

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

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

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

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

Translated from Russian into English by Lyudmila O. Bychkova
Edited for English language and spelling by Dr. David Mossop