

СИСТЕМНЫЙ АНАЛИЗ, УПРАВЛЕНИЕ И ОБРАБОТКА ИНФОРМАЦИИ/SYSTEM ANALYSIS, MANAGEMENT AND PROCESSING OF INFORMATION

DOI: https://doi.org/10.60797/IRJ.2026.168.52 EDN: RDNUVT

КЛАССИФИКАЦИЯ И СРАВНИТЕЛЬНЫЙ АНАЛИЗ ПОДХОДОВ К СБОРУ КОЛОНОЧНОГО УРОВНЯ DATA LINEAGE НА БАЗЕ APACHE SPARK

Научная статья

Волков Д.Д.[1,] *, Разумовский Д.А.[2], Стучилин В.В.[3]
[1] ORCID : 0009-0005-9347-6888;
[2] ORCID : 0009-0006-1802-6956;
[3] ORCID : 0000-0001-8259-3206;
[1, 2, 3] Национальный исследовательский технологический университет МИСиС, Москва, Российская Федерация

* Корреспондирующий автор (volkov.deenis[at]gmail.com)

Предложена: 20.11.2025; Принята: 16.01.2026; Опубликована: 17.06.2026

Аннотация

Исследование посвящено анализу подходов и инструментов сбора происхождения данных на уровне отдельного столбца (column-level data lineage) в корпоративных конвейерах обработки больших данных на базе Apache Spark. Рассматривается проблема обеспечения прозрачности преобразований данных для задач аудита, воспроизводимости аналитики, расследования инцидентов качества и соблюдения регуляторных требований в условиях растущих объёмов и сложности конвейеров, а также ограничений импортонезависимости и требований к открытости исходного кода. Цель исследования: критически оценить существующие архитектурные подходы и инструменты столбцового lineage для Apache Spark, разработать их классификацию и формализованную систему критериев, позволяющую обоснованно выбирать решения для корпоративных платформ обработки данных. Тип исследования: обзорно-аналитическое с элементами классификационного и сравнительного анализа. Объект исследования: подходы и программные средства, обеспечивающие сбор столбцового lineage в конвейерах обработки данных на базе Apache Spark. Использованы методы системного анализа архитектуры типового корпоративного конвейера, анализа документации и открытого исходного кода, типологизации решений по способу интеграции, а также сравнительного анализа по формализованной совокупности функциональных и технических критериев. Проведена идентификация основных классов решений, разработана система критериев с дискретными шкалами значений, выполнено сопоставление отобранных инструментов, позволившее выделить их ключевые преимущества и ограничения и очертить профиль перспективной гибридной архитектуры, минимизирующей инфраструктурную сложность при сохранении полноты столбцового lineage. Основные выводы касаются особенностей применения различных подходов к lineage в корпоративных Spark-конвейерах, их влияния на эксплуатационные характеристики платформы и возможностей использования предложенных критериев при проектировании импортонезависимых решений управления данными.

Ключевые слова: data lineage, Apache Spark, column lineage, big data, большие данные, data governance, качество данных.

CLASSIFICATION AND COMPARATIVE ANALYSIS OF APPROACHES TO COLLECTING COLUMN-LEVEL DATA LINEAGE BASED ON APACHE SPARK

Research article

Volkov D.D.[1,] *, Razumovskii D.A.[2], Stuchilin V.V.[3]
[1] ORCID : 0009-0005-9347-6888;
[2] ORCID : 0009-0006-1802-6956;
[3] ORCID : 0000-0001-8259-3206;
[1, 2, 3] National University of Science and Technology, Moscow, Russian Federation

* Corresponding author (volkov.deenis[at]gmail.com)

Suggested: 20.11.2025; Accepted: 16.01.2026; Published: 17.06.2026

Abstract

The study analyses approaches and tools for tracking data lineage at the column level in enterprise big data processing pipelines based on Apache Spark. The challenge of ensuring transparency of data transformations for auditing, reproducibility of analytics, investigation of quality incidents, and compliance with regulatory requirements is examined, given the growing volume and complexity of pipelines, as well as constraints on vendor lock-in and open-source requirements. Research objective: to critically evaluate existing architectural approaches and columnar lineage tools for Apache Spark, develop a classification system and a formalised set of criteria to enable informed selection of solutions for enterprise data processing platforms. Type of research: review and analysis with elements of classification and comparative analysis. Research object: approaches and software tools that enable the collection of columnar lineage in data processing pipelines based on Apache Spark. Methods used include systematic analysis of the architecture of a typical enterprise pipeline, analysis of documentation and open-source code, typology of solutions by integration method, and comparative analysis based on a formalised set of functional and technical criteria. The main classes of solutions have been identified, a system of criteria with discrete value

scales has been developed, and a comparison of the selected tools has been carried out, which has made it possible to highlight their key advantages and limitations and to outline the profile of a promising hybrid architecture that minimises infrastructure complexity while maintaining the completeness of columnar lineage. The main conclusions concern the specific traits of applying various lineage approaches in enterprise Spark pipelines, their impact on the platform's performance, and the potential for using the proposed criteria when designing import-independent data management solutions.

Keywords: data lineage, Apache Spark, column lineage, big data, data governance, data quality.

Введение

За последние 15 лет наблюдается устойчивый мировой тренд перехода бизнеса от интуитивных решений к data-driven management. Исследования подтверждают, что Big Data и аналитика стали центральными инструментами стратегического и операционного управления в большинстве отраслей [1]. На основе обрабатываемых данных работают модели машинного обучения, строятся аналитические витрины, визуализируется отчетность, однако все большая интеграция больших данных в бизнес-процессы ведет к росту требований к комплексности и производительности конвейеров по обработке данных [2].

Цепочки преобразований данных, формирующие удобные пользовательские витрины и аналитические панели, требуют тщательного отслеживания и документирования. Отсутствие четкого понимания природы происхождения данных значительно повышает риски деградации пользовательского опыта, качества принятия решений и иных бизнес-процессов [3]. В этих условиях возрастает потребность в формализованном описании происхождения данных (data lineage), позволяющем проследить путь отдельных элементов данных от источников до конечных потребителей. Традиционным вариантом решения указанной задачи является формирование графа связанности уровня данных и таблиц: собираются и комбинируются зависимости между таблицами, отчетами и любыми другими наборами данных. Базовый подход обладает значительным количеством преимуществ: простота сбора, сравнительно небольшие требования к хранилищу и простые алгоритмы комбинации связей, однако он не учитывает комплексные зависимости данных на уровне столбцов [4]. Исследователи выделяют риск неправильного принятия решения при оценке воздействия изменения структуры родительских данных, так как с использованием базового (табличного) представления невозможно понять влияние одного конкретно взятого атрибута в большом объеме данных [5]. Таким образом, необходимость в более глубоком понимании связей между данными приводит к необходимости использования столбцового уровня представления (column lineage). Именно такой уровень позволяет выявлять скрытые зависимости между признаками, оценивать воздействие изменений структуры данных и корректно интерпретировать аналитические показатели.

Проблему сбора связанности данных невозможно рассматривать в отрыве от самого популярного инструмента обработки больших объемов данных — Apache Spark [6]. Фреймворк предоставляет простые и надежные интерфейсы по построению big data процессов. Низкий порог входа, наличие привычного SQL-диалекта и гибкость в обработке данных делают Apache Spark предпочтительным выбором для многих организаций. Функционал фреймворка ограничивается только построением процессов, но не обеспечивает сбора информации о происхождении данных. Поэтому в корпоративной практике используются внешние решения, предлагающие различные способы извлечения и представления column-level data lineage для Spark-конвейеров. Эти решения отличаются архитектурой, уровнем детализации, накладными расходами и степенью интеграции в экосистему работы с данными. Неоднозначность в выборе подходящего инструмента, низкая степень проработанности в научной литературе и даже отсутствие формальных критериев сравнения инструментов создают сложности при повышении уровня зрелости платформы по обработке данных путем внедрения указанных инструментов. В сложившейся ситуации возникает необходимость систематизировать существующие подходы и инструменты, оценить их применимость к корпоративным конвейерам на базе Apache Spark и определить направления дальнейшего развития в области столбцового data lineage [7].

В рамках исследования внимание уделяется подходам и инструментам, обеспечивающим сбор столбцового происхождения данных (column-level data lineage) в корпоративных конвейерах обработки данных на базе Apache Spark, включая как встроенные механизмы платформы, так и внешние решения уровня каталогов данных и систем управления метаданными.

Цель работы заключается в проведении обзора и классификации этих подходов, а также в создании системы требований и критериев их оценки с точки зрения практического применения в корпоративной среде.

Данная статья предлагает обобщённую классификацию решений уровня столбцов для Spark-конвейеров и формализованный набор критериев их сопоставления. Авторский вклад состоит в формулировке требований к столбцовому data lineage в корпоративных конвейерах на базе Apache Spark и в построении аналитической рамки, позволяющей выявить пробелы существующих решений и задать ориентиры для разработки новых методов и инструментов в данной области.

Методы и принципы исследования

2.1. Архитектурный контекст

Для полноценного понимания требований к системам сбора связей данных необходимо изучить актуальные принципы формирования архитектур конвейеров обработки данных. В данном подразделе формализуются архитектурные предпосылки, лежащие в основе дальнейшей классификации подходов и критериев. Такие процессы, как правило, строятся по принципу Extract-Transform-Load (ETL) или же Extract-Load-Transform (ELT). Источниками данных могут являться транзакционные системы, журналы логов и событий, шины данных, а также специализированные доменные системы (CRM, биллинг, антифрод и т.д.) [8]. Данные могут поступать как с определенной регулярностью (пакетная обработка), так и в непрерывном режиме (стриминг или near real time обработка).

В пакетной обработке Apache Spark используется как SQL движок, позволяющий читать данные из типовых форматов Parquet, ORC, CSV, JSON, Avro, в непрерывной обработке источниками служат шины данных, такие как RabbitMQ и Kafka, или специализированные внутренние инструменты [6]. Суть преобразований данных, как правило, не зависит от режима работы движка, оба сценария подразумевают фильтрации, расчеты агрегатов, простые преобразования. Последним базовым шагом является дедупликация и запись в оперативные витрины, аналитические хранилища или выгрузка данных во внешние системы. Уже на этом уровне формируется часть информации о происхождении данных: какие физические источники (конкретные базы, схемы, топики, каталоги в файловой системе) соответствуют логическим наборам данных, используемым в Spark-задачах. Для столбцового lineage важно, чтобы на этапе приёма данных были явно определены схемы входных наборов (имена и типы столбцов), так как дальнейшие преобразования будут оперировать именно этими атрибутами [4].

Типичная архитектура корпоративного конвейера на базе Apache Spark включает один или несколько уровней хранилищ и витрин данных. Наиболее распространённым является подход, близкий к архитектурам datalake или lakehouse: данные последовательно проходят слои сырых данных (raw), очищенных и нормализованных данных и аналитических представлений [9]. Описанные типовые конвейеры в значительной степени опираются на слой метаданных (Apache Hive, Apache HBase). Большинство хранилищ метаданных отвечают за три уровня информации: технические, операционные и бизнес-метаданные [10] см. таблицу 1.

Таблица 1 - Структура хранилищ метаданных

DOI: https://doi.org/10.60797/IRJ.2026.168.52.1

Слой метаданных	Описание	Примеры информации
Технический	Информация использующаяся big data движками для оптимизации внутренних вычислений	Соотношение имени схемы-таблицы к физическому месту хранению данных. Структура, партиционирование и список доступных партиций. Предварительно созданные конфигурации чтения данных для движков.
Операционный	Логи и журналы обращения к данным	Журналы выполненных задач. Параметры запусков. Метрики чтения и записи.
Бизнес	Воспринимаемые пользователями сведения о данных	Смысловые описания атрибутов, бизнес-правила, соответствие полей отчётам и показателям.

Столбцовое происхождение данных определяется на пересечении этих слоёв, оно описывает, как конкретные столбцы целевых таблиц и представлений связаны со столбцами источников и промежуточных наборов данных, и через какие операции Apache Spark осуществляется эта трансформация. В архитектуре конвейера можно выделить несколько типичных точек, где формируется или может быть извлечён столбцовый lineage:

- На границе чтения данных: соответствие физической схемы источника и логической схемы, используемой в Spark;
- Внутри Spark-приложений: логические планы запросов (LogicalPlan), планы DataFrame-преобразований и DAG-графы задач;
- На границе записи данных: соответствие результата расчета и целевой структуры таблиц/витрин.

Большая часть инструментов сбора lineage для Apache Spark, опирается именно на эти точки. Основные подходы: анализ логических и физических планов Spark, интеграция с каталогами данных и системами управления метаданными [11]. В контексте дальнейших разделов статьи важно, что столбцовый уровень lineage не является отдельным уровнем архитектуры, а встраивается во всю цепочку конвейера, опираясь на уже существующие технические и бизнес-метаданные и расширяя их формализованной информацией о зависимостях между столбцами.

2.2. Методика отбора и анализа инструментов lineage

На этапе подготовки сравнительного анализа был сформирован перечень кандидатов, включающий все доступные инструменты, связанные со сбором data lineage, косвенно упоминающие Apache Spark. Также в перечень вошли типовые архитектурные паттерны. Таким образом, было отобрано 45 решений и подходов. Далее к этому перечню применялись отборочные критерии, позволяющие оставить исключительно релевантных кандидатов. Из выборки были исключены инструменты, лишь косвенно упоминающие Spark и ориентированные преимущественно на другие СУБД или ETL-движки. В качестве следующего критерия выступала самостоятельность и завершённость реализации. В итоговую выборку не попали методические паттерны, а также вспомогательные системы, не применимые к обозреваемой теме. Кроме того, из выборки были исключены решения, для которых доступный объём публичной информации оказался недостаточным для оценки по всем финальным критериям раздела 3.2 (табл. 2). В результате предварительного отбора из расширенного множества были отобраны двенадцать решений, представляющих различные классы подходов и типов интеграции: Spline; OpenLineage (Spark integration); Spark Atlas Connector;

DataHub; OpenMetadata; Databricks Unity Catalog; AWS Glue + Amazon Neptune + Spline; Google Cloud Dataplex (с Dataproc / Serverless for Apache Spark); Collibra (в связке с OpenLineage); Monte Carlo Data; Atlan.

Основные результаты

3.1. Подробная классификация подходов к сбору столбцового lineage

Большинство зрелых решений отслеживания происхождения данных на уровне столбцов в Apache Spark основываются на анализе логического плана запросов и дерева выражений Catalyst. Логический план позволяет фиксировать операции, определяющие, как итоговые данные зависят от исходных. Основные различия между подходами заключаются в способе интеграции в архитектуру процесса обработки данных.

В plan-based подходе точка интеграции находится внутри Spark-приложения. Инструмент напрямую работает с LogicalPlan и, при необходимости, с физическим планом и деревом выражений. Через API Spark извлекается логический план запроса, анализируется дерево операторов, для каждого оператора анализируются выражения, и на этой основе восстанавливаются зависимости между входными и выходными атрибутами. В результате формируется граф lineage, где вершинами выступают столбцы таблиц и представлений, а рёбрами — операции трансформации. Показательный пример - Spline [12]. Инструмент встраивается в приложение, перехватывает планы выполнения, анализирует их и передаёт структуру lineage на серверную часть, где она сохраняется и визуализируется.

Преимущество этого подхода заключается в максимально полной и точной картине зависимостей в рамках SQL/DataFrame-логики, за счет опоры на максимально близкий к вычислениям логический план. Однако данный подход ограничен в обработке пользовательских функций-расширений Apache Spark, где выражения остаются не формализованными в рамках плана. Еще одним существенным недостатком является невозможность определения конкретного запуска приложения, например его идентификатора, без дополнительной логики.

Следующий подход основан на потоке событий о выполнении задач. Инструмент встраивается на уровень SparkListener или аналогичных механизмов и реагирует на события запуска приложения, исполнения конкретной задачи и окончания приложения [11]. В процессе инструмент получает сведения о том, какие наборы данных были прочитаны и записаны, какие SQL-запросы выполнялись и с какими параметрами и формирует lineage-записи для конкретных запусков. Открытый стандарт OpenLineage [13] задаёт модель таких событий и формат JSON-сообщений, которые могут публиковаться в центральный сервис.

В metadata-based подходе собираемые связи встраиваются в более широкий контекст метаданных и представляются через каталоги данных и метаданных [14]. Здесь ключевым артефактом становится запись в каталоге: объект таблицы, представления или отчёта, к которому привязан граф зависимостей. Классический пример такого подхода — Databricks Unity Catalog [15]. Платформа автоматически фиксирует runtime-lineage для SQL-запросов, ноутбуков, процессов и дашбордов, выполняемых в управляемой среде, и сохраняет его в виде связей между объектами каталога.

Преимущество metadata-based подхода заключается в тесной интеграции lineage с остальными метаданными: схемами, политиками доступа, бизнес-терминами, владельцами. Недостатки типичны: привязка к конкретной платформе и её API, неполный охват сценариев (например, операций вне SQL-движка или внешних кластеров), а также задержки между выполнением операций и появлением актуальных данных в каталоге.

В корпоративных платформенных решениях редко используется один из описанных подходов в изоляции. Преимущественно встречается гибрид, объединяющий несколько источников информации. Распространена комбинация анализа планов и событий, в указанной схеме логический план используется для вычисления зависимостей столбцов, а события исполнения служат вспомогательной информацией для привязки к конкретным запускам и обеспечивают удобную интеграцию с внешними сервисами [11]. Гибридные подходы позволяют одновременно достигать высокой точности на уровне столбцов, сохранять связь с конкретными запусками и встраивать lineage в процессы корпоративного управления данными с минимизацией рисков [16]. В следующих разделах эти классы решений будут рассмотрены с точки зрения предложенных критериев оценки и сравнительного анализа.

3.2. Система критериев оценки решений

Критерии для проведения сравнительного анализа разделены на два блока: функциональные и технологические. Первый блок критериев связан с функциональными возможностями решений и отражает, насколько глубоко и полно они покрывают задачи столбцового lineage. Ключевым является критерий уровня детализации и покрытия. Решения отличаются наличием или отсутствием столбцового уровня представления, а также поддержкой нетривиальных операций со столбцами, например оконных функций, пользовательских и внешних функций. Отдельно оценивается поддержка различных типов конвейеров. Решения могут быть разработаны специфично для классических пакетных загрузок и не учитывать потоковые сценарии. Потенциально существенным функциональным аспектом является наличие и качество средств визуализации: от минималистичных графов зависимостей до интерактивных интерфейсов, позволяющих ограничивать отображение lineage по объектам, столбцам, времени выполнения и другим параметрам. Удобство имеющегося интерфейса для задач аналитики, инженерии данных и аудита позволяет ускорить бизнес-процессы и снизить риск человеческой ошибки [17].

Второй блок критериев отражает технические свойства решений и их влияние на эксплуатацию инфраструктуры данных. Прежде всего рассматриваются накладные расходы на сбор lineage. Процессы по сбору связей создают дополнительные затраты времени на выполнение Spark-задач, с ними может быть связан рост потребления памяти и объёма записываемых метаданных. Для крупных конвейеров с большим числом процессов и комплексными комбинациями источников этот фактор может быть критичным [18].

Критерий масштабируемости призван оценить способность решения обслуживать растущее число источников, конвейеров и пользователей без существенного ухудшения производительности. Оценивается возможность

распределить нагрузку по нескольким узлам, поддержка отказоустойчивости, предусмотрены ли механизмы резервирования самого хранилища lineage. Инфраструктурные требования также формализуются в виде критериев. Ряд решений требует развёртывания отдельных сервисов и специализированных хранилищ (например, графовых СУБД), другие ограничиваются использованием существующих компонентов платформы. Наконец, основополагающим критерием при принятии решения об использовании инструмента является открытость исходного кода. Современные реалии в крупных российских компаниях диктуют необходимость снижения зависимости от внешних решений и развития собственных, обеспечивая импортонезависимость, в то же время такой подход приводит к снижению скорости развития области в целом [19].

Перечисленные критерии образуют достаточно широкое пространство признаков. Для практического сравнительного анализа в следующем разделе целесообразно выделить подмножество наиболее репрезентативных критериев. Выбранные критерии и их возможные множества значений указаны в таблице 2.

Таблица 2 - Множество значений выбранных критериев

DOI: https://doi.org/10.60797/IRJ.2026.168.52.2

Критерий	Возможные значения	Интерпретация значений
Уровень детализации сбора	Табличный, столбцовый	Табличный — фиксируются зависимости только между наборами данных/таблицами. Столбцовый — фиксируются зависимости между отдельными столбцами, включая результаты сложных выражений.
Поддержка пакетных и потоковых процессов	Только пакетный, только потоковый, комбинированный	Только пакетный — поддерживаются исключительно batch-конвейеры. Только потоковый — ориентированность на streaming-сценарии. Комбинированный — поддерживаются и пакетные, и потоковые процессы.
Уровень покрытия операций над столбцами	Отсутствует, базовые функции, поддержка UDF	Отсутствует — операции над столбцами не анализируются. Базовые функции — поддерживаются стандартные SQL/DF-операции. Поддержка UDF — дополнительно корректно обрабатывается часть пользовательских функций.
Накладные расходы	Отсутствует, существенные, фоновые	Отсутствует — влияние на производительность практически не фиксируется. Существенные — заметное увеличение времени выполнения задач, блокировка или значительное потребление ресурсов. Фоновые — асинхронный сбор, умеренная дополнительная нагрузка.
Масштабируемость и отказоустойчивость	Отсутствует, ограниченная, полная	Отсутствует — решение не рассчитано на рост нагрузки, отсутствуют механизмы отказоустойчивости. Ограниченная — доступно базовое масштабирование и минимальные механизмы отказоустойчивости. Полная — поддерживается горизонтальное масштабирование и развитые

Критерий	Возможные значения	Интерпретация значений
		механизмы отказоустойчивости.
Инфраструктурные требования	Минимальные, умеренные, высокие	Минимальные — используются только существующие компоненты платформы. Умеренные – требуется развёртывание отдельного сервиса. Высокие — необходима внешняя архитектура с несколькими сервисами и специализированными хранилищами.
Открытый исходный код	Отсутствует, частично, полностью	Отсутствует — решение полностью проприетарное. Частично — часть компонентов или функциональности доступна как open source. Полностью — ключевые компоненты решения доступны с открытым исходным кодом.

3.3. Сравнительный анализ решений по заданным критериям

3.3.1. Обзор отобранных инструментов

Spline.

Spline [12] состоит из двух компонентов: агент, встраиваемый в Spark-приложение, и отдельный сервер для сбора данных. В терминах раздела 3 реализуется plan-based подход: агент анализирует логический план выполнения и формирует граф lineage. Основная область применения — Spark-конвейеры, полученный граф может далее использоваться внешними системами метаданных. Исходный код Spline полностью открыт.

OpenLineage (Spark integration).

Реализованная интеграция формата OpenLineage со Spark работает в пространстве событийного подхода (event-based) [13]. Слушатель событий Spark формирует события фиксированного формата. Анализ логического плана выполняется внутри приложения, далее вовне передаются стандартизованные JSON-сообщения. Решение функционирует как тонкий агент, ориентированный на внешний OpenLineage-совместимый сервис. Исходный код открыт.

Spark Atlas Connector.

Spark Atlas Connector обеспечивает интеграцию Spark с платформой Apache Atlas и реализует гибридный подход: сведения о выполнении операций Spark преобразуются в метаданные, публикуемые в каталог Atlas. Основное внимание уделяется табличному уровню lineage, поддержка столбцового уровня ограничена. Исходный код открыт [20].

DataHub.

DataHub представляет собой платформу, комбинирующую lineage из различных источников [21]. Для Spark реализован специализированный агент и приём событий формата OpenLineage. В результате формируется граф столбцового уровня. Решение имеет гибридный характер (комбинация event-based и metadata-based подходов). Базовая версия распространяется с открытым исходным кодом, расширенная функциональность доступна в enterprise-редакции.

OpenMetadata.

OpenMetadata является платформой управления метаданными и lineage. Сбор осуществляется через OpenMetadata Spark Agent или через прием событий формата OpenLineage, после чего данные сохраняются в централизованном каталоге [22]. По своей природе решение также относится к гибридному классу. Столбцовый уровень поддерживается для ряда интеграций. Исходный код открыт.

Databricks Unity Catalog.

Unity Catalog является централизованным метастором крупной big data платформы Databricks [23]. Сбор lineage осуществляется при выполнении запросов, результаты сохраняются в виде связей между объектами каталога, включая столбцы. Решение глубоко интегрировано в экосистему Databricks и предоставляет специфичное для платформы представления lineage. Исходный код сервиса закрыт.

AWS Glue + Amazon Neptune + Spline.

Данная связка представляет собой архитектурный шаблон в среде AWS: Spark-задачи в Glue запускаются с агентом Spline, который собирает столбцовый lineage, далее он сохраняется в графовой СУБД Amazon Neptune. Компонент Spline открыт, в то время как сервисы Glue и Neptune являются управляемыми через AWS и закрытыми [24].

Google Cloud Dataplex.

Dataplex Universal Catalog обеспечивает сбор lineage для Spark в Dataproc и Serverless for Apache Spark посредством Data Lineage API и OpenLineage-совместимых событий [25]. Подход сочетает event-based и metadata-based

уровни: сведения агрегируются в централизованном каталоге Dataplex и доступны на табличном и столбцовом уровне. Решение предоставляется в виде управляемого сервиса, исходный код закрыт.

Collibra (в связке с OpenLineage).

Collibra относится к классу enterprise-платформ data-governance и в рассматриваемом контексте выступает потребителем технического lineage. Spark-lineage загружается в Collibra из Unity Catalog, OpenLineage-совместимых источников и иных каталогов [26]. Реализуется metadata-based подход; сбор lineage в кластере Spark платформой не выполняется, исходный код закрыт.

Monte Carlo Data.

Monte Carlo является платформой data observability, в которой lineage рассматривается как один из аналитических источников. Для Spark-процессов платформа опирается на внешние источники lineage (например, Unity Catalog) и специализируется на анализе уже собранных связей в задачах мониторинга качества и поиска первопричин инцидентов. По отношению к lineage решение выступает как metadata-based потребитель, исходный код закрыт [27].

Atlan.

Atlan представляет enterprise-каталог и платформу data-governance с акцентом на столбцовый уровень data lineage. Для Spark-конвейеров Atlan обычно использует OpenLineage, Unity Catalog и другие источники. Сбор lineage в самом кластере выполняют внешние агенты или платформенные механизмы. Таким образом, Atlan относится к метаданных-ориентированным потребителям технического lineage. Продукт проприетарный [28].

3.3.2. Классификация по типу подхода и открытости исходного кода

Для компактного представления результатов классификации удобно совместить два критерия классификации:

1. Тип технологического подхода к сбору lineage (plan-based, event-based, metadata-based, гибридный);
2. Наличие открытого исходного кода (наличие open-source-компоненты или полная закрытость решения).

На рис. 1 показано, как рассматриваемые инструменты распределяются по этим двум измерениям. В группе «С открытым исходным кодом» находятся решения, где ключевые компоненты доступны в виде open-source (Spline, OpenLineage, Spark Atlas Connector, DataHub, OpenMetadata, Apache Atlas). В группе «Без открытого исходного кода» собраны платформы, предоставляемые как проприетарный сервис (Databricks Unity Catalog, Google Cloud Dataplex, Collibra, Monte Carlo Data, Atlan, а также облачная связка AWS Glue + Amazon Neptune + Spline, где открытым остаётся только агент, но инфраструктура целиком принадлежит поставщику).

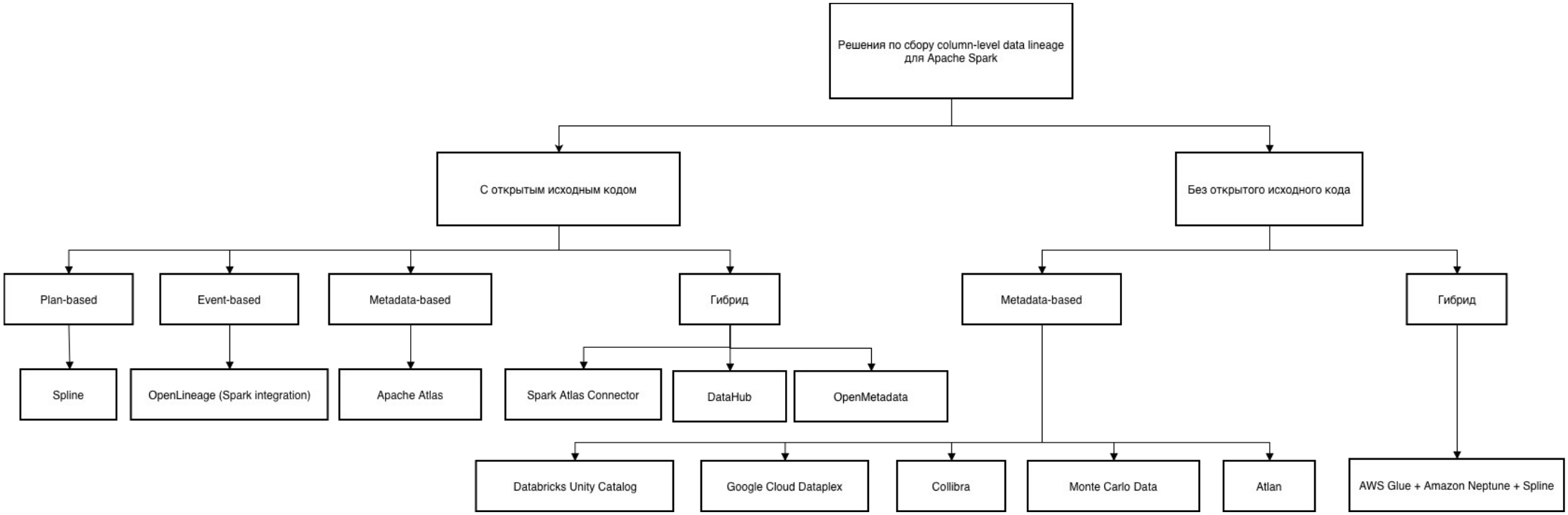

Рисунок 1 - Диаграмма классификации column lineage инструментов

DOI: https://doi.org/10.60797/IRJ.2026.168.52.3

Рисунок 1 подчёркивает несколько моментов, важных для дальнейшего обсуждения:

● plan-based и event-based подходы в текущей выборке представлены только open-source-решениями (Spline и OpenLineage);

● metadata-based и гибридные подходы реализованы как в открытых, так и в полностью проприетарных продуктах;

● облачные и enterprise-платформы lineage практически всегда оказываются в правой части схемы, тогда как низкоуровневые агенты и часть каталогов сконцентрированы в левой.

Такое разделение позволяет в следующих подразделах отдельно обсуждать технические свойства классов решений и стратегические риски, связанные с зависимостью от закрытых платформ.

3.3.3. Таблица сравнительного анализа по итоговым критериям

В таблице 3 представлены значения критериев для каждого инструмента в соответствии с перечнем в разделе 3.2.

Таблица 3 - Сравнительный анализ решений по сбору lineage

DOI: https://doi.org/10.60797/IRJ.2026.168.52.4

Инструмент	Уровень детализации сбора	Поддержка пакетных и потоковых процессов	Уровень покрытия операций над столбцами	Накладные расходы	Масштабируемость и отказоустойчивость	Инфраструктурные требования	Открытый исходный код
Spline	столбцовый	комбинированный	базовые функции	фоновые	ограниченная	умеренные	полностью
OpenLineage (Spark integration)	столбцовый	комбинированный	базовые функции	фоновые	ограниченная	умеренные	полностью
Spark Atlas Connector	табличный	только пакетный	отсутствует	фоновые	ограниченная	высокие	полностью
DataHub	столбцовый	комбинированный	базовые функции	фоновые	ограниченная	высокие	частично

Инструмент	Уровень детализации сбора	Поддержка пакетных и потоковых процессов	Уровень покрытия операций над столбцами	Накладные расходы	Масштабируемость и отказоустойчивость	Инфраструктурные требования	Открытый исходный код
OpenMetadata	столбцовый	комбинированный	базовые функции	фоновые	ограниченная	умеренные	полностью
Apache Atlas	табличный	только пакетный	отсутствует	фоновые	ограниченная	высокие	полностью
Databricks Unity Catalog	столбцовый	комбинированный	базовые функции	фоновые	полная	минимальные	отсутствует
AWS Glue + Amazon Neptune + Spline	столбцовый	комбинированный	базовые функции	фоновые	полная	высокие	частично
Google Cloud Dataplex (Dataproc / Serverless Spark)	столбцовый	комбинированный	базовые функции	фоновые	полная	минимальные	отсутствует

Инструмент	Уровень детализации сбора	Поддержка пакетных и потоковых процессов	Уровень покрытия операций над столбцами	Накладные расходы	Масштабируемость и отказоустойчивость	Инфраструктурные требования	Открытый исходный код
Collibra (с OpenLineage)	столбцовый	только пакетный	базовые функции	отсутствует	полная	высокие	отсутствует
Monte Carlo Data	столбцовый	комбинированный	базовые функции	отсутствует	полная	минимальные	отсутствует
Atlan	столбцовый	комбинированный	базовые функции	отсутствует	полная	умеренные	отсутствует

Проведенный анализ показал, что среди рассмотренных кандидатов ни один не является идеальным с точки зрения выдвинутых критериев сравнения. Plan-based инструменты в наибольшей степени интегрированы с вычислительной моделью Spark, а также не лишены поддержки как пакетных, так и непрерывных процессов. Их преимуществом является относительная технологическая независимость от конкретных платформ, а недостатком — необходимость развёртывания собственной инфраструктуры хранения и самостоятельной интеграции с системами управления метаданными.

Системы, относящиеся к метахабам (DataHub и OpenMetadata) выгодно выделяются за счет возможностей комбинации связей внутри Spark-процессов с другими СУБД и BI-системами, расширяя тем самым контекст для анализа происхождения данных, однако для них характерна повышенная сложность и значительные требования к квалификации сотрудников, отвечающих за настройку процессов загрузки метаданных.

Платформенные решения (Databricks Unity Catalog, AWS Glue + Amazon Neptune + Spline, Google Cloud Dataplex) характеризуются высокой масштабируемостью и умеренными инфраструктурными требованиями. Основное ограничение здесь связано с жёсткой привязкой к конкретным поставщикам и ограниченной возможностью переноса решений в другие окружения.

Enterprise-платформы data-governance и observability (Collibra, Monte Carlo Data, Atlan) обеспечивают развитые средства визуализации, управления доступом и анализа влияния изменений. Однако в контексте Spark они выступают преимущественно потребителями технического lineage, зависящими от наличия внешних агентов или платформенных механизмов его сбора.

Данный анализ показал, что выбор решения для реализации столбцового уровня data lineage в корпоративных конвейерах на базе Apache Spark представляет собой компромисс между глубиной технической интеграции, накладными расходами, степенью зависимости от конкретной платформы, возможностями интеграции с контурами data-governance и требованиями к открытости исходного кода.

Обсуждение

4.1. Ограничения и риски существующих решений

Проведенный анализ и изучение предметной области позволили выявить ряд ограничений, свойственных современным решениям.

1) Неполнота столбцового уровня. Практически все системы заявили поддержку column-level lineage, но фактически обеспечили его только для стандартных SQL-операций и простых выражений. При использовании комплексных конструкций, UDF или пользовательских функций, зависимости не могут быть обнаружены.

2) Сильная зависимость гибридных решений от облачных компонент и поставляемых вендором технологий. Databricks Unity Catalog, Google Cloud Dataplex и связка AWS Glue + Amazon Neptune + Spline демонстрируют высокую масштабируемость, отказоустойчивость и низкие собственные требования к дополнительной инфраструктуре. Ценой за богатый функционал становится жёсткая привязка к конкретному поставщику услуги. Для компаний, которые вынуждены учитывать вопросы юрисдикции и импортонезависимости, этот фактор становится серьёзным риском.

3) Enterprise-решения data-governance и observability (Collibra, Atlan, Monte Carlo Data) показали лучший уровень отказоустойчивости и пользовательского удобства. Эти платформы хорошо решают задачи визуализации lineage, работы с бизнес-метаданными, аудита и анализа влияния изменений, но сами по себе не собирают lineage для Apache Spark. Они полностью зависят от слоя технического lineage, который должен предоставляться извне. В результате общая архитектура усложняется за счет комбинации различных инструментов.

4) Закрытость исходного кода значительной части решений. Проприетарные сервисы (Databricks Unity Catalog, Google Cloud Dataplex, Collibra, Monte Carlo Data, Atlan) имеют общеизвестный перечень недостатков. К ним можно отнести отсутствие возможности самостоятельной реализации доработок под внутренние требования, сложности в проведении внутреннего аудита и риски в перспективах долгосрочного сопровождения.

Указанные ограничения показывают актуальность построения комбинированной архитектуры сбора lineage, а также критичность стратегического планирования при определении целевого решения.

4.2. Требования к перспективным архитектурам столбцового data lineage

Опираясь на выявленные ограничения и систему критериев, можно сформулировать профиль “целевой” архитектуры столбцового lineage. С точки зрения уровня детализации и покрытия операций целесообразно ориентироваться на устойчивый столбцовый уровень lineage для основных сценариев Spark - как пакетных, так и потоковых. Это означает поддержку как стандартных SQL-операций, так и пользовательских функций и сложных выражений. В нашей терминологии такое решение должно стремиться к значениям «столбцовый» и «поддержка UDF» при «комбинированной» поддержке процессов. Решения должны стремиться к минимизации замедления рабочих процессов, а инфраструктура должна линейно масштабироваться при росте количества задач и объёма метаданных. Инфраструктурные требования в целевой архитектуре желательно удерживать на уровне минимальных или умеренных. Речь идёт о повторном использовании существующих компонентов кластеров (каталоги данных, брокеры сообщений, СУБД) и ограничении количества специализированных сервисов, которые нужно развернуть и сопровождать только ради lineage.

Отдельно стоит подчеркнуть важность открытого исходного кода. В российских условиях управление рисками всё чаще требует опоры на решения, которые можно развивать и модифицировать независимо от внешних поставщиков. Перспективная архитектура должна использовать открытые компоненты в качестве ядра (агенты, модели, хранилища lineage), а при необходимости дополняться проприетарными системами на уровне потребления lineage, а не сбора.

Заключение

Цель работы заключалась в обзоре и классификации подходов и инструментов сбора столбцового уровня data lineage в корпоративных конвейерах на базе Apache Spark, а также в формализации системы критериев их сравнения.

Основные результаты можно резюмировать следующим образом.

Во-первых, работа вводит и обосновывает классификацию подходов к сбору lineage на три базовых подхода и один гибридный. Современные системы в большинстве случаев комбинируют указанные подходы, при этом основным отличием является способ интеграции в архитектуру платформы.

Во-вторых, предлагается система критериев оценки со строгим набором допустимых значений. Такая формализация делает возможным воспроизводимое расширение набора инструментов при необходимости в будущих исследованиях.

В-третьих, фиксируется сравнительный анализ двенадцати инструментов и архитектурных решений. Сводная таблица 3 показывает ключевые сходства и отличия решений без необходимости глубокого технического анализа в будущем.

С практической точки зрения, предложенная система может служить ориентиром при выборе и комбинировании инструментов для своих Spark-конвейеров. Систему оценки возможно использовать при планировании миграции от проприетарных решений к архитектурам на базе открытых компонентов.

Дальнейшее развитие работы может идти по нескольким направлениям. Имеет смысл разработать и экспериментально оценить прототип гибридного решения для столбцового lineage в Apache Spark, основанный преимущественно на открытых инструментах. Также требуется более формально описать модель полноты и корректности column-level lineage в терминах логического плана и событий выполнения. В-третьих, представляется важным провести серию экспериментов на реальных производственных конвейерах, чтобы количественно оценить накладные расходы различных подходов и подтвердить применимость предложенной системы критериев в индустриальной практике.

Конфликт интересов

Не указан.

Рецензия

Все статьи проходят рецензирование. Но рецензент или автор статьи предпочли не публиковать рецензию к этой статье в открытом доступе. Рецензия может быть предоставлена компетентным органам по запросу.

Conflict of Interest

None declared.

Review

All articles are peer-reviewed. But the reviewer or the author of the article chose not to publish a review of this article in the public domain. The review can be provided to the competent authorities upon request.

Список литературы / References

1. Bhushan V. Data-Driven Decision-Making: Leveraging Analytics for Performance Improvement / V. Bhushan, G. Lakshmi, S. Dhivya devi et al. // Journal of Informatics Education and Research. — 2024. — 4. — DOI: 10.52783/jier.v4i3.1298

2. Olayinka O.H. Big data integration and real-time analytics for enhancing operational efficiency and market responsiveness / O.H. Olayinka // *International Journal of Science and Research Archive*. — 2021. — 4. — P. 280–296. — DOI: 10.30574/ijsra.2021.4.1.0179

3. Galster M. Towards Understanding Provenance in Industry / M. Galster, J. Dietrich // *ArXiv*. — 2023. — 2302.06038v1. — DOI: 10.48550/arxiv.2302.06038

4. Tan Z. A Column-Level Data Lineage Processing System Based on Hive / Z. Tan, E. Haihong, M. Song. // International Conference on Big Data; — New York, Ny, Usa: Association for Computing Machinery, 2020. — P. 47–52. doi: 10.1145/3422713.3422719

5. Wang L. Attribute level lineage in uncertain data with dependencies / L. Wang, Z. Peng // *Wuhan University Journal of Natural Sciences*. — 2016. — 21. — P. 376–386. — DOI: 10.1007/s11859-016-1184-3

6. Documentation // Apache Spark. — 2025. — URL: https://spark.apache.org/documentation.html. (дата обращения: 12.11.25)

7. Lin Y. Efficient Row-Level Lineage Leveraging Predicate Pushdown / Y. Lin, C. Yan // ArXiv. — 2024. — abs/2412.16864. — DOI: 10.48550/arxiv.2412.16864

8. Naphade D. The Evolution and Modernization of Data Pipeline Architectures / D. Naphade // European Journal of Computer Science and Information Technology. — 2025. — 13. — DOI: 10.37745/ejcsit.2013/vol13n64253

9. Shubham S. From Traditional Data Warehouses to Lakehouse Architectures: Tackling Modern Data Challenges / S. Shubham // International Journal of Scientific Research in Computer Science Engineering and Information Technology. — 2025. — 11. — P. 2142–2155. — DOI: 10.32628/cseit251112230

10. Adamov A. Large-scale Data Modelling in Hive and Distributed Query Processing using MapReduce and Tez / A. Adamov // ArXiv. — 2023. — abs/2301.12454. — DOI: 10.48550/arXiv.2301.12454

11. Schoenenwald A. Collecting and visualizing data lineage of Spark jobs / A. Schoenenwald, S. Kern, J. Viehhauser et al. // Datenbank Spektrum. — 2021. — 21. — P. 179–189. — DOI: 10.1007/s13222-021-00387-7

12. Data Lineage Tracking And Visualization Solution // Spline. — 2025. — URL: https://absaoss.github.io/spline/. (дата обращения: 03.11.25)

13. Documentation // OpenLineage. — 2025. — URL: https://openlineage.io. (дата обращения: 27.10.25)

14. Gayakwad M. Real-Time Clickstream Analytics with Apache / M. Gayakwad, T. Patil, P. Paygude et al. // Journal of Electrical Systems. — 2024. — 20. — DOI: 10.52783/jes.1466

15. Documentation // Databricks Unity Catalog. — 2025. — URL: https://www.databricks.com/product/unity-catalog. (дата обращения: 13.11.25)

16. Rao B. SODA: A Semantics-Aware Optimization Framework for Data-Intensive Applications Using Hybrid Program Analysis / B. Rao, Z. Liu, H. Zhang et al. // ArXiv. — 2021. — abs/2107.11536. — DOI: 10.48550/arXiv.2107.11536

17. Chen Y. An Empirical Study on Core Data Asset Identification in Data Governance / Y. Chen, Y. Zhao, W. Xie et al. // Big Data and Cognitive Computing. — 2023. — 7. — DOI: 10.3390/bdcc7040161

18. Wang Z. Efficient Fault Tolerance for Pipelined Query Engines via Write-ahead Lineage / Z. Wang, A. Aiken // IEEE 40th International Conference on Data Engineering. — 2024. — 40. — P. 436–448. — DOI: 10.1109/ICDE60146.2024.00040

19. Туманян Ю.Р. Импортозамещение как фактор инновационного развития экономики России / Ю.Р. Туманян, М.А. Индустриев // Известия Саратовского Университета. Новая Серия. Серия: Экономика. Управление. Право. — 2022. — 4. — С. 396-405. — DOI: 10.18500/1994-2540-2022-22-4-396-405

20. Spark Atlas Connector // GitHub. — 2025. — URL: https://github.com/hortonworks-spark/spark-atlas-connector. (дата обращения: 08.11.25)

21. Spark Connector Documentation // DataHub. — 2025. — URL: https://docs.datahub.com/docs/metadata-integration/java/acryl-spark-lineage. (дата обращения: 10.11.25)

22. Spark Lineage // OpenMetaData. — 2025. — URL: https://docs.open-metadata.org/latest/connectors/ingestion/lineage/spark-lineage. (дата обращения: 10.11.25)

23. Data Lineage Docs // DataBricks Unity Catalog. — 2025. — URL: https://docs.databricks.com/aws/en/data-governance/unity-catalog/data-lineage. (дата обращения: 10.11.25)

24. Nguyen K. Build data lineage for data lakes using AWS Glue, Amazon Neptune, and Spline / K. Nguyen, K. Balasubramaniyan, R. Shaurya // Amazon Blog. — 2022. — URL: https://aws.amazon.com/ru/blogs/big-data/build-data-lineage-for-data-lakes-using-aws-glue-amazon-neptune-and-spline/. (дата обращения: 04.11.25)

25. Enable Spark data lineage in Dataproc // Dataproc Google Cloud Docs. — 2025. — URL: https://docs.cloud.google.com/dataproc/docs/guides/spark-lineage. (дата обращения: 08.11.25)

26. Campabadal M. Uncover data blindspots with OpenLineage / M. Campabadal, K. Van Coillie, P. Vitovec // Collibra blog. — 2025. — URL: https://www.collibra.com/blog/uncover-data-blindspots-with-openlineage. (дата обращения: 03.11.25)

27. Spark // Monte Carlo. — 2025. — URL: https://docs.getmontecarlo.com/docs/spark. (дата обращения: 13.11.25)

28. Spark // Atlan. — 2024. — URL: https://developer.atlan.com/models/spark/. (дата обращения: 05.11.25)

Список литературы на английском языке / References in English

1. Bhushan V. Data-Driven Decision-Making: Leveraging Analytics for Performance Improvement / V. Bhushan, G. Lakshmi, S. Dhivya devi et al. // Journal of Informatics Education and Research. — 2024. — 4. — DOI: 10.52783/jier.v4i3.1298

2. Olayinka O.H. Big data integration and real-time analytics for enhancing operational efficiency and market responsiveness / O.H. Olayinka // *International Journal of Science and Research Archive*. — 2021. — 4. — P. 280–296. — DOI: 10.30574/ijsra.2021.4.1.0179

3. Galster M. Towards Understanding Provenance in Industry / M. Galster, J. Dietrich // *ArXiv*. — 2023. — 2302.06038v1. — DOI: 10.48550/arxiv.2302.06038

4. Tan Z. A Column-Level Data Lineage Processing System Based on Hive / Z. Tan, E. Haihong, M. Song. // International Conference on Big Data; — New York, Ny, Usa: Association for Computing Machinery, 2020. — P. 47–52. doi: 10.1145/3422713.3422719

5. Wang L. Attribute level lineage in uncertain data with dependencies / L. Wang, Z. Peng // *Wuhan University Journal of Natural Sciences*. — 2016. — 21. — P. 376–386. — DOI: 10.1007/s11859-016-1184-3

6. Documentation // Apache Spark. — 2025. — URL: https://spark.apache.org/documentation.html. (accessed: 12.11.25)

7. Lin Y. Efficient Row-Level Lineage Leveraging Predicate Pushdown / Y. Lin, C. Yan // ArXiv. — 2024. — abs/2412.16864. — DOI: 10.48550/arxiv.2412.16864

8. Naphade D. The Evolution and Modernization of Data Pipeline Architectures / D. Naphade // European Journal of Computer Science and Information Technology. — 2025. — 13. — DOI: 10.37745/ejcsit.2013/vol13n64253

9. Shubham S. From Traditional Data Warehouses to Lakehouse Architectures: Tackling Modern Data Challenges / S. Shubham // International Journal of Scientific Research in Computer Science Engineering and Information Technology. — 2025. — 11. — P. 2142–2155. — DOI: 10.32628/cseit251112230

10. Adamov A. Large-scale Data Modelling in Hive and Distributed Query Processing using MapReduce and Tez / A. Adamov // ArXiv. — 2023. — abs/2301.12454. — DOI: 10.48550/arXiv.2301.12454

11. Schoenenwald A. Collecting and visualizing data lineage of Spark jobs / A. Schoenenwald, S. Kern, J. Viehhauser et al. // Datenbank Spektrum. — 2021. — 21. — P. 179–189. — DOI: 10.1007/s13222-021-00387-7

12. Data Lineage Tracking And Visualization Solution // Spline. — 2025. — URL: https://absaoss.github.io/spline/. (accessed: 03.11.25)

13. Documentation // OpenLineage. — 2025. — URL: https://openlineage.io. (accessed: 27.10.25)

14. Gayakwad M. Real-Time Clickstream Analytics with Apache / M. Gayakwad, T. Patil, P. Paygude et al. // Journal of Electrical Systems. — 2024. — 20. — DOI: 10.52783/jes.1466

15. Documentation // Databricks Unity Catalog. — 2025. — URL: https://www.databricks.com/product/unity-catalog. (accessed: 13.11.25)

16. Rao B. SODA: A Semantics-Aware Optimization Framework for Data-Intensive Applications Using Hybrid Program Analysis / B. Rao, Z. Liu, H. Zhang et al. // ArXiv. — 2021. — abs/2107.11536. — DOI: 10.48550/arXiv.2107.11536

17. Chen Y. An Empirical Study on Core Data Asset Identification in Data Governance / Y. Chen, Y. Zhao, W. Xie et al. // Big Data and Cognitive Computing. — 2023. — 7. — DOI: 10.3390/bdcc7040161

18. Wang Z. Efficient Fault Tolerance for Pipelined Query Engines via Write-ahead Lineage / Z. Wang, A. Aiken // IEEE 40th International Conference on Data Engineering. — 2024. — 40. — P. 436–448. — DOI: 10.1109/ICDE60146.2024.00040

19. Tumanyan Yu.R. Importozameshhenie kak faktor innovacionnogo razvitiya e'konomiki Rossii [Import substitution as a factor in the innovative development of the Russian economy] / Yu.R. Tumanyan, M.A. Industriev // Proceedings of Saratov University. New Series. Series: Economics. Management. Law. — 2022. — 4. — P. 396-405. — DOI: 10.18500/1994-2540-2022-22-4-396-405 [in Russian]

20. Spark Atlas Connector // GitHub. — 2025. — URL: https://github.com/hortonworks-spark/spark-atlas-connector. (accessed: 08.11.25)

21. Spark Connector Documentation // DataHub. — 2025. — URL: https://docs.datahub.com/docs/metadata-integration/java/acryl-spark-lineage. (accessed: 10.11.25)

22. Spark Lineage // OpenMetaData. — 2025. — URL: https://docs.open-metadata.org/latest/connectors/ingestion/lineage/spark-lineage. (accessed: 10.11.25)

23. Data Lineage Docs // DataBricks Unity Catalog. — 2025. — URL: https://docs.databricks.com/aws/en/data-governance/unity-catalog/data-lineage. (accessed: 10.11.25)

24. Nguyen K. Build data lineage for data lakes using AWS Glue, Amazon Neptune, and Spline / K. Nguyen, K. Balasubramaniyan, R. Shaurya // Amazon Blog. — 2022. — URL: https://aws.amazon.com/ru/blogs/big-data/build-data-lineage-for-data-lakes-using-aws-glue-amazon-neptune-and-spline/. (accessed: 04.11.25)

25. Enable Spark data lineage in Dataproc // Dataproc Google Cloud Docs. — 2025. — URL: https://docs.cloud.google.com/dataproc/docs/guides/spark-lineage. (accessed: 08.11.25)

26. Campabadal M. Uncover data blindspots with OpenLineage / M. Campabadal, K. Van Coillie, P. Vitovec // Collibra blog. — 2025. — URL: https://www.collibra.com/blog/uncover-data-blindspots-with-openlineage. (accessed: 03.11.25)

27. Spark // Monte Carlo. — 2025. — URL: https://docs.getmontecarlo.com/docs/spark. (accessed: 13.11.25)

28. Spark // Atlan. — 2024. — URL: https://developer.atlan.com/models/spark/. (accessed: 05.11.25)