Сравнительный анализ MapReduce-алгоритмов (Hadoop/Spark) для оценки панельных данных

Котин Иван Станиславович
аспирант, Санкт-Петербургский государственный экономический университет, kotin2000@mail.ru

В работе представлен сравнительный анализ двух ведущих реализаций парадигмы распределенных вычислений MapReduce – Apache Hadoop и Apache Spark – применительно к задачам эконометрического оценивания на массивных панельных данных. Развитие информационных систем привело к формированию наборов объемом в десятки и сотни гигабайт, обработка которых требует горизонтального масштабирования. Фундаментальное различие между платформами заключается в стратегии управления промежуточными данными: классический Hadoop опирается на жесткую материализацию результатов каждого этапа на дисках (HDFS), что обеспечивает высокую отказоустойчивость для однопроходных ETL-конвейеров, но становится критическим узким местом при итерационных алгоритмах из-за многократного перечитывания данных. В отличие от него, Apache Spark реализует резидентную модель вычислений в оперативной памяти через концепцию устойчивых распределенных наборов данных (RDD) и DataFrame API. Это позволяет кэшировать данные между итерациями, минимизируя дорогостоящий сетевой обмен и дисковый ввод-вывод, что кардинально меняет экономику высокопроизводительных вычислений для методов максимального правдоподобия и численной оптимизации. Существенным фактором практической реализации выступает зрелость экосистемы. Высокоуровневые API Spark SQL, оптимизатор Catalyst и библиотека MLlib значительно снижают трудозатраты на разработку сложных эконометрических моделей по сравнению с низкоуровневым программированием функций map/reduce в Java для Hadoop. Результаты исследования подтверждают тезис о взаимодополняемости платформ: надежность дисковой модели Hadoop сочетается с интерактивностью Spark. Перспективы дальнейших исследований связаны с адаптацией подходов к гетерогенным средам с GPU-ускорителями и разработкой фреймворков автоматической генерации оптимального плана исполнения

Ключевые слова: MapReduce, Hadoop, Spark, DataFrame, SQL, Big Data.

Введение

Развитие информационных финансовых и экономических систем привело к формированию массивов данных, объём которых измеряется миллионами наблюдений, характеризуется панельной (продольной) структурой [7]: каждая единица совокупности наблюдается в течение нескольких последовательных периодов времени. Панельные данные, объединяющие признаки микропеременных (индивидуальные) компоненты и временные (хронологические) компоненты, представляют собой основу для построения широкого класса эконометрических моделей – от классической линейной регрессии с фиксированными эффектами до динамических моделей с инструментальными переменными и многопараметрических моделей стохастических границ. Традиционные средства статистического анализа, функционирующие в рамках модели «одна машина – одна память», сталкиваются с принципиальными ограничениями при обработке панельных данных объёмом порядка десятков и сотен гигабайт. Во-первых, ограничение объёма оперативной памяти не позволяет разместить матрицу плана целиком. Во-вторых, время последовательных вычислений, требуемых для обращения матриц размерности, сопоставимой с числом единиц наблюдений, становится недопустимо большим. В-третьих, процедуры предобработки (очистка, заполнение пропусков, масштабирование признаков) [4, 5], предшествующие оцениванию, сами по себе требуют многократного обхода данных. Парадигма MapReduce, предложенная в начале 2000-х годов и реализованная в открытой экосистеме Hadoop, а затем существенно усовершенствованная в рамках вычислительного движка Apache Spark, предоставляет механизм горизонтального масштабирования вычислений на кластер из тысяч узлов.

Принципиальной особенностью данной парадигмы является декомпозиция вычислительной задачи на две стадии – map, на которой исходные записи преобразуются в промежуточные пары «ключ-значение», и reduce, на которой промежуточные значения, сгруппированные по ключам, агрегируются в итоговый результат. Такая схема оказывается естественной для целого класса задач оценивания, сводящихся к вычислению сумм, свёрток и рекуррентных агрегатов, лежащих в основе методов наименьших квадратов и методов максимального правдоподобия. Целью настоящей работы является сравнительный анализ возможностей и ограничений двух ведущих реализаций парадигмы MapReduce – Hadoop и Apache Spark.

Парадигма MapReduce и её реализации

Базовая модель Hadoop MapReduce

Классическая модель MapReduce, реализованная в Hadoop, определяет вычисление через пару функций, задаваемых пользователем: $map\colon (k, v) \to \{(k'_j, v'_j)\}_{j=1}^{m}$, $reduce\colon (k', \{v'_1, \dots, v'_{m'}\}) \to (k'', v'')$. Выполнение задачи заключается в следующих фазах:

чтение входных блоков с распределённой файловой системы HDFS и применение функции map к каждой записи;

сортировка и перемешивание промежуточных пар по ключу с распределением групп между узлами-редьюсерами;

применение функции reduce к каждой группе и запись результата обратно в HDFS.

Ключевой характеристикой классической модели является материализация промежуточных данных на диске между стадиями [3]. Каждое выполнение задачи map образует локальную сортировку, после которой данные передаются по сети в редьюсеры, где вновь сортируются и затем обрабатываются. Для задачи, требующей L проходов по данным (таких как итерационные процедуры), общий объём дискового ввода-вывода и сетевого трафика растёт пропорционально L, что при больших L делает использование Hadoop MapReduce крайне неэффективным.

Модель Apache Spark

Apache Spark развивает парадигму MapReduce, вводя концепцию устойчивого распределённого набора данных (Resilient Distributed Dataset, RDD) и его неявной формы - DataFrame. В отличие от Hadoop, Spark выполняет вычисления «в памяти»: промежуточные данные не материализуются на диск между стадиями, если их объём умещается в оперативной памяти кластера. Это достигается введением направленного ациклического графа вычислений, который планировщик исполняет с удержанием промежуточных партиций в памяти. Для итерационных алгоритмов, при которых одна и та же выборка данных многократно проходится на каждой итерации, разница становится определяющей. Если в Hadoop каждая итерация требует перечитывания исходных данных с диска и повторного перешивания, то в Spark исходный набор, может быть, один раз распределён по узлам и «закэширован» в памяти; последующие итерации работают только с кэшированными партициями без обращения к диску и минимизируют сетевые передачи. Формально, если стоимость одного прохода через функцию map-reduce в Hadoop равна C_H (включая дисковый ввод-вывод и перемешивание), а в Spark стоимость кэшированной итерации равна C_S, то суммарная стоимость L итераций:

$$C_H(L) = L * C_H$$
$$C_S(L) = C_S^{(0)} + L * C_S,$$

где $C_S^{(0)}$ - однократная стоимость распределения и кэширования данных. Поскольку практически $C_S < C_H$ (в соотношении, зависящем от отношения пропускной способности диска к пропускной способности оперативной памяти), при достаточно большом L выполняется $C_S(L) < C_H(L)$, что и определяет преимущество Spark для итерационных алгоритмов.

Сравнительная характеристика архитектур

Таблица 1
Сравнение Hadoop MapReduce и Apache Spark.

Критерий	Hadoop	Apache Spark
Модель исполнения	Линейная	Граф
Промежуточные данные	Материализуются на диск HDFS	Кэшируются в памяти
Устойчивость к сбоям	Перезапуск редьюсеров с диска	Пересчет потерянной партиции
Шифтинг	Сортировка и дисковое перемещение	Network shuffle с возможностью агрегации
Число проходов по данным	Один проход	Множество итераций

Сопоставление архитектурных особенностей двух реализаций представлено в таблице 1. Из таблицы следует, что Hadoop и Spark не являются взаимозаменяемыми в общем. случае: выбор инструментария должен определяться характером вычислительной задачи.

Для количественного сравнения введём модель затрат, учитывающую четыре компонента: стоимость дискового ввода-вывода, стоимость передачи данных по сети, стоимость вычислений и накладные расходы на планирование. Пусть единицей измерения служит стоимость обработки одного байта в операциях условного эталонного кластера. Обозначим через c_d удельную стоимость дискового ввода-вывода, c_n- удельную стоимость передачи байта по сети, c_p – удельную стоимость вычисления в расчёте на операцию, а c_o - фиксированные накладные расходы на запуск одной стадии. Для задачи, требующей L проходов по данным объема V с объемом перемешивания V_{sh}, суммарные затраты в Hadoop оцениваются как $C_H(L) = L\big(Vc_d + V_{sh}c_n + nc_p\big) + Lc_o^H$,в то время как затраты в Spark при условии полного удержания данных в памяти $C_S(L) = Vc_d^0 + L\big(Vc_m + V_{sh}^{iter}c_n + nc_p\big) + Lc_o^S$ где c_m – удельная стоимость работы с байтом в оперативной памяти, c_d^0 - однократная стоимость первичной загрузки, V_{sh}^{iter} - объём перемешивания на итерацию. Поскольку выполняются соотношения $c_m \ll c_d$ и $V_{sh}^{iter} \le V_{sh}$, разность затрат линейно растёт с числом итераций:

$$C_H(L) - C_S(L) =$$
$$L\big[V(c_d - c_m) + \big(V_{sh} - V_{sh}^{(iter)}\big)c_n + \big(c_o^{(H)} - c_o^{(S)}\big)\big] - Vc_d^{(0)}$$

Уравнение $C_H(L) = C_S(L)$ определяет пороговое число проходов

$$L^* = \frac{Vc_d^0}{V(c_d - c_m) + \big(V_{sh} - V_{sh}^{iter}\big)c_n + \Delta c_o}$$

При $L > L^*$ Spark вычислительно оправдан; при $L < L^*$ - более рационально использование Hadoop при условии низких накладных расходов на планирование $c_o^H < c_o^S$. Полученная модель наглядно объясняет качественные выводы: поскольку в современных кластерах отношение c_d/c_m достигает нескольких порядков (диск медленнее оперативной памяти), порог L^* оказывается малым, порядка единиц-десятков итераций. Это означает, что практически всякий итерационный алгоритм оценивания панельных моделей выигрывает от применения Spark, тогда как для строго однопроходных стандартных операций выигрыш незначителен или отсутствует.

Сравнительный анализ производительности

Критерии сравнения

Для объективного сравнения инструментов необходимо зафиксировать семейство показателей: (1) время полного исполнения задачи t_{total}; (2) объём дискового ввода-вывода V_{disk}; (3) объём сетевого трафика V_{net}; (4) коэффициент масштабируемости ξ; (5) накладные расходы на планирование. Коэффициент масштабируемости определяется отношением ускорения к числу узлов:

$$\xi = \frac{t_{total}(1)}{t_{total}(P)} * \frac{1}{P},$$

где $t_{total}(1)$ – время исполнения на одном узле, $t_{total}(P)$ – на P узлах. При идеальной масштабируемости $\xi = 1$; на практике $0 < \xi < 1$, поскольку коммуникационные издержки и дисбаланс растут с числом узлов.

Результаты теоретического анализа

Для однопроходной задачи суммарное время в Hadoop определяется стоимостью чтения данных с диска, их перемешивания и записи результатов:

$$t_H = \frac{V_{disk}}{W_{disk}} + \frac{V_{net}}{W_{net}} + O_H,$$

где W_{disk}, W_{net} – пропускные способности дискового и сетевого каналов соответственно, O_H- накладные расходы на запуск задач. Для того же объёма данных в Spark при условии полного размещения в памяти

$$t_S = \frac{V_{mem}}{W_{mem}} + \frac{V_{net}}{W_{net}} + O_S,$$

где W_{mem} – пропускная способность оперативной памяти, V_{mem} – объём передаваемых в память данных. Поскольку $W_{mem} \gg W_{disk}$, даже в однопроходных задачах Spark при прочих равных обеспечивает выигрыш по времени за счёт устранения дискового ввода-вывода, однако этот выигрыш частично нивелируется более высокими накладными расходами на планирование графов ($O_S > O_H$) и необходимостью первичной загрузки и сериализации данных.

В итерационных задачах разница усиливается. Формально суммарное время для числа итераций L:

$$t_H(L) = L\left(\frac{V_{disk}}{W_{disk}} + \frac{V_{net}}{W_{net}}\right) + L * O_H + t_{load}^{(H)},$$
$$t_S(L) = L\left(\frac{V_{mem}}{W_{mem}} + \frac{V_{net}^{(iter)}}{W_{net}}\right) + L * O_S + t_{load}^{(S)},$$

где $V_{net}^{(iter)}$ – объём перемешивания на итерации (в общем случае меньше полного объёма данных). Анализ этих соотношений показывает, что при $L > 10 \div 20$ критическим становится слагаемое $L * \frac{V_{disk}}{W_{disk}}$, отсутствующее в выражении для Spark, что обеспечивает последнему доминирование.

Рекомендации по выбору инструментария

Существенным фактором, влияющим на практический выбор инструмента, является наличие высокоуровневых библиотек, сокращающих трудозатраты на реализацию распределённого оценивания. Экосистема Hadoop, помимо реализации MapReduce, предлагает набор инструментов, в частности пакет программ для машинного обучения Mahout, который, однако, исторически ограничен поддержкой ограниченного перечня алгоритмов и не содержит специализированных средств эконометрического оценивания. Вследствие этого реализация моделей панельных данных в Hadoop требует ручного написания функций map и reduce, а также управления планированием промежуточных заданий.

Экосистема Spark предоставляет более развитые средства: Scala/Python API, язык запросов Spark SQL поверх DataFrame, а также библиотеку машинного обучения MLlib [9]. Высокоуровневое выражение вычислений через SQL-агрегации и операции реляционного соединения сопровождается автоматической оптимизацией плана исполнения (Catalyst optimizer), которая способна устранять избыточные перемешивания, сворачивать фиксированные операции и использовать широковещательные соединения при малом размере эталонной таблицы. В Hadoop аналогичная оптимизация возможна лишь вручную, что увеличивает трудоёмкость и риск ошибок при построении конвейера. Наличие готовых библиотечных примитивов делает Spark предпочтительным инструментом с точки зрения скорости разработки и сопровождения специфических эконометрических конвейеров. При этом для стандартных, хорошо формализуемых потоков обработки и наличия опытной команды, работающей с дисковой моделью, Hadoop сохраняет конкурентоспособность за счёт простоты и предсказуемости конечного плана исполнения.

Проведённый анализ позволяет сформулировать практические критерии выбора платформы Hadoop MapReduce иили Spark:

Простота задачи и дефицит памяти: если оценивание сводится к одному или немногим проходам по данным, а кластер не имеет избыточных ресурсов оперативной памяти, Hadoop MapReduce остаётся надёжным выбором. Он обеспечивает предсказуемые профили нагрузки на дисковую подсистему. [6]

Итерационность и кэширование: если процедура оценивания итерационна, а объём данных умещается в суммарную оперативную память кластера, предпочтителен Spark. Он сокращает число обращений к диску и сетевому перемешиванию за счёт эффективного кэширования RDD и DataFrame.

Большие данные: если объём данных существенно превышает суммарную память кластера, необходимо применять гибридные стратегии. К ним относятся использование уровня персистентности MEMORY_AND_DISK со стратегией пересчёта в Spark либо переход к потоковой обработке (Spark Streaming) для обновляемых панелей.

Помимо базовых критериев, следует учитывать особенности структуры данных и инфраструктуры:

Неравномерная нагрузка (Data Skew): для панелей с сильным перекосом по числу наблюдений целесообразно предварительное выравнивание нагрузки посредством составных ключей независимо от выбранного инструмента.

Коммуникационные издержки: при доминирующей роли расходов на перемешивание данных в кластере с медленной сетью следует оценивать пороговое число итераций L^*. Правило выбора инструмента зависит от величины фактического числа итераций L относительно порога L^*.

Гибридная архитектура: при построении сложного конвейера оценивания рекомендуется разделять однопроходные и итерационные этапы. Этапы центрирования исходных переменных и построения агрегатов эффективнее выполнять в Hadoop (или в однопроходном Spark-проходе), тогда как последующие итерационные этапы оптимизации — в Spark с кэшированием полученных остатков.

Заключение

Проведенное в настоящей работе комплексное исследование архитектурных особенностей, вычислительных моделей и практических сценариев применения двух ведущих реализаций парадигмы распределенных вычислений Hadoop MapReduce и Apache Spark.

Во-первых, фундаментальное различие между рассматриваемыми платформами заключается не столько в самой идее декомпозиции задачи на этапы map и reduce, сколько в стратегии управления промежуточными данными. Классический Hadoop опирается на жесткую материализацию результатов каждого этапа на дисках распределенной файловой системы (HDFS). Данная архитектура обеспечивает высочайшую отказоустойчивость и предсказуемость профилей нагрузки на подсистему хранения, что делает ее надежным фундаментом для построения ETL-конвейеров и строго однопроходных аналитических задач [2]. Однако при решении итерационных алгоритмов, характерных для методов максимального правдоподобия или численной оптимизации функции потерь в моделях с фиксированными и случайными эффектами, эта особенность становится критическим узким местом. Многократное перечитывание данных с медленного дискового массива приводит к экспоненциальному росту времени выполнения и сетевого трафика.

Во-вторых, переход от дисково-ориентированной модели Hadoop к резидентной модели вычислений в оперативной памяти, реализованной в Apache Spark через концепцию устойчивых распределенных наборов данных (RDD) и DataFrame API, кардинально меняет экономику высокопроизводительных вычислений.

В-третьих, анализ масштабируемости продемонстрировал, что преимущество Spark усиливается нелинейно с ростом числа узлов кластера за счет минимизации дорогостоящего сетевого перемешивания на каждой итерации. Тем не менее, данное преимущество не является абсолютным. При обработке массивов данных [1], объем которых существенно превышает суммарную оперативную память доступных вычислительных узлов (сценарий Big Data out-of-core), эффективность Spark может снижаться из-за частой подкачки страниц памяти на диск. В таких условиях гибридные стратегии персистентности (уровень MEMORY_AND_DISK) или разделение конвейера обработки становятся необходимыми мерами. Проведенный формальный вывод порога эффективности показал, что окончательный выбор платформы должен базироваться на сопоставлении удельной стоимости операций ввода-вывода диска, пропускной способности сети и накладных расходов планировщика ресурсов (YARN против собственного менеджера Spark).

В-четвертых, существенным фактором практической реализации является зрелость экосистемы прикладных библиотек. Экосистема Hadoop исторически ориентирована на низкоуровневое управление потоками данных, что требует от исследователя написания трудоемких функций map и reduce на языке Java, увеличивая риск ошибок при реализации специфических эконометрических формул. Напротив, высокоуровневые API Spark (Scala, Python/PySpark, R/SparkR) [8] в сочетании со встроенным оптимизатором Catalyst и библиотекой MLib предоставляют готовую среду для прототипирования и промышленного развертывания сложных моделей. Возможность декларативного описания агрегатов через Spark SQL автоматически устраняет избыточные операции shuffle и использует широковещательные соединения для малых таблиц измерений, что значительно снижает трудозатраты на разработку без потери производительности исполняемого кода.

Таким образом, результаты исследования подтверждают тезис о том, что Hadoop и Spark являются не конкурирующими альтернативами, а взаимодополняющими компонентами современной архитектуры обработки больших данных. Будущее моделирования лежит в плоскости их синергетического использования, где надежность дисковой модели первого сочетается с интерактивностью и скоростью резидентной модели второго. Перспективы дальнейших исследований видятся в адаптации рассмотренных подходов к гетерогенным вычислительным средам, включающим ускорители (GPU) для тензорных вычислений, а также в разработке специализированных фреймворков автоматической генерации оптимального DAG-плана для заданного класса структурных эконометрических уравнений.

Литература

1. Губин Е.И. Методика подготовки больших данных для прогнозного анализа // Наука и бизнес: Пути развития. 2020. № 3 (105). С. 27–31. URL: globaljournals.ru/assets/files/journals/science-and-business/105/sb-3(105) -2020-main.pdf?ysclid=mttpihd56g469647141
2. Битченков Д.И., Злобина М.С., Ермаков С.Г. Ключевые проблемы внедрения ETL/ ELT-процессов и методы их преодоления // Интеллектуальные технологии на транспорте. 2025. № 3 (43). С. 41-48. URL: https://cyberleninka.ru/article/n/klyuchevye-problemy-vnedreniya-etl-elt-protsessov-i-metody-ih-preodoleniya/viewer.
3. Документация Apache Software Foundation. Hadoop MapReduce Tutorial (версия 3.4). – URL: https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html?ref=blog.min.io (дата обращения: 06.09.2026).
4. Массарон Л., Банахевич К. Книга Kaggle. Машинное обучение и анализ данных /Л. Массарон, К. Банахевич: Пер. с англ. — СПб.: БХВ-Петербург, 2024. — 448 с.: URL: https://djvu.online/file/uqY9iOQFZHSen?ysclid=mttq7uf0zs858272278.
5. Марков А.И. «Big Data: технологии обработки больших данных». – Москва: Наука, 2021. URL: https://cyberleninka.ru/article/n/kompyuternye-tehnologii-dlya-obrabotki-bolshih-dannyh/viewer.
6. Томас Эртл. «Большие данные и облачные вычисления». – СПб: Питер, 2020. URL: https://naukamirowozreniya.ru/public/202504/application/1744921819465506841/kompyuternye-tehnologii-dlya-obrabotki-bolshih-dannyh.pdf
7. Baltagi B.H. Econometric Analysis of Panel Data. — 7th ed. — Springer Texts in Business and Economics, 2021. DOI: 10.1007/978-3-030-53953-5. URL: https://www.sci-hub.ru/10.1007/978-3-030-53953-5?ysclid=mttppxfg30944067851
8. Karau H., Konwinski A., Wendell P., Zaharia M. Learning Spark: Lightning-Fast Big Data Analytics. — 3rd ed. — O'Reilly Media, 2023. — 342 p. URL: https://cs.famaf.unc.edu.ar/~damian/tmp/bib/Learning_Spark_Lightning-Fast_Big_Data_Analysis.pdf

Comparative Analysis of MapReduce Algorithms (Hadoop/Spark) for Panel Data Estimation

Kotin I.S.
Saint-Petersburg State University of Economics

JEL classification: C01, C02, C1, C4, C5, C6, C8

This paper presents a comparative analysis of the two leading implementations of the MapReduce distributed computing paradigm—Apache Hadoop and Apache Spark—applied to econometric estimation tasks on massive panel datasets. The evolution of information systems has led to datasets ranging in size from tens to hundreds of gigabytes, the processing of which requires horizontal scaling. The fundamental difference between the platforms lies in their intermediate data management strategies: classic Hadoop relies on rigid materialization of the results of each stage to disk (HDFS), which ensures high fault tolerance for single-pass ETL pipelines but becomes a critical bottleneck in iterative algorithms due to repeated data rereading. In contrast, Apache Spark implements an in-memory computing model via the Resilient Distributed Dataset (RDD) concept and the DataFrame API. This allows data to be cached between iterations, minimizing costly network shuffles and disk I/O, which fundamentally transforms the economics of high-performance computing for maximum likelihood methods and numerical optimization. A significant factor in practical implementation is ecosystem maturity. Spark's high-level APIs (Spark SQL, the Catalyst optimizer, and the MLlib library) considerably reduce development effort for complex econometric models compared to low-level map/reduce function programming in Java for Hadoop. The findings support the thesis of platform complementarity: Hadoop's disk-based reliability pairs with Spark's interactivity. Future research directions include adapting these approaches to heterogeneous environments with GPU accelerators and developing frameworks for automatic optimal execution plan generation.

Keywords: MapReduce, Hadoop, Spark, DataFrame, SQL, Big Data.

References

1. Gubin E.I. Methodology of big data preparation for predictive analysis // Science and business: Ways of development. 2020. No. 3 (105). pp. 27-31. URL: globaljournals.ru/assets/files/journals/science-and-business/105/sb-3 (105) -2020-main.pdf?ysclid=mttpihd56g469647141
2. Bitchenkov D.I., Zlobina M.S., Ermakov S.G. Key problems of implementing ETL/ELT technologies and methods of their application // Intelligent technologies. technology in transport. 2025. Number 3 (43). pp. 41-48. URL: https://cyberleninka.ru/article/n/klyuchevye-problemy-vnedreniya-etl-elt-protsessov-i-metody-ih-preodoleniya/viewer.
3. Introduction to the Apache Software Foundation. Hadoop MapReduce Manual (version 3.4). – URL: https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html?ref=blog.min.io (date of reference: 09/06/2026).
4. Masaron L., Banashevich K. A book to read. Machine learning and data analysis / L. Massaron, K. Banakhevich: Translated from English — St. Petersburg: BV-Peter, 2024. — 448 p.: URL: https://djvu.online/file/uqY9iOQFZHSen?ysclid=mttq7uf0zs858272278.
5. Markov A.I. "Big data: big data processing technologies". – Moscow: Nauka Publ., 2021. URL: https://cyberleninka.ru/article/n/kompyuternye-tehnologii-dlya-obrabotki-bolshih-dannyh/viewer.
6. Thomas Ertl. "Big Data and cloud computing." – St. Petersburg: Peter, 2020. URL: https://naukamirowozreniya.ru/public/202504/application/1744921819465506841/kompyuternye-tehnologii-dlya-obrabotki-bolshih-dannyh.pdf
7. Baltagi B.B. Astronomical data analysis. — 7th ed. - Springer Texts in Business and Economics, 2021. DOI: 10.1007/978-3-030-53953-5. URL: https://www.sci-hub.ru/10.1007/978-3-030-53953-5? ysclid=mttppxfg30944067851
8. Karau H., Konunsky A., Wendell P., Zachariah M. The Spark of Learning: an international analysis of big books. — 3rd ed. — O'Reilly Media, 2023. - 342 p. URL: https://cs.famaf.unc.edu.ar /~damian/tmp/bib/Learning_Spark_Lightning-Fast Big Data Analysis.pdf