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

DOI: https://doi.org/10.60797/IRJ.2025.162.81

АРХИТЕКТУРА СИСТЕМЫ СБОРА И ХРАНЕНИЯ МЕТРИК ИСПОЛЬЗОВАНИЯ РЕСУРСОВ SPARK-ПРИЛОЖЕНИЙ В КЛАСТЕРНЫХ СИСТЕМАХ ОБРАБОТКИ БОЛЬШИХ ДАННЫХ

Научная статья

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

* Корреспондирующий автор (diman-rraazz[at]mail.ru)

Аннотация

Объектом исследования является процесс сбора и анализа вычислительных метрик Apache Spark-приложений в условиях обработки больших объемов данных на кластерах. Целью работы является разработка и обоснование архитектуры системы сбора, передачи и хранения метрик использования вычислительных ресурсов Spark-приложений для последующего анализа и поддержки оптимизации их конфигураций. В работе проанализированы существующие подходы к мониторингу Spark-приложений и сформулированы требования к системе сбора метрик в условиях промышленного кластера. Сформирован минимально достаточный набор метрик, отражающих ключевые аспекты поведения приложений и загрузки ресурсов. Предложено архитектурное решение, включающее использование GraphiteSink, Logstash, Kafka и ClickHouse, позволяющее реализовать интервальный сбор метрик с шагом 10 секунд. Проведена опытная эксплуатация системы в условиях реального вычислительного кластера, подтвердившая её надежность, масштабируемость и применимость для промышленных задач. Представленная система может служить основой для оптимизации распределенных вычислений.

Ключевые слова: Apache Spark, утилизация ресурсов, большие данные, система мониторинга, метрики.

ARCHITECTURE OF A SYSTEM FOR COLLECTING AND STORING METRICS ON THE RESOURCE USAGE OF SPARK APPLICATIONS IN CLUSTERED BIG DATA PROCESSING SYSTEMS

Research article

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

* Corresponding author (diman-rraazz[at]mail.ru)

Abstract

The object of the study is the process of collecting and analysing computational metrics of Apache Spark applications in conditions of processing large volumes of data on clusters. The aim of the work is to develop and substantiate the architecture of a system for collecting, transmitting and storing metrics on the use of Spark application computing resources for subsequent analysis and support for optimising their configurations. The work analyses existing approaches to monitoring Spark applications and formulates requirements for a metric collection system in an industrial cluster environment. A minimally sufficient set of metrics has been formed, reflecting key aspects of application behaviour and resource utilisation. An architectural solution has been proposed that includes the use of GraphiteSink, Logstash, Kafka, and ClickHouse, allowing for interval metric collection every 10 seconds. The system was tested in a real computing cluster, confirming its reliability, scalability, and applicability to industrial tasks. The presented system can serve as a basis for optimising distributed computing.

Keywords: Apache Spark, resource utilisation, big data, monitoring system, metrics.

Введение

С ростом объемов данных в различных областях по данным IDC на 2025 [1], их обработка становится все более важной частью современных информационных систем. По оценкам IDC, общий объем данных в мире достигнет 175 зеттабайт к концу 2025 году, по сравнению с 33 ЗБ в 2018 году [2]. Сокращение времени обработки, минимизация накладных расходов может быть достигнута путем эффективного использования вычислительных ресурсов приложений, работающих в рамках кластеров. Apache Spark — это унифицированный вычислительный движок и набор библиотек для массовой параллельной обработки данных на компьютерных кластерах. Spark поддерживает несколько широко используемых языков программирования (Python, Java, Scala и R), включает библиотеки для различных задач, обеспечивает выполнение SQL запросов, потоковою обработку данных, а также используется в рамках машинного обучения. Несмотря на развитые механизмы параллельных вычислений, обеспечение эффективного использования ресурсов остается одной из наиболее распространенных проблем в Spark-приложениях, что рассмотрено в статье [3]. Неправильная настройка используемых ресурсов приложениями может привести к значительной деградации производительности, увеличению времени выполнения как самого приложения, так и других

приложений, работающие на кластере. Совокупное влияние перечисленных факторов способствует увеличению инфраструктурных затрат и снижению общей эффективности эксплуатации вычислительных ресурсов.

Для решения задачи оптимизации используемых ресурсов приложениями Spark на кластере требуется получать достоверную информацию о нагрузке на систему и о количестве потребляемых ресурсов. Формирование устойчивого и масштабируемого подхода к сбору и структуризации метрик представляет собой ключевую задачу при работе с большими объемами данных.

Цель данной работы состоит в разработке и обосновании архитектуры системы сбора, передачи и хранения вычислительных метрик Apache Spark-приложений в кластерной среде для последующего анализа и оптимизации использования ресурсов.

Для достижения поставленной цели требуется решить следующие задачи:

- определить минимально необходимый набор метрик, отражающих поведение Spark-приложений и использование ресурсов;
- выполнить анализ существующих методов и инструментов мониторинга Spark-приложений с точки зрения применимости в условиях высоконагруженных кластеров;
- разработать архитектуру системы сбора метрик;
- провести промышленную проверку архитектуры, включая анализ зависимости между частотой сбора метрик и точностью анализа, выбрав компонентов системы (источник метрик, протокол передачи, брокер сообщений, СУБД для хранения) с учетом ограничений на нагрузку и масштаб кластера.

Объектом исследования является процесс мониторинга вычислительных характеристик Spark-приложений в распределенных средах. Предметом исследования выступают методы сбора, передачи и хранения вычислительных метаданных с целью их последующего использования для оптимизации выполнения приложений и управления ресурсами.

Методы и принципы исследования

2.1. Определение необходимого набора метрик

Для полноценного анализа методов сбора метрик, необходимо определить их минимально достаточный набор, обеспечивающий репрезентативную оценку поведения Spark-приложений и использования ресурсов вычислительного кластера.

Apache Spark представляет собой распределенную платформу обработки данных, основанную на модели RDD. Все вычисления над данными производятся на процессоре, с хранением промежуточных результатов вычислений в оперативной памяти. Ключевой особенностью Spark является его ленивая модель вычислений: операции над данными не выполняются в момент вызова. На этапе планирования Spark разбивает всю цепочку вычислений на стадии, каждая из которых состоит из параллельно исполняемых задач. Задачи, в свою очередь, могут быть запущены на различных исполнителях, которые расположены на разных узлах кластера. Несмотря на масштабируемость, данная модель порождает ряд существенных проблем:

Наличие повторных трансформаций, кэширование больших объемов данных, неоптимальные стратегии соединений, неверные параметры shuffle — все это приводит к неоптимальному расходу ресурсов и замедлению работы приложений [4], [5];

Неправильно выбранные параметры приложений приводят либо к переутилизации отдельных исполнителей, что влечет за собой снижение скорости обработки данных [6], либо к простою больших объемов ресурсов, что приводит к неоправданному увеличению эксплуатационных затрат на инфраструктуру;

Избыточная нагрузка на систему хранения данных [5].

Исходя из представленных проблем, необходимо собирать информацию об использовании оперативной памяти, утилизации NameNode кластера, а также о spill данных в память и на диск. Согласно статье [7] в Spark 3.0 были существенно расширены средства мониторинга памяти, что дает возможность наиболее полного и детального анализа утилизации ресурсов приложениями: введены новые метрики, позволяющие отслеживать использование heap и off-heap памяти на уровне исполнителей и задач, а также события disk spill, в случае нехватки оперативной памяти, что критично для оптимизации ресурсоемких приложений.

Согласно официальной документации Spark [8], при работе приложений Spark автоматически собирает и публикует более 100 различных метрик, охватывающих уровни задач (task), стадий (stage), исполнителей (executor), драйвера (driver) и JVM. Для решения обозначенных в работе проблем были выделены и классифицированы пять ключевых категорий метрик, охватывающих наиболее критичные аспекты производительности Spark-приложений.

В Spark 3.0 добавлены точные метрики для мониторинга heap и off-heap памяти (memory.OnHeapStorageMemory, memory.OffHeapStorageMemory и др.), а также динамики использования памяти в JVM (jvm.heap.used, jvm.non-heap.used). Эти показатели позволяют своевременно выявлять утечки памяти, избыточное кэширование и неэффективные конфигурации выделяемой оперативной памяти.

Метрики memoryBytesSpilled и diskBytesSpilled фиксируют моменты, когда данные не помещаются в память и вынужденно сбрасываются на диск (disk spill). Это позволяет выявлять узкие места в конфигурации памяти при выполнении shuffle, join и sort операций и избегать дорогостоящих I/O-операций.

Метрики filesystem.hdfs.read_bytes, filesystem.hdfs.read_ops, filesystem.hdfs.write_bytes, filesystem.hdfs.write_ops позволяют оценить объем данных, проходящих через задачи и стадии. Эти данные используются для оценки влияния на файловое хранилище. Позволяют выявлять чрезмерную нагрузку на файловую систему при работе с большими объемами данных.

Метрики shuffleReadMetrics и shuffleWriteMetrics фиксируют объемы и распределение данных между узлами, включая долю локальных и удаленных блоков. Они критически важны для анализа распределения нагрузки и выбора оптимальной стратегии операции join.

Метрики task.executorRunTime, task.executorCpuTime, task.jvmGCTime, task.duration, task.peakExecutionMemory поз воляют определить узкие места на уровне отдельных задач, исправить неравномерная нагрузка между исполнителями, ускорить долгие задачи, снизить перерасход CPU и время на ожидание ресурсов.

В рамках данного исследования необходимо определить в каждой группе минимальный набор базовых метрик, достаточный для:

- оценки загрузки ресурса;
- диагностики типовых проблем, таких как spill, data skew, перегруз памяти/GC [4].

Подобный подход соответствует практике работ по подстройке параметров для повышения быстродействия процессов, где для построения моделей производительности и автонастройки конфигураций выбирается ограниченный набор наиболее информативных параметров, покрывающих ресурсы CPU, памяти и диска [9].

В результате из исходного множества метрик был сформирован минимально достаточный набор из 35 показателей, список которых приведён в табл. 1. Данный набор обеспечивает покрытие всех выделенных ресурсов и стадий выполнения приложений и служит основой для дальнейшего анализа и визуализации производительности Spark-приложений. Расширение набора за счёт дополнительных счётчиков, дублирующих информацию по смыслу, было признано нецелесообразным с точки зрения объёма хранимых данных и сложности последующей обработки.

Таблица 1 - Ключевые метрики мониторинга нагрузки Spark-приложения

DOI: https://doi.org/10.60797/IRJ.2025.162.81.1

Категория	Метрика	Описание
Использование памяти	jvm.heap.used	Объем используемой heap-памяти JVM
	jvm.heap.committed	Объем выделенной heap-памяти JVM
	jvm.heap.max	Максимально доступный объем heap-памяти JVM
	jvm.non-heap.used	Объем используемой non-heap памяти JVM
	jvm.non-heap.committed	Объем выделенной non-heap памяти JVM
	jvm.non-heap.max	Максимально доступный объем non-heap памяти JVM
	ExecutorMetrics.OnHeapStorageMemory	Используемая on-heap память для хранения данных
	ExecutorMetrics.OffHeapStorageMemory	Используемая off-heap память для хранения данных
	ExecutorMetrics.OnHeapExecutionMemory	Общий объем on-heap памяти для хранения
	ExecutorMetrics.OffHeapExecutionMemory	Общий объем off-heap памяти для хранения
Spill	executor.memoryBytesSpilled.count	Объем данных, сброшенных из памяти на диск на уровне executor
	executor.diskBytesSpilled.count	Объем данных, сброшенных на диск на уровне executor
	task.memoryBytesSpilled.count	Объем данных, сброшенных из памяти на диск на уровне задачи
	task.diskBytesSpilled.count	Объем данных, сброшенных на диск на уровне задачи
Ввод/вывод	executor.filesystem.hdfs.largeRead_ops	Количество больших операций чтения данных executor из hdfs
	executor.filesystem.hdfs.read_bytes	Объем данных, прочитанных executor из hdfs
	executor.filesystem.hdfs.read_ops	Количество операций чтения данных executor из hdfs
	executor.filesystem.hdfs.write_bytes	Объем данных, записанных executor из hdfs
	executor.filesystem.hdfs.write_ops	Количество операций записи данных executor из hdfs

Категория	Метрика	Описание
	task.outputMetrics.bytesWritten	Объем данных, записанных задачей
Задачи	task.duration	Продолжительность выполнения задачи
	task.jvmGCTime	Время на сборку мусора во время выполнения задачи
	task.resultSize	Размер результата задачи
	task.executorRunTime	Время выполнения задачи на executor
	task.executorCpuTime	Время использования CPU задачей
	task.peakExecutionMemory	Пиковое использование памяти задачей
Shuffle-чтение	shuffleReadMetrics.totalBlocksFetched	Количество блоков, полученных в shuffle-чтении
	shuffleReadMetrics.localBlocksFetched	Локальные блоки, полученные в shuffle-чтении
	shuffleReadMetrics.remoteBlocksFetched	Удаленные блоки, полученные в shuffle-чтении
	shuffleReadMetrics.totalBytesRead	Общий объем байт, прочитанных в shuffle-чтении
	shuffleReadMetrics.localBytesRead	Объем байт, прочитанных локально в shuffle-чтении
	shuffleReadMetrics.remoteBytesRead	Объем байт, прочитанных удаленно в shuffle-чтении
	shuffleWriteMetrics.bytesWritten	Объем байт, записанных в shuffle-записи
	shuffleWriteMetrics.recordsWritten	Количество записей, записанных в shuffle-записи
	shuffleWriteMetrics.writeTime	Время записи в shuffle (в наносекундах)

2.2. Анализ существующих решений

Современные исследования в области мониторинга Apache Spark-приложений охватывают широкий спектр решений — от базовых инструментов визуализации (Spark UI, Ganglia) до комплексных систем потоковой аналитики на базе Prometheus, Kafka и Grafana [10], [11]. Вместе с тем, подобные решения зачастую не адаптированы к промышленной эксплуатации в высоконагруженных кластерах, что отмечается в работе [12]. Кроме того, в литературе слабо представлены вопросы агрегации и оптимизации хранения метрик в условиях высокой частоты сбора. По данным «ieeexplore» на 2025 год, по ключевым словам, «Spark», «Memory» и «Optimization» было найдено 5 релевантных статей выпущенных позднее 2020 года.

Для построения архитектуры введем требования к высоконагруженной системе:

- Масштабируемость. Возможность горизонтального расширения как всей системы, так и отдельных ее компонент.
- Доступность. Обеспечивать непрерывность работы системы с минимальным временем простоя.
- Минимальная нагрузка. Иметь минимальное влияние на вычислительные процессы кластера, включая в себя необходимость поддержания сложной внешней инфраструктуры.

В данной работе сделан акцент на построении отказоустойчивой архитектуры с минимальной нагрузкой на вычислительные ресурсы кластера, что позволяет восполнить существующий пробел между теоретическими разработками и практическими потребностями промышленной эксплуатации Spark-приложений.

В рамках дальнейшего анализа предлагаю следующую систему классов источников получения метрик от Spark-приложений:

- Spark UI;
- Listener;
- Встроенные в Spark коннекторы.

Первым классом источников является Spark UI, а также Spark Rest API. Они представляют агрегированные сведения, включая длительность стадий, объемы обрабатываемых данных и уровень загруженности ресурсов. Данный подход предоставляет обобщенную картину без разбивки по интервалам времени, что является минусом при анализе утилизации ресурсов приложениями. В свою очередь, вычисления в рамках Spark-приложений характеризуются тем, что могут динамически изменять используемые ресурсы в пределах выделенного пользователем максимального их объема.

Следующий класс источников основывается на отдельном сопроцессе Listener в рамках основного Spark-приложения. Пользователь может как реализовать свой сопроцесс позволяющий проводить операции по сбору и отправке метрик в систему потребитель, так и в самом Spark уже есть встроенный EventLoggingListener, сохраняющий всю информацию о событиях выполнения приложения: задания, стадии, задачи, операции ввода-вывода, использование памяти и другие параметры. Такой подход позволяет анализировать использование различных операций и их влияние на ресурсоемкость выполнения [3]. Обработка Event Log в таком случае выполняется внешним инструментом. Существуют готовые решения, которые позволяют получать метрики, считаемые автоматически Spark-приложениями. Например, использование PrometheusServletSink в таких решениях описано в [11], [13]. Он занимается обработкой Event Log сообщений после завершения работы приложения. Представляет собой готовое решение, которое выполняет собственный анализ и выводит результат пользователю в виде UI представления. Но требует отдельной инфраструктуры, плохо ложится на существующий on-prem стек, ориентирован на Kubernetes-окружение [11], [13].

Еще одним инструментом сбора метрик являются реализованные в Apache Spark коннекторы для отправки. Они заменяют требующие собственной реализации Listener, а также позволяют реализовать анализ приложения уже во время его работы, посредством выгрузки уже посчитанных Spark`ом метрик в систему потребитель. Настройка интеграции сводится к заданию двух конфигурационных параметров в Spark-приложении и не предполагает необходимость развертывания дополнительной инфраструктуры, обладающей высокой степенью сложности, в отличие от готовых систем мониторинга на основе Event Log. Сводный обзор методов оформлен в виде таблицы 2.

Таблица 2 - Сводное сравнение подходов сбора метрик Spark-приложений

DOI: https://doi.org/10.60797/IRJ.2025.162.81.2

Подход	Уровень детализации	Нагрузка на систему	Гибкость настройки	Требования к инфраструктуре
Spark UI / Spark Rest API	Низкий	Низкая	Низкая	Инфраструктура хранения, сервис сбора метрик
Listener	Высокий	Средняя	Высокая	Инфраструктура хранения, требуется реализация слушателя и подключение ко всем приложениям
Встроенный Connector	Высокий	Низкая	Средняя	Требуется инфраструктура для приема и хранения

2.3. Методика и архитектура предлагаемого подхода

Выбор конкретного способа сбора и отправки метрик зависит от объема метрик, количества узлов, ядер на каждом узле, объемов оперативной памяти каждого узла кластера, а также от количества работающих Spark-приложений. Введем характеристики кластера:

- 200 узлов, с характеристиками 100 Гб оперативной памяти и 20 ядер на каждом;
- ежедневная работа минимум 3000 Spark-приложений.

Совокупный объем данных определяется частотой сбора метрик и размерностью результирующего набора параметров. Наиболее сбалансированным решением представляется интервальных сбор метрик с приложения, ввиду отсутствия реактивных вычислений и для снижения нагрузки как на само приложение, так и на систему приемщик. Эмпирические исследования показывают, что оптимизация частоты сбора напрямую влияет на нагрузку системы хранения [12]. Возможность интервального сбора реализована как во встроенных коннекторах Spark, так и при использовании пользовательских Listener-компонентов. Наиболее оптимальным в базовом ключе является готовый коннектор Spark, так как не требует никаких доработок со стороны приложений и кластера. Использование Listener предполагает разработку отдельного механизма сбора и маршрутизации метрик, а также является инфраструктурно ограниченным — его требуется подключить ко всем приложениям кластера. В рамках предлагаемого решения использовался метод интервального сбора метрик с шагом 10 секунд посредством GraphiteSink. Интервал в 10 секунд между отправками, был выбран экспериментальным путем, результаты чего представлены на рисунке 1. Он позволяет соблюсти баланс между упущенными релевантными метриками и объемом отправляемых метрик из Spark-приложений, который растет линейно в зависимости от частоты.

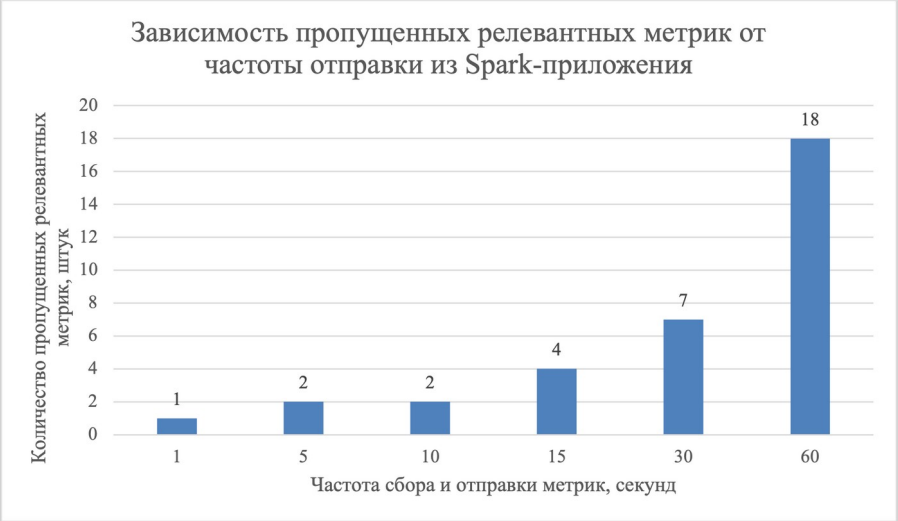

Рисунок 1 - Зависимость пропущенных релевантных метрик от частоты отправки из Spark-приложения
DOI: https://doi.org/10.60797/IRJ.2025.162.81.3

В базовом сценарии взаимодействия с данным коннектором требуется развернутый Graphite, что является слишком дорогим в эксплуатации за счет большого объема дискового пространства необходимого для Graphite в рамках кластера с большими объемами данных. Поэтому требуется потребитель, способный выдерживать большой объем хранимых логов в рамках кластера. Из вариантов можно рассмотреть хранение в HDFS и Clickhouse. HDFS недостаточно эффективен в контексте рассматриваемой архитектуры, так как под каждую метрику будет требовать создания отдельного файла, что может вызвать деградацию производительности или сбои в функционировании кластера. Либо же для использования потребуется создание дополнительного процесса, который будет сжимать несколько отдельных файлов и сохранять их в единый файл большего объема. Данный подход сопряжён с рядом архитектурных ограничений, снижающих его применимость в производственных условиях:

- на промежуточном этапе неизбежно образуется значительное число мелких файлов;
- дополнительные вычислительные ресурсы на работу потока;
- недоступность данных на время перезаписи.

Указанные ограничения существенно снижают применимость HDFS в качестве системы хранения метрик в рассматриваемом сценарии.

Clickhouse представляет собой более предпочтительное решение с точки зрения производительности и устойчивости хранения. Примеры успешного применения ClickHouse для потоковой аналитики приведены в [14], [15]. Учитывая значительное количество метрик и характерную стабильность их значений на протяжении фиксированных временных интервалов, требуется реализация механизма агрегации, позволяющего эффективно минимизировать объём сохраняемой информации, в результате чего повторяющиеся значения агрегируются в единственное представление в пределах заданного окна, на котором они имеют эффект. Для реализации агрегации применяется движок AggregatingMergeTree, обеспечивающий свёртку всех строк с одинаковыми ключами в единую запись, представляющую агрегированное состояние по заданным функциям в пределах одного сегмента данных. Такой подход способствует сокращению объема хранимых данных и снижению нагрузки на систему визуализации.

Последним элементом, который необходимо выбрать — это система передачи данных от Spark-приложения к системе хранения данных. В GraphiteSink отсутствует возможность прямой интеграции с ClickHouse по стандартным интерфейсам передачи данных. Коннектор поддерживает передачу метрик по протоколу UDP в систему, которая имеет возможность принимать такого вида сообщения. Наиболее используемыми такими инструментами являются Logstash [16] и Ni-Fi. Ni-Fi требует больших вычислительных ресурсов и является кластерным решением, характеризующимся высокой сложностью развертывания и сопровождения. Logstash представляет собой легковесное решение для маршрутизации и трансформации потоков данных между компонентами распределенной системы. В контексте рассматриваемой архитектуры данный инструмент обеспечивает оптимальный баланс между производительностью и гибкостью настройки. При этом Logstash требует наличия приёмника данных. Возможна прямая запись в ClickHouse, однако такой подход не обеспечивает необходимого уровня отказоустойчивости из-за потенциальной недоступности ClickHouse в периоды обслуживания или при пиковых нагрузках. Потеря метрик в подобной конфигурации приведёт к нарушению целостности исторических данных, что может привести к снижению достоверности и полноты последующего анализа метрик. Необходимо использовать брокер для минимизации недоступности, а также для дальнейшей балансировки нагрузки при записи в Clickhouse. Для задач в области обработки больших данных целесообразно применение Kafka, зарекомендовавшей себя как промышленный стандарт брокеров сообщений.

В результате сформирована следующая архитектура системы:

- Spark-приложение — в режиме выполнения вычислений считает метрики использования ресурсов, а также метрик о сути вычислений;
- GraphiteSink — встроенный подпроцесс в Spark, позволяющий экспортировать метрики во внешнюю систему, в данном случае по протоколу UDP;

- Logstash — инструмент для маршрутизации и преобразования сообщений между источниками и приемниками данных;
- Kafka — брокер сообщений, для балансировки нагрузки и минимизации риска неуспешности вставки данных;
- AggregatingMergeTree Clickhouse – база данных для хранения метрик, для дальнейшего анализа и визуализации.

Взаимодействие в рамках архитектуры системы представлено на рисунке 2.

Рисунок 2 - Архитектура системы сбора и хранения метрик
DOI: https://doi.org/10.60797/IRJ.2025.162.81.4

Оценка эффективности и практическая применимость

В целях верификации работоспособности и оценки применимости разработанной архитектуры была проведена её опытная эксплуатация в условиях реального кластера, включающего 200 вычислительных узлов, каждый из которых обладает 100 Гб оперативной памяти и 20 вычислительными ядрами. Система использовалась в течение рабочего дня при запуске более 3000 Spark-приложений, функционирующих в рамках типичной загрузки производственной среды.

Сбор метрик осуществлялся с использованием встроенного коннектора GraphiteSink с интервалом в 10 секунд. Передача данных происходила через Logstash с маршрутизацией через Kafka в ClickHouse, где происходила агрегация и долговременное хранение информации с использованием движка AggregatingMergeTree.

Результаты эксперимента были представлены в виде интерактивных дашбордов, включающих ключевые метрики:

- Executor HDFS reads / writes — отображают динамику операций чтения и записи с HDFS в разрезе по исполнителям, отражая фрагментированную и конкурентную нагрузку на систему хранения, представлены на рисунке 3;
- HDFS Read Rate — демонстрирует пиковые значения пропускной способности при интенсивной фазе чтения, что позволяет выявлять узкие места в I/O, также представлены на рисунке 3;
- Executor Heap Usage / Driver Heap Usage — позволяют анализировать использование кучи в динамике, как на уровне отдельных исполнителей, так и на уровне управляющего драйвера и представлены на рисунке 4.

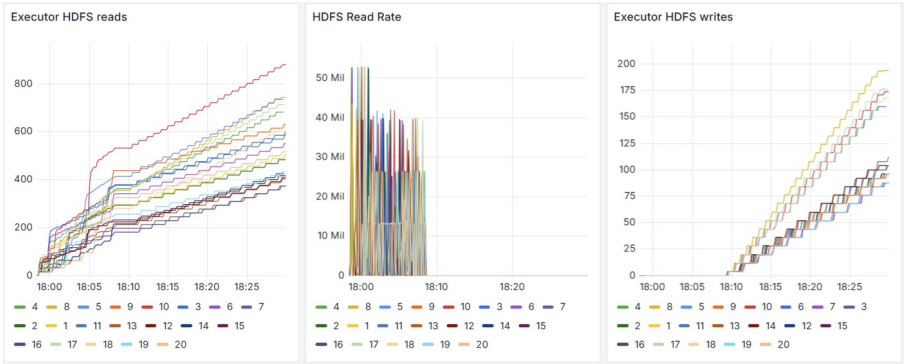

Рисунок 3 - Визуализация метрик input-output операций случайного приложения
DOI: https://doi.org/10.60797/IRJ.2025.162.81.5

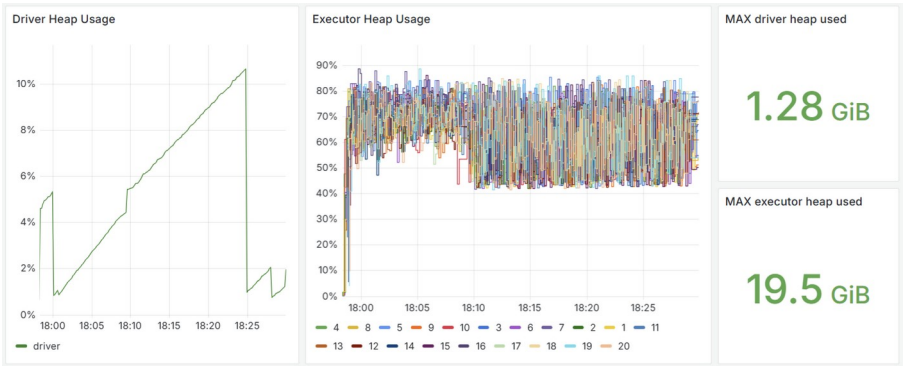

Рисунок 4 - Визуализация метрик утилизации оперативной памяти случайного приложения
DOI: https://doi.org/10.60797/IRJ.2025.162.81.6

Представленные графики демонстрируют корректную работу системы мониторинга: данные поступают с заданной частотой, покрывают ключевые аспекты поведения приложений и инфраструктуры, а также отображаются в удобной для анализа форме. Максимальные зафиксированные значения использования памяти (12.3 GiB для исполнителей и 1.20 GiB для драйвера) подтверждают возможность мониторинга ресурсоемких задач.

Практическая применимость разработанного решения подтверждается следующими наблюдениями:

- стабильная работа при высокой интенсивности поступления метрик (свыше 3000 приложений в сутки);
- снижение числа утраченных метрик за счет использования промежуточного брокера Kafka;
- повышению эффективности визуального анализа благодаря предварительной агрегации данных в ClickHouse и интеграции с системой визуализации.

Таким образом, разработанная архитектура демонстрирует высокую степень масштабируемости, устойчивость к нагрузке и пригодность для внедрения в промышленные системы мониторинга вычислительных процессов. Полученные результаты также создают предпосылки для построения надстройки в виде рекомендательной системы по оптимизации параметров Spark-приложений на основе анализа собранных вычислительных метаданных.

Заключение

В данной работе рассмотрена проблема оптимизации использования вычислительных ресурсов в Apache Spark-приложениях при обработке больших объемов данных в условиях кластерных вычислений. На основе анализа существующих решений в области мониторинга и анализа вычислительных метрик предложена интегрированная архитектура сбора, передачи и хранения метаинформации о вычислениях, направленная на повышение эффективности эксплуатации кластерных ресурсов.

Научная новизна исследования заключается в следующем:

- обоснован выбор метрик, отражающих ключевые характеристики ресурсопотребления и поведенческой логики Spark-приложений, включая информацию о DAG, spill, утилизацию памяти и узлов кластера;
- предложена архитектура отказоустойчивой системы сбора метрик с минимальной нагрузкой на вычислительные ресурсы кластера с использованием GraphiteSink и Logstash с маршрутизацией через Kafka в ClickHouse, что обеспечивает надежность, отказоустойчивость и масштабируемость системы;
- экспериментально определен и обоснован оптимальный интервал сбора метрик (10 секунд), обеспечивающий баланс между точностью наблюдений и снижением нагрузки на инфраструктуру хранения.

Реализация предложенной архитектуры позволяет сформировать устойчивую систему мониторинга и анализа, пригодную для интеграции в среду промышленных кластеров с высокой интенсивностью выполнения приложений. Система обеспечивает не только сбор, но и структурированное представление данных, пригодное для последующего автоматического анализа и визуализации.

В то же время работа имеет ряд ограничений, которые необходимо учитывать при дальнейшем развитии:

- в рамках исследования не проводилось формализованное сравнение производительности предложенного решения с существующими промышленными реализациями (например, Prometheus + Thanos);
- система предполагает единый протокол передачи метрик (UDP), что может потребовать дополнительных мер по обеспечению надежности при масштабировании;
- архитектура ориентирована на статический набор метрик и требует ручного обновления при изменении требований к мониторингу;
- агрегирование метрик на стороне ClickHouse требует тонкой настройки для конкретных сценариев анализа, особенно при построении сложных витрин.

В дальнейшем данную систему сбора и хранения метрик вычислений Spark-приложений можно будет использовать для построения систем автоматизированного анализа производительности Spark-приложений и принятия решений по их настройке.

Таким образом, представленная система может быть рассмотрена как универсальная основа для создания интеллектуальной инфраструктуры мониторинга и оптимизации вычислительных процессов в распределенных средах.

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. Большие данные (Big Data) мировой рынок // Tadviser. — 2025. — URL: https://clck.ru/3Qsj4R (дата обращения: 18.10.25)

2. IDC: EXPECT 175 ZETTABYTES OF DATA WORLDWIDE BY 2025 // Virstor. — 2025. — URL: https://virstor.co.uk/idc-data-growth-predictions/. (дата обращения: 27.09.25)

3. Nicholls B. Benchmarking Resource Usage of Underlying Datatypes of Apache Spark. / B. Nicholls, M. Adangwa, R. Estes et al. // arXiv. — 2020. — 2012.04192v1. — DOI: 10.48550/arXiv.2012.04192

4. Dessokey M. Memory Management Approaches in Apache Spark: A Review. / M. Dessokey, S. Saif, S. Salem et al. // Proceedings of the International Conference on Advanced Intelligent Systems and Informatics; — 1261. — Cham: Springer, 2020. — P. 394–403. doi: 10.1007/978-3-030-58669-0_36

5. Zhai M. Query optimization Approach with Shuffle Intermediate Cache Layer for Spark SQL. / M. Zhai, A. Song, J. Qiu et al. // 38th International Performance Computing and Communications Conference (IPCCC); — London: IEEE, 2019. — P. 1–6. doi: 10.1109/IPCCC47392.2019.8958719

6. Chen S. Optimizing Performance and Computing Resource Management of In-memory Big Data Analytics with Disaggregated Persistent Memory. / S. Chen, W. Wang, X. Wu et al. // 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID); — Larnaca: IEEE, 2019. — P. 21–30. doi: 10.1109/CCGRID.2019.00012

7. Apache Spark 3.0 Memory Monitoring Improvements // CERN. — 2020. — URL: https://db-blog.web.cern.ch/blog/luca-canali/2020-08-spark3-memory-monitoring. (дата обращения: 29.09.25)

8. Monitoring and Instrumentation // Apache Spark. — 2025. — URL: https://spark.apache.org/docs/latest/monitoring.html#executor-task-metrics. (дата обращения: 23.10.25)

9. Li Y. Towards General and Efficient Online Tuning for Spark. / Y. Li, H. Jiang, Y. Shen et al. // arXiv. — 2023. — 2309.01901v1. — DOI: 10.48550/arXiv.2309.01901

10. Castro D. Apache Spark usage and deployment models for scientific computing / D. Castro, P. Kothuri, P. Mrowczynski et al. // EPJ Web Conf. — 2019. — № 214. — 07020. — URL: https://doi.org/10.1051/epjconf/201921407020 (accessed: 02.10.25) — DOI: 10.1051/epjconf/201921407020

11. Bhavya J. Monitoring Apache Spark Metrics with Prometheus / J. Bhavya // Medium. — 2024. — URL: https://medium.com/@bhavya.joshi1901/monitoring-apache-spark-metrics-with-prometheus-a2846fe94028. (дата обращения: 07.10.25)

12. Mehdi A. Unleashing the Potential of Grafana: A Comprehensive Study on Real-Time Monitoring and Visualization / A. Mehdi, M. Bali, S. Abbas et al. // 14th International Conference on Computing Communication and Networking Technologies (ICCCNT). — Delhi: Indian Institute of Technology, 2023. — URL: https://clck.ru/3QsjDP (accessed: 07.10.25). — DOI: 10.1109/ICCCNT56998.2023.10306699

13. Molnar B. Monitoring Apache Spark with Prometheus on Kubernetes / B. Molnar // Outshift by Cisco. — 2017. — URL: https://outshift.cisco.com/blog/spark-monitoring. (дата обращения: 13.10.25)

14. Dahimene R. Real-time data for real-time analytics with Kafka and ClickHouse / R. Dahimene, D. McDiarmid // Confluent. — 2023. — URL: https://www.confluent.io/events/current/2023/real-time-real-time-data-for-real-time-analytics-with-kafka-and-clickhouse/. (дата обращения: 06.10.25)

15. Schulze R. ClickHouse - Lightning Fast Analytics for Everyone. / R. Schulze, T. Schreiber, I. Yatsishin et al. // Proceedings of the VLDB Endowment. — 2024. — № 17. — DOI: 10.14778/3685800.3685802

16. Tokar D. The IoT Applications Productivity: Data Management Model and ELK Tool Based Monitoring and Research. / D. Tokar, O. Morozova, V. Kharchenko. // IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET); — Lviv-Slavske: IEEE, 2022. — P. 162–167. doi: 10.1109/TCSET55632.2022.9766930

Список литературы на английском языке / References in English

1. Bolshie dannie (Big Data) mirovoi rinok [Big Data global market] // tadviser.ru. — 2025. — URL: https://clck.ru/3Qsj4R (accessed: 18.10.25) [in Russian]

2. IDC: EXPECT 175 ZETTABYTES OF DATA WORLDWIDE BY 2025 // Virstor. — 2025. — URL: https://virstor.co.uk/idc-data-growth-predictions/. (accessed: 27.09.25)

3. Nicholls B. Benchmarking Resource Usage of Underlying Datatypes of Apache Spark. / B. Nicholls, M. Adangwa, R. Estes et al. // arXiv. — 2020. — 2012.04192v1. — DOI: 10.48550/arXiv.2012.04192

4. Dessokey M. Memory Management Approaches in Apache Spark: A Review. / M. Dessokey, S. Saif, S. Salem et al. // Proceedings of the International Conference on Advanced Intelligent Systems and Informatics; — 1261. — Cham: Springer, 2020. — P. 394–403. doi: 10.1007/978-3-030-58669-0_36

5. Zhai M. Query optimization Approach with Shuffle Intermediate Cache Layer for Spark SQL. / M. Zhai, A. Song, J. Qiu et al. // 38th International Performance Computing and Communications Conference (IPCCC); — London: IEEE, 2019. — P. 1–6. doi: 10.1109/IPCCC47392.2019.8958719

6. Chen S. Optimizing Performance and Computing Resource Management of In-memory Big Data Analytics with Disaggregated Persistent Memory. / S. Chen, W. Wang, X. Wu et al. // 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID); — Larnaca: IEEE, 2019. — P. 21–30. doi: 10.1109/CCGRID.2019.00012

7. Apache Spark 3.0 Memory Monitoring Improvements // CERN. — 2020. — URL: https://db-blog.web.cern.ch/blog/luca-canali/2020-08-spark3-memory-monitoring. (accessed: 29.09.25)

8. Monitoring and Instrumentation // Apache Spark. — 2025. — URL: https://spark.apache.org/docs/latest/monitoring.html#executor-task-metrics. (accessed: 23.10.25)

9. Li Y. Towards General and Efficient Online Tuning for Spark. / Y. Li, H. Jiang, Y. Shen et al. // arXiv. — 2023. — 2309.01901v1. — DOI: 10.48550/arXiv.2309.01901

10. Castro D. Apache Spark usage and deployment models for scientific computing / D. Castro, P. Kothuri, P. Mrowczynski et al. // EPJ Web Conf. — 2019. — № 214. — 07020. — URL: https://doi.org/10.1051/epjconf/201921407020 (accessed: 02.10.25) — DOI: 10.1051/epjconf/201921407020

11. Bhavya J. Monitoring Apache Spark Metrics with Prometheus / J. Bhavya // Medium. — 2024. — URL: https://medium.com/@bhavya.joshi1901/monitoring-apache-spark-metrics-with-prometheus-a2846fe94028. (accessed: 07.10.25)

12. Mehdi A. Unleashing the Potential of Grafana: A Comprehensive Study on Real-Time Monitoring and Visualization / A. Mehdi, M. Bali, S. Abbas et al. // 14th International Conference on Computing Communication and Networking Technologies (ICCCNT). — Delhi: Indian Institute of Technology, 2023. — URL: https://clck.ru/3QsjDP (accessed: 07.10.25). — DOI: 10.1109/ICCCNT56998.2023.10306699

13. Molnar B. Monitoring Apache Spark with Prometheus on Kubernetes / B. Molnar // Outshift by Cisco. — 2017. — URL: https://outshift.cisco.com/blog/spark-monitoring. (accessed: 13.10.25)

14. Dahimene R. Real-time data for real-time analytics with Kafka and ClickHouse / R. Dahimene, D. McDiarmid // Confluent. — 2023. — URL: https://www.confluent.io/events/current/2023/real-time-real-time-data-for-real-time-analytics-with-kafka-and-clickhouse/. (accessed: 06.10.25)

15. Schulze R. ClickHouse - Lightning Fast Analytics for Everyone. / R. Schulze, T. Schreiber, I. Yatsishin et al. // Proceedings of the VLDB Endowment. — 2024. — № 17. — DOI: 10.14778/3685800.3685802

16. Tokar D. The IoT Applications Productivity: Data Management Model and ELK Tool Based Monitoring and Research. / D. Tokar, O. Morozova, V. Kharchenko. // IEEE 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET); — Lviv-Slavske: IEEE, 2022. — P. 162–167. doi: 10.1109/TCSET55632.2022.9766930