Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Кафедра математической
кибернетики и компьютерных наук

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ МОНИТОРИНГА ПРОИЗВОДИТЕЛЬНОСТИ ВЫСОКОНАГРУЖЕННЫХ СИСТЕМ

АВТОРЕФЕРАТ МАГИСТЕРСКОЙ РАБОТЫ

Студентки 2 курса 273 группы
направления 02.04.03 — Математическое обеспечение и администрирование информационных систем
факультета КНиИТ
Сытежевой Арины Олеговны

Научный руководитель
к. ф.-м. н., доцент _____ А. С. Иванов

Заведующий кафедрой
к. ф.-м. н., доцент _____ С. В. Миронов

Саратов 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время высоконагруженные системы используются во всех областях жизни человека, более того, количество таких систем непрерывно увеличивается.

Однако ограниченная техническая база серверов во всем мире создает серьезные проблемы эффективного использования высоконагруженных систем. Это обусловливает актуальность разработки. Для более эффективного применения таких систем необходимо использовать различные механизмы для мониторинга их производительности и анализа использования ресурсов [1].

Актуальность темы данной работы в современных условиях обусловлена тем, что именно качественный анализ, осуществляемый с использованием корректно подобранных метрик и данных, предоставляемых системой о её функционировании, способен обеспечить необходимую информацию для эффективного мониторинга процессов работы этой системы [2]. Внедрение мониторинга производительности систем позволяет выявлять узкие места и аномалии, способствуя повышению общей эффективности и надежности высоконагруженных приложений. При этом разработанные методы сбора и обработки метрик обладают высокой степенью адаптируемости.

Научная новизна работы заключается в универсальности подхода, которая не только расширяет область применения мониторинга, но и позволяет использовать единую методологию для анализа распределенных систем, сокращая затраты на разработку специализированных решений для каждого конкретного случая.

Практически значимой задачей является разработка специализированной системы для мониторинга производительности приложений высоконагруженной системы, автоматизирующей сбор, агрегацию и визуализацию ключевых метрик с последующей отправкой отчётов по электронной почте. В отличие от существующих решений, предлагаемый подход не требует постоянно запущенный процесс, что снижает нагрузку на кластер и экономит используемые ресурсы.

Цель данной работы состоит в создании приложения для мониторинга производительности и использования ресурсов высоконагруженных систем.

Ниже приведены задачи для достижения поставленной цели:

— Анализ актуальной информации по работе с высоконагруженными системами;

— Выбор технологий для реализации практической части работы;
— Разработка среды окружения;
— Разработка высоконагруженной системы;
— Выбор метрик для оценки потребления и рационального использования ресурсов;
— Разработка приложения для мониторинга;
— Интеграция приложения для мониторинга в разработанную высоконагруженную систему.

Работа выполнена на 63 страницах машинописного текста, состоит из введения, теоретической, практической частей и заключения, содержит 3 формулы, 36 рисунков, 4 приложения, список литературных источников содержит 22 наименования.

1 Исследование подходов к мониторингу систем

1.1 Инструменты и методы для реализации задачи

В процессе выполнения магистерской работы были проанализированы ключевые источники, на основе которых были определены и выбраны необходимые технологии.

Для реализации программного кода поставленной задачи был выбран высокоуровневый язык программирования Python, для передачи и обработки информации — инструмент Apache Spark, для оркестровки потоков передачи информации было принято решение использовать инструмент Apache Airflow. Хранение полученной информации осуществлялось с помощью Apache Hadoop. Кроме того, для обеспечения надежной и масштабируемой передачи данных между компонентами системы была использована Apache Kafka [3].

1.2 Разработка и проектирование архитектуры системы

Анализ надежности и нагрузки Перед созданием архитектуры системы необходимо определить требования к ее компонентам, а также произвести оценку нагрузки, которая будет генерироваться системой. Это требуется, чтобы убедиться в том, что система способна справиться с задаваемой нагрузкой. Вычислительные мощности позволили взять за основу кластер, состоящий из трех узлов, каждый из которых содержал 8 ядер ЦП и 32 Гб оперативной памяти. Таким образом, на узел есть возможность выделить максимум 7 executors-cores (так как 1 ядро выделяется для ОС) и 30–31 Гб оперативной памяти (1–2 Гб выделяется для стабильной работы ОС). В представленной работе ожидается использование 3 одновременно работающих процессов, нагруженных данными 50–100 Гб в каждом. CPU и RAM каждого исполнителя не должны превышать ресурсы узла, поэтому необходимо было провести дополнительную оценку нагрузки на систему.

Изначально было выделено по одному ядру на одного исполнителя. Тогда значения параметров spark-config получились следующие:

```
--num-executors = total-cores-in-cluster = 3 * 7 = 21
num-executors-per-node = 7
mem-per-node = --executor-memory * num-executors-per-node = 35Гб
```

Так как по общему количеству оперативной памяти на узел получилось превышение — было сокращено количество ОП на исполнителя с 5 до 4:

```
--executor-memory = 4GB
mem-per-node = --executor-memory * num-executors-per-node = 28Гб
```

Исполнитель, в основе которого находится 1 ядро, не может воспользоваться преимуществом запуска нескольких задач в одной и той же JVM. Более того, в этом случае общие и кэшированные переменные (например, широковещатели или аккумулирующие переменные), реплицируются в каждом ядре. При использовании одноядерных исполнителей также не остается достаточно памяти для процессов системных сервисов Hadoop/YARN и ApplicationManager. Поэтому такой подход в данной задаче нельзя назвать оптимальным [4].

Была рассмотрена альтернатива с одним исполнителем на машину, то есть, в таком случае на каждого исполнителя приходится 7 ядер (фактически доступно 8 ядер, так как 1 ядро выделяется для ОС) и 30–31 Гб оперативной памяти. Значения параметров spark-config при таком подходе:

```
--num-executors = total-nodes-in-cluster = 3
--executor-cores = total-cores-in-a-node = 7
--executor-memory = mem-per-node / num-executors-per-node =
30GB / 1 = 30GB
```

При использовании сразу всех ядер, имеющихся у исполнителей, пропускная способность HDFS ухудшится, и это приведет к чрезмерному количеству мусора, поэтому необходимо сбалансировать выделенные ресурсы.

- Было выделено одно ядро для ОС и системных служб Hadoop/YARN, т.е. количество доступных ядер на узел — $7 = 8 - 1$.
- Согласно вышеприведенным примерам двух крайностей, было назначено 3 ядра на каждого исполнителя для того, чтобы оставить достойную пропускную способность HDFS (так как в узле 7 ядер, соответственно, на каждый узел было выделено по 2 исполнителя): `--executor-cores = 3`, `--num-executors = 6`.
- Также один исполнитель был оставлен для ApplicationManager, то есть рабочие исполнители — `--num-executors = 5`.
- Количество оперативной памяти на узел определяется без учета ОП, выделенной для ОС: `--executor-memory = 32GB - 2GB = 30GB`.
- Память на одного исполнителя определяется через отношение общего количества памяти на узле к числу исполнителей на нем: $15 = 30/2$ Гб.

— Дополнительные расходы на использование JVM-кучи составят примерно 5-7% от 15 Гб, то есть 750 Кб. Поэтому фактический размер памяти исполнителя составил `--executor-memory = 15 - 0.75 = 14.25 Гб`.

Таким образом, наиболее оптимальная конфигурация в случае статического размещения ресурсов получена: 5 рабочих исполнителей по 3 ядра каждый с использованием 14.25 Гб памяти.

Разработка архитектуры высоконагруженной системы и системы мониторинга. Так как ранее было решено использовать инструмент Apache Airflow, то каждое приложение будет представлять из себя отдельный DAG. Кроме того, проанализировав возможный функционал фреймворка spark, было принято решение разработать систему мониторинга как отдельный процесс, которое позволяло бы при его запуске проходить по всем модулям в виде графов для сбора и обработки информации о статусах их процессов, логов и т.д.

Каждый граф, в свою очередь, должен состоять из Spark-задач, каждая из которых будет извлекать данные из источников информации, преобразовывать для дальнейшей отправки полученных данных в хранилища.

2 Реализация компонентов мониторинга

2.1 Подготовка среды окружения

Для реализации поставленной задачи требовалось дополнительно подготовить среду окружения, включающую установленный и настроенный кластер Apache Hadoop, в котором должны работать следующие инструменты Big Data: Apache Hive и Apache Spark, а также взаимодействовать между собой: Apache Airflow, Apache Kafka.

Для корректной работы кластера Hadoop потребовалось дополнительно установить JDK 11 и настроить конфигурационные файлы Hadoop: core-site.xml, hdfs-site.xml и mapred-site.xml были указаны параметры, такие как имя узла, порты и режим работы кластера [5]. После настройки и запуска служб Hadoop стал доступен веб-интерфейс ResourceManager (информация о текущих работающих задачах) по адресу: `http://namenode:8088`. Адрес History Server в кластере Hadoop (необходим для предоставления информации об отображении информации о завершенных задачах Spark) находится по адресу `http://namenode:19888`.

Следующим этапом подготовки среды была установка в кластер инструмента Apache Spark. Для этого была загружена версия Spark, совместимая с установленным Hadoop с официального сайта Apache Spark в виде файла с расширением .jar. Для обращения сервиса Spark к базе Hive необходимо было настроить взаимодействие между Spark и Hive, а также дополнительно установить и настроить Hive Metastore.

После настройки кластера Hadoop и spark была добавлена возможность использования интерфейса для выполнения запросов к данным, хранящимся в HDFS, — был настроен Apache Hive. Для работы инструмента необходимо было настроить конфигурационный файл, где были указаны параметры подключения к хранилищу мета-данных, включая URL базы данных, драйвер и учетные данные. После установки и запуска инструмента hive его интерфейс Hue доступен по адресу: `hdfs://namenode:9000/user/hive/warehouse`.

Для автоматизации и управления рабочими процессами обработки данных была проведена настройка Apache Airflow. Для запуска необходимых контейнеров airflow (postgresql, airflow webserver и airflow scheduler) был определен файл docker-compose.yml. После запуска контейнера `airflow webserver` становится доступа веб-страница по адресу `http://localhost:8080/`.

В рамках работы для построения распределенной системы был развернут Apache Kafka. Для корректной работы инструмента потребовалось определить конфигурационный файл, где были указаны уникальные идентификаторы брокеров, адреса слушателей и директории для хранения логов. Для обеспечения работы Kafka был определен файл docker-compose.yml — были запущены необходимые для Kafka контейнеры: zookeeper для управления брокерами, сервис брокера Kafka и интерфейс, доступный по адресу `http://localhost:8082/`.

2.2 Реализация компонент высоконагруженной системы

Для реализации приложения мониторинга в первую очередь необходимо определить высоконагруженную систему. Так, было разработано три приложения передачи данных, каждое из которых последовательно выполняет задачи: загрузка информации из источника, ее обработка и последующая загрузка в базу данных Hive.

Инструмент Airflow предоставляет набор встроенных операторов для использования в построении задач. Однако они не удовлетворяли существующим требованиям задачи сбора информации из источника Gismeteo, поэтому был реализован собственный оператор сбора и передачи информации — GismeteoDataLoader, со вспомогательным классом GismeteoGetData, который реализует отправку http get запроса с целью получения данных из URL-адреса. Оператор также представляет из себя Python класс, отправляющий полученные данные в топик «gismeteo» распределенного брокера сообщений. Для отправки полученных сообщений из брокера в Hive был также создан оператор — KafkaToHiveLoader (использован во всех приложениях представленной системы). В методе точки входа оператор была реализована следующая последовательность действий: извлечение набора данных из брокера с помощью spark сессии и дальнейшая его отправка в Hive.

Для сбора информации из источников reddit и Yahoo Finance реализация собственных операторов не потребовалась, как следствие, для оставшихся приложений был использован встроенный оператор PythonOperator, для которого были реализованы соответствующие функции сбора и отправки информации в топики брокера сообщений.

2.3 Выбор метрик для оценки потребления и рационального использования ресурсов

Интерфейс планировщика ресурсов предоставляет некоторые метрики, полезные при определении функционирования систем. За основу определения эффективности работы вышеописанных приложений были взяты следующие метрики:

— $memorySeconds$ — кумулятивная метрика, которая увеличивается раз в секунду, добавляя общее количество Мбайт;

— $vcoreSeconds$ — кумулятивная метрика, которая увеличивается раз в секунду, добавляя общее количество виртуальных ядер.

На основании данных метрик можно получить метрику использования памяти во времени. В YARN существуют $memory\_mb$ и $vcores$ — общее количество затрачиваемой памяти и ядер. Их использование невозможно для текущих работающих приложений, так как они определяют только конечные результаты приложения, следовательно, в данном контексте целесообразно осуществлять сбор данных о завершенных приложениях. Также имеется метрика $elapsed\_time$ — общее время с начала запуска приложения. Таким образом, для вычисления среднего количества всех используемых ядер за полное время работы была определена формула 1:

$$avg\_gb\_per\_second = \frac{memorySeconds}{elapsed\_time} \tag{1}$$

Для вычисления среднего значения используемой памяти в секунду была определена формула 2:

$$avg\_cores\_per\_second = \frac{vcoreSeconds}{elapsed\_time} \tag{2}$$

Для определения полной затрачиваемой памяти этапов в YARN существует метрика $stage\_memory\_bytes\_spilled$. Необходимо было задать критическую величину для возможности выделения памяти на каждый этап и контролировать каждый процесс путем сравнения с заданной величиной. Для определения возможной некорректной загрузки в хранилище требуется вычисление представленных ниже значений.

$Min\_task\_duration\_ms$ и $max\_task\_duration\_ms$ (предоставляются ин-

терфейсом YARN) — минимальное и максимальное время выполнения задачи в рамках одного этапа процесса. Также были вычислены: среднее время, затрачиваемое на выполнение задачи (взятие среднего от метрики $task\_duration$ — $avg(task\_duration)$) и медианное время, которое вычисляется через функцию $percentile\_approx(task\_duration)$.

С помощью среднего и медианного значений была определена метрика 3:

$$\frac{avg(task\_duration)}{percentile\_approx(task\_duration)} \tag{3}$$

Отношение из формулы 3 приблизительно равное единице показывает нормальное распределение длительности выполнения задач, то есть среднее значение достаточно близко к медиане [6]. Отношение 3 больше единицы (среднее значение больше медианы) показывает наличие выбросов во времени выполнения задач (среднее сдвигается в большую сторону, если время одной или нескольких задач гораздо больше остальных).

Ниже приведены метрики для расчета оптимального распределения ресурсов.

В интерфейсе YARN предоставляются отчеты по генерируемым дополнительным объемам данных на жестком диске, используемым для выполнения запроса (отчеты по spill-ам) через метрику $stage\_memory\_byte\_spilled$. Также из интерфейса YARN определяется вычисление количества Мбайт на входе и выходе в каждой задаче. Эти значения используются для определения метрики использования ресурсов. Общий объем входных данных определяется с помощью $sum(input\_bytes)$, общий объем выходных — $sum(output\_bytes)$. Данные метрики необходимы для определения максимального из значений: общий объем всех входных и выходных данных, общий объем всех записанных сериализованных данных на всех исполнителях в начале и в конце каждого из этапов — $max\_rw\_metric\_bytes$ — критическая величина для среднего объема памяти, используемого приложением ($avg\_gb\_per\_second$). То есть в зависимости от того, больше или меньше $avg\_gb\_per\_second$ по сравнению с $max\_rw\_metric\_bytes$, в задаче используется больше ресурсов, чем на самом деле читается или пишется (или меньше). В зависимости от этого принимается решение об изменении выделяемых на задачу ресурсов.

2.4 Реализация системы мониторинга высоконагруженной системы

Цель системы мониторинга — обеспечение стабильной работы и быстрого реагирования на возникающие проблемы, как следствие, осуществление поиск узких мест (загрузка данных на диск), использование некорректного количества ресурсов и т.д.

Реализация сбора метаданных о высоконагруженной системе После анализа функционирования инструмента разработки высоконагруженной системы было установлено, что сбор данных о временном хранении информации невозможен: одновременно с завершением работы приложения очищается хранение промежуточных результатов в памяти (в связи со спецификой работы инструмента). Кроме того, невозможно определить количество выделяемых ресурсов в точечный момент времени работы исполняемой задачи — предоставляется состояние только на конец ее работы.

С учетом поставленной цели было разработано Spark-приложение, осуществляющее обход по всей высоконагруженной системе, функционировавшей в течение предыдущего дня. В результате проведенных экспериментов было принято решение разработать систему мониторинга, которая циклично проходит по всем высоконагруженным приложениям так, что для каждого приложения осуществляется запрос на получение списка задач, при этом происходит последовательный обход всей иерархии приложения, что дает возможность получить полное представление о структуре и взаимосвязях компонентов приложения. Собранная информация о каждом этапе иерархии структурирована и организована в таблицы Hive. При разработке таблиц была обеспечена высокая степень связности между ними для возможности выполнения операций соединения. С целью минимизации количества операций соединения и упрощения понимания структуры данных были разработаны представления, использующиеся в генерации конечных отчетов.

Последовательный циклический обход стадий дал максимальную скорость по сравнению с распределенной обработкой.

Реализация сбора данных об использовании ресурсов высоконагруженной системой Данное приложение выполняет запрос к YARN REST API для получения информации о системе, таким образом с помощью ресурса YARN REST

API были получены все метрики, описываемые в разделе 2.3.

Приложение сбора данных об использовании ресурсов также запускается регулярно, запрашивая данные за прошедший промежуток времени.

Во избежание дублирования данных и увеличения числа файлов в файловой системе была реализована дополнительная группировка в одну партицию (данные запрашиваются несколько раз в течение суток, что приводит к многократной записи в одну и ту же партицию. Данная ситуация может привести к возникновению дублирующихся записей, что, в свою очередь, создает риски для целостности и точности хранимой информации).

Из полученной таблицы извлекаются уникальные даты, используемые для разбиения (партиционирования) таблицы, осуществляется чтение уже существующих данных, после чего происходит объединение этих таблиц с последующей очисткой от повторяющихся записей. Объединённые данные сохраняются обратно в целевую таблицу `monitoring_resources` с перезаписью, что обеспечивает актуальность и целостность информации.

Реализация генерации отчетов Полученные по метрикам данные необходимо было структурировать в отчеты. Каждый отчет представляет из себя sql-запрос над созданными ранее представлениями. Отчеты были разделены:

— spills (данные превышают доступный объем оперативной памяти),
— skew (неравномерное распределение данных между партициями),
— resources (полученные данные по использованию ресурсов).

В отчете о превышении доступного объема выделенной памяти используется фильтрация стадий. В конечный отчет было решено включить только те стадии, в которых объем данных, выведенных в память, превышает 10 Гб.

В отчете о неравномерном распределении данных между партициями за основу были взяты две метрики — отношение средней длительности процесса к медианной и отношение максимальной длительности процесса к минимальной. Для формирования данного отчета было определено, что должны быть выбраны только те процессы задач, у которых отношение средней длительности процесса к медианной больше пяти. При этом был задан минимальный порог продолжительности выполнения — 10 секунд (фильтрация на сложность операций).

Отчет по ресурсам необходим для выявления приложений, в которых потребление ресурсов превышает требования этих приложений. Avg_gbs_per_sec

больше max_rw_metric_bytes означает, что потребление ресурсов превышает фактический объем операций чтения или записи.

Реализация системы оповещений для сформированных отчетов Для реализации системы оповещений был создан отдельный оператор в рамках системы мониторинга, который обеспечивал подключение к Hive, выполнял sql запросы указанные в соответствующих файлах, отдавая результат в виде словаря с названиями полей и результатами выполнения запроса. Jinja-шаблон был необходим для корректного отображения текста оповещения в электронном письме, формируемые в шаблоне таблицы заполнялись на основании полученных данных из sql-запросов. Сообщение отправляется на почту через библиотеку airflow.utils.email.

Реализация графов системы мониторинга Для реализации последовательного запуска задач сбора метаданных о завершившихся приложениях и отправки отчета на электронную почту письмом с определенной периодичностью был объявлен граф, аналогично приложениям высоконагруженной системы. В самом графе были определены две последовательные задачи — задача заполнения Hive таблиц вычисленными и полученными из HistoryServer метриками и задача построения и отправки отчетности.

Для получения информации о потребляемых ресурсах приложений и записи в таблицу monitoring_resources также был сформирован DAG. Полученные с помощью него данные использовались для формирования конечного отчета.

3 Результаты

В результате выполненной работы были созданы: высоконагруженная система, приложение мониторинга за высоконагруженной системой и их взаимодействие.

Высоконагруженная система представляет собой три одновременно запущенных приложения по загрузке данных с удаленных серверов в созданное локальное хранилище. Для предоставления результатов работы системы мониторинга были реализованы два примера некорректного использования приложений системы, описанные ниже, так как при корректном использовании системы таблицы не отображают информации.

Результат работы системы мониторинга Результатом работы системы мониторинга за высоконагруженной системой является отправка оповещения на личную почту, указанную в параметрах графа. Пример сгенерированного отчета, отправленного на почту, представлен на рисунке 1:

Отчет по Spark-приложениям за 2024-12-21 Входящие ×

arrishok@mail.ru
кому: мне

Spill

Топ приложений по spill-ам в память.

app_name	job_id	stage_id	memory_spilled	disk_spilled
weather	1	1	50.2 GiB	7.4 GiB

Skew

Топ приложений по перекосу во времени работы задач внутри stage.

- avg_med_ratio - отношение средней длительности задачи к медианной
- max_min_ratio - отношение максимальной длительности задачи к минимальной

app_name	job_id	stage_id	avg_med_ratio	max_min_ratio	avg_task_duration	med_task_duration	min_task_duration	max_task_duration
weather	1	1	6.68	36.96	0m 5s 92ms	0m 0s 764ms	0m 0s 273ms	0m 13s 751ms

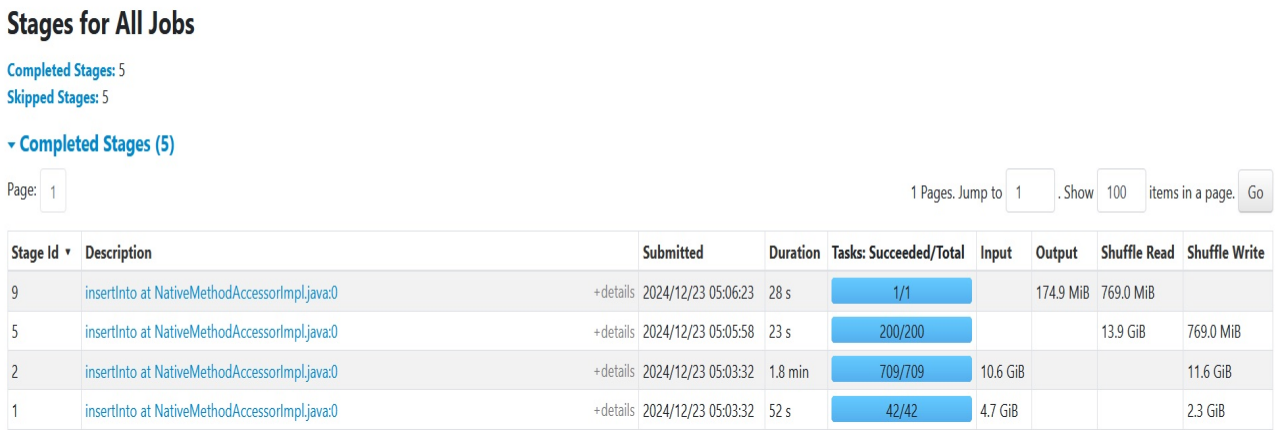
Использование ресурсов

Приложений с высоким использованием ресурсов не найдено.

Рисунок 1 – Ежедневный отчет о работе приложений за прошедшие сутки

Превышение данными доступного объема оперативной памяти Превышение данными доступного объема оперативной памяти отслеживается путем сбора информации из интерфейса YARN.

Указанный в отчете раздел «Spill» предоставляет ссылки на приложение, задачу, в рамках которой выполняется стадия, а также на саму стадию. Дополнительно в отчете выводятся: размер данных, которые не могли быть обработаны в оперативной памяти и размер данных, которые были сериализованы и записаны на диск. Попавшая в отчет стадия 1 в Spark UI представлена на рисунке 2:



Stages for All Jobs

Completed Stages: 5
Skipped Stages: 5

▾ Completed Stages (5)

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
9	insertInto at NativeMethodAccessorImpl.java:0 +details	2024/12/23 05:06:23	28 s	1/1		174.9 MiB	769.0 MiB	
5	insertInto at NativeMethodAccessorImpl.java:0 +details	2024/12/23 05:05:58	23 s	200/200			13.9 GiB	769.0 MiB
2	insertInto at NativeMethodAccessorImpl.java:0 +details	2024/12/23 05:03:32	1.8 min	709/709	10.6 GiB			11.6 GiB
1	insertInto at NativeMethodAccessorImpl.java:0 +details	2024/12/23 05:03:32	52 s	42/42	4.7 GiB			2.3 GiB

Рисунок 2 – Стадии в рамках задачи

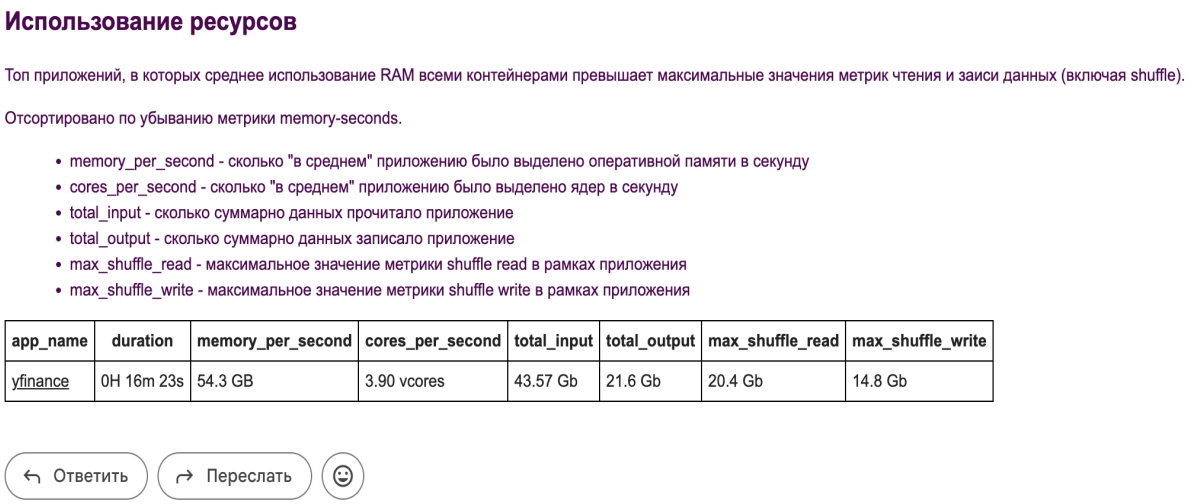
Результирующие метрики — общий объем данных, которые были временно сохранены в памяти, но не смогли быть обработаны из-за нехватки доступной оперативной памяти и объем данных, которые были временно записаны на диск вычисляются путем сложения соответствующих метрик по каждой из операций, работающих в рамках стадии.

Неравномерное распределение данных Неравномерное распределение данных (раздел «Skew») отслеживается путем измерения времени работы каждого из процессов внутри стадий и использования соответствующих метрик.

На основании метрик, полученных из интерфейса Spark (медианное время, минимальное и максимальное время, затрачиваемое на выполнение процесса), вычисляются остальные, как было описано ранее. На основании результатов отчета метрика avg_med_ratio составляет 6.68. Данное соотношение указывает на значительное расхождение во времени выполнения операций, что, в свою очередь, указывает на то, что некоторые операции обрабатывают гораздо больший объем данных, чем остальные.

Превышение использования ресурсов Превышение использования ресурсов отслеживается путем выявления приложений, в которых среднее использование

RAM превышает максимальные значения метрик чтения и записи данных. Раздел «Использование ресурсов» отправленного на email отчета представлен на рисунке 3:



Использование ресурсов

Топ приложений, в которых среднее использование RAM всеми контейнерами превышает максимальные значения метрик чтения и заиси данных (включая shuffle).

Отсортировано по убыванию метрики memory-seconds.

- memory_per_second - сколько "в среднем" приложению было выделено оперативной памяти в секунду
- cores_per_second - сколько "в среднем" приложению было выделено ядер в секунду
- total_input - сколько суммарно данных прочитало приложение
- total_output - сколько суммарно данных записало приложение
- max_shuffle_read - максимальное значение метрики shuffle read в рамках приложения
- max_shuffle_write - максимальное значение метрики shuffle write в рамках приложения

app_name	duration	memory_per_second	cores_per_second	total_input	total_output	max_shuffle_read	max_shuffle_write
yfinance	0H 16m 23s	54.3 GB	3.90 vcores	43.57 Gb	21.6 Gb	20.4 Gb	14.8 Gb

Ответить Переслать

Рисунок 3 – Отчет по использованию ресурсов

Из отчета видно, что для приложения yfinance было выделено 54 Гб памяти и 4 виртуальных ядра (1 из них на драйвер, 3 — на исполнители), чтобы прочитать 43 Гб данных. Фактический объем операций чтения и записи составляет 35.2 Гб, что означает превышение потребления ресурсов почти на 20 Гб, по этой причине приложение yahoo_finance попало в отчет.

ЗАКЛЮЧЕНИЕ

В заключение следует подчеркнуть актуальность и значимость данной темы в области информационных технологий. Основная идея отражена мною в статье «Стратегии оптимизации высоконагруженных систем обработки данных» опубликованной научным журналом «Студенческий вестник».

В выпускной квалификационной работе более детально была изучена тема высоконагруженных систем, а именно:

- рассмотрены области применения высоконагруженных систем и их мониторинга в современном мире;
- выявлены требования к реализации высоконагруженной системы и приложения для ее мониторинга;
- проведен анализ надежности и нагрузки на систему, сформирована архитектура системы, определено взаимодействие компонентов;
- реализованы основные компоненты высоконагруженной системы;
- исследованы и определены основные метрики для выявления неравномерного распределения данных в процессе обработки и загрузки, превышения доступного объема памяти, оптимального использования ресурсов для задач;
- разработана система мониторинга высоконагруженных приложений;
- продемонстрированы результаты работы системы мониторинга.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Сытежева, А. СТРАТЕГИИ ОПТИМИЗАЦИИ ВЫСОКОНАГРУЖЕННЫХ СИСТЕМ ОБРАБОТКИ ДАННЫХ [Текст] // Студенческий вестник. — 2025. — Т. 3, № 14(347). — С. 9–11.

2 The True Cost of Downtime 2022 [Электронный ресурс] [Текст]. — Alpharetta : Siemens. — URL: https://assets.new.siemens.com/siemens/assets/api/uuid:3d606495-dbe0-43e4-80b1-d04e27ada920/dics-b10153-00-7600truecostofdowntime2022-144.pdf (Дата обращения 17.10.2023). Загл. с экр. Яз. англ.

3 Подольный, В. Архитектура высоконагруженных систем [Текст]. — Москва : Сам Полиграфист, 2024.

4 Харазян, А. Анализ методов и инструментов оптимизации функционирования высоконагруженных систем [Текст] // Цифровая экономика. — 2023. — Т. 23. — С. 85–89.

5 White, T. Hadoop: The Definitive Guide [Text]. — Sebastopol : O'Reilly Media, 2021.

6 Кремер, Н. Теория вероятностей и математическая статистика [Текст]. — Москва : Юрайт, 2023.