РЕФЕРАТ

Магистерская диссертация содержит 59 страниц, 4 таблицы, 11 рисунков. Список использованных источников содержит 11 позиций.

НЕЙРОННАЯ СЕТЬ, БОЛЬШИЕ ДАННЫЕ, АВТОМАТИЧЕСКАЯ НАСТРОЙКА ПАРАМЕТРОВ, СИСТЕМА HADOOP

Магистерская диссертация посвящена разработке подхода к построению комплекса программного обеспечения и его архитектуре, что позволяет производить автоматическую оптимальную настройку системы Hadoop под задачи, выполняющие преобразования над большим объемом информации в базе данных. Результатом работы разрабатываемого комплекса является определение значений параметров Hadoop при которых выполняемая задача либо выполняется за оптимальное количество времени, либо потребляет оптимальное количество ресурсов.

Для решения поставленной задачи предложен и разработан подход, который включает в себя приложение для сбора статистики и уменьшения размерности полученных данных, модель машинного обучения, обученная для предсказания целевой метрики, а также подход к расчету оптимальных параметров конфигурации в условиях ограниченности ресурсов.

Рассматривается также методика и подход для внедрения такой модели в процесс запуска задач в системе Hadoop.

Оглавление

СПИСОК УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СОКРАЩЕНИЙ, ТЕРМИНОВ 4
ВВЕДЕНИЕ 6
ОСНОВНАЯ ЧАСТЬ 10
1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 11
1.1 Обзор литературы 11
1.2 Постановка формальной задачи определения оптимальных параметров 17
1.3 Архитектура разрабатываемого комплекса 19
1.4 Java приложение для сбора статистики 21
1.5 Модель машинного обучения 25
1.6 Алгоритм определения оптимальных параметров конфигурации 28
1.7 Внедрение в Oozie Workflow 30
2. ПРАКТИЧЕСКАЯ ЧАСТЬ 33
2.1 Сбор данных для обучения 33
2.2 Обучение нейронной сети 36
2.3 Реализация алгоритма определения оптимальных параметров 39
2.4 Реализация внедрения в Oozie Workflow 42
ЗАКЛЮЧЕНИЕ 47
2.5 Оценка и достоверность результатов 47
2.6 Дальнейшее развитие 48
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 50

СПИСОК УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СОКРАЩЕНИЙ, ТЕРМИНОВ

HADOOP	–	Проект фонда Apache Software Foundation, свободно распространяемый набор утилит, библиотек и фреймворк для разработки и выполнения распределённых программ
SPARK	–	Фреймворк с открытым исходным кодом для реализации распределённой обработки неструктурированных и слабоструктурированных данных, входящий в экосистему проектов Hadoop
YARN	–	Система для планирования заданий и управления кластером
HDFS	–	Распределенная файловая система Hadoop
APACHE OOZIE	–	Серверная система планирования рабочих процессов для управления заданиями Hadoop
KERAS	–	Открытая нейросетевая библиотека, написанная на языке Python
REST	–	Архитектурный стиль взаимодействия компонентов распределённого приложения в сети
API	–	Описание способов, которыми одна компьютерная программа может взаимодействовать с другой программой
DATANODE	–	Один их множества серверов кластера с программным кодом, отвечающим за файловые операции и работу с блоками данных
RESOURCE MANAGER	–	Планировщик ресурсов YARN

UNIX	–	Система описания моментов во времени, принятая в Unix и других POSIX-совместимых операционных системах.
RELU	–	Функция активации rectified linear unit

ВВЕДЕНИЕ

Работа посвящается разработке подхода к построению комплекса программного обеспечения, позволяющего производить автоматическую оптимальную настройку системы Hadoop под выполняемые на ней задачи.

Актуальность работы заключается в том, что в настоящее время с увеличением объема хранимых данных возникает потребность эти данные обрабатывать для получения статистик и новой информации, необходимых для дальнейшего анализа работающих систем. Для этого применяются системы обработки больших данных: Hadoop, Spark и другие. В связи с этим становится актуальной проблема по их оптимальной настройке под выполняемые задачи.

Цель работы: разработать комплексное приложение, которое может быть использовано в командах, занимающихся разработкой задач в системе Hadoop. Приложение оптимизирует целевые показатели (время выполнения, используемая память) с целью улучшения производительности путем автоматического подбора параметров запуска на основе собранных ранее статистик.

Для достижения заявленной цели в работе предусматривается решение следующего комплекса задач:

- разработка приложения, которое будет собирать различные статистики и генерировать набор данных для обучения;
- использование методов машинного обучения для уменьшения размерности данных;
- построение и обучение модели машинного обучения для предсказания целевой метрики;
- разработка алгоритма расчета оптимальных параметров конфигурации;
- модификация алгоритма в целях работоспособности при ограниченности ресурсов кластера

- внедрение алгоритма в процесс запуска задач.

Важность работы обуславливается тем, что специалисты по обработке больших данных сталкиваются с необходимостью обрабатывать информацию из баз данных путем запуска задач в системе Hadoop. Но одной из проблем использования этой системы является предварительная настройка большинства параметров. При этом для каждой выполняемой задачи необходимо минимизировать целевые затраты: время выполнения, объем используемой памяти, другие целевые критерии, присущие конкретной области. Таких параметров предоставляется от нескольких десятков до нескольких сотен [1], что сильно усложняет подход к их определению.

Большинство параметров настраиваются администратором единожды по регламенту, предоставленному руководством, что может негативно сказаться на выполнение конкретной задачи. Также обычно не учитывается топология архитектуры кластеров, текущая нагрузка, обязательные параллельные задачи, которые запускаются самой системой обработки больших данных (например, удаление мусора, перераспределение данных между узлами, очистка буферов и др.). Важно понимать, что большинство параметров непредсказуемо взаимосвязаны. Например, при увеличении значения одного параметра можно получить и положительное, и отрицательное влияние на систему другого параметра. Чтобы не задумываться обо всех смежных процессах, система рассматривается в виде “черного ящика”, который будет моделироваться нейронной сетью.

Нейросетевое моделирование – это один из современных подходов машинного обучения, который широко применяется для прогнозирования различных показателей в случаях, когда удобно представлять систему “черным ящиком”. Кроме того, подходы нейросетевого моделирования способны выдавать достаточно точный результат для вновь поступающих входных данных, но для этого требуется большой объем данных для обучения самой модели.

Для получения информации о выполняемых задачах каждый компонент системы Hadoop предоставляет специальный интерфейс API, к которому можно подключаться путем отправки соответствующих запросов. Поэтому необходимо многократно запускать задачи с различными параметрами для сбора эффективного набора данных для обучения нейронной сети. Для получения такого набора данных разработано java-приложение, которое подключается к компонентам Hadoop: Yarn, HDFS, Hbase, Oozie с целью выгрузки всех необходимых значений за какой-либо промежуток времени и генерации файлов, которые удобны для считывания программами разработки нейронных сетей.

Полученные данные подаются на вход нейронной сети, которая написана с использованием библиотеки Keras на языке Python. Сеть разработана с целью прогнозирования целевой метрики, например, времени выполнения, которую нужно минимизировать. Для минимизации разработан отдельный алгоритм на языке Python, который модифицирован для решения проблемы ограниченности ресурсов кластера, с которого собираются метрики. Например, такая проблема может возникнуть в ситуациях, когда алгоритм рекомендует выставить объем памяти в недопустимо большое значение для увеличения скорости обработки данных.

Полученный алгоритм встраивается в выполнение той же задачи путем определения потока управления процессов системы Apache Oozie с целью автоматической подстановки рекомендованных параметров.

В разделе 1.1 описаны существующие публикации по теме работы, приведены их достоинства и недостатки. В разделе 1.2 ставится формальная постановка задачи прогнозирования оптимальных параметров запуска задачи. В разделе 1.3 приведена архитектура разработанного комплекса программного обеспечения, каждый компонент которого описан подробнее в последующих пунктах 1.4 - 1.7. В пункте 2.1 практической части представлен процесс сбора данных для обучения нейронной сети. Сам процесс обучения, настройка,

обработка нейронной сети описана в пункте 2.2. Пункт 2.3 представляет программное описание разработанного алгоритма для определения оптимальных параметров, а также результаты его работы. Пункт 2.4 отражает то, как было произведено внедрение разработанного программного комплекса в систему запуска задач Hadoop. В заключении к работе представлены оценки результатов и их достоверность (пункт 3.1), а также дальнейшие пути развития темы (пункт 3.2).

ОСНОВНАЯ ЧАСТЬ

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Обзор литературы

В главе представлены основные идеи и принципы, которые опубликованы на данный момент в открытых источниках и научных статьях. Описывается их основная концепция, достоинства и недостатки, которые необходимо учитывать в реализации собственного алгоритма оптимизации.

1) *A Survey on Automatic Parameter Tuning for Big Data Processing Systems, 2020. Herodotos Herodotou, Yuxing Chen, Jiaheng Lu.* [1]

В работе приведен анализ подходов, которые решают проблемы, связанные с автоматической настройкой параметров приложений и систем обработки больших данных (Hadoop, Spark, Apache Storm, Heron, Flink и другие). К таким проблемам можно отнести существование огромного количества параметров конфигураций, которые могут влиять на производительность системы неочевидным и сложным образом, а также неструктурированный характер входных данных. Эти проблемы ведут к непредсказуемому времени выполнения задач, а также к нестабильным нагрузкам на аппаратуру.

В статье представлено исследование нескольких подходов, направленных на решение вышеупомянутых проблем, либо для точного прогнозирования производительности приложения при различных настройках параметров, либо для нахождения почти оптимальных настроек параметров для различных сценариев.

Подходы делятся на шесть широких категорий: *на основе правил, моделирование затрат, на основе моделирования, экспериментально, машинное обучение, и адаптивный тюнинг.*

Каждый из шести подходов превосходит один или более аспектов, имеющих свои уникальные сценарии применения. Начиная с незнания настройки параметров, экспериментальный метод является наиболее доступным. По мере накопления опыта можно создавать правила или функции затрат для эффективного повышения производительности системы. По мере увеличения сложности систем моделирование небольших компонентов помогает лучше понять характеристики системы. При попытке настроить очень сложные системы и приложения с большим пространством параметров могут быть полезны методы машинного обучения с соответствующими обучающими данными, поскольку они обычно игнорируют внутренние компоненты системы. Наконец, для специальных и длительных рабочих нагрузок адаптивный подход – лучший вариант.

Так как по мере увеличения размера и сложности систем аналитики больших данных растет и потребность в автоматическом обеспечении хорошей и надежной производительности системы, способной справляться с постоянно растущими темпами производства данных, авторы ставят ряд открытых проблем:

1) *неоднородность* – проблема, вызванная наличием аппаратных узлов с различными емкостями и количествами ядер/памяти;
2) *облачная среда* – проблемы, связанные с несколькими арендаторами, накладными расходами и взаимодействием производительности при виртуализации;
3) *аналитика в реальном времени* – проблемы, вызванные предоставлением аналитики на лету, так как обычно такие системы используют несколько других систем обработки больших данных.

Углубленное понимание существующих подходов, представленное в этой статье, имеет ключевое значение для решения этих новых проблем эффективным и действенным способом для достижения конечной цели

разработки действительно самонастраивающихся и самоконфигурируемых систем.

2) *Using Machine Learning to Optimize Parallelism in Big Data Applications, 2017. Alvaro Brandon Hernandeza, Maria S. Pereza, Smrati Gupta, Victor Muntes-Mulero.* [2]

В этой статье предлагается метод на основе машинного обучения, который рекомендует оптимальные параметры для распараллеливания задач в рабочих нагрузках с большими данными. Путем мониторинга и сбора метрик на уровне системы и приложения, находятся статистические корреляции, которые позволяют характеризовать и прогнозировать влияние различных настроек параллелизма на производительность. Эти прогнозы используются, чтобы рекомендовать пользователям оптимальную конфигурацию перед запуском их рабочих нагрузок в кластере, избегая возможных сбоев, снижения производительности и нерационального использования ресурсов. Метод оценивается с помощью теста из 15 приложений Spark на стенде Grid5000. Наблюдается прирост производительности до 51% при использовании рекомендуемых настроек параллелизма. Модель также поддается интерпретации и может дать пользователю представление о том, как различные показатели и параметры влияют на производительность.

Результаты этой статьи показывают, что можно точно предсказать время выполнения приложения на основе больших данных с разными размерами файлов и настройками параллелизма с использованием правильных моделей. В качестве моделей в статье рассматривались: Bayesian Ridge, Linear Regression, SGD Regressor, Lasso, Gradient Boosting Regressor, Support Vector Regression, MLPRegressor.

3) *Learning-based Automatic Parameter Tuning for Big Data Analytics Frameworks, 2018. Liang Bao, Xin Liu, Weizhao Chen.* [3]

В статье представлена собственная реализация системы автоматической настройки параметров, которая направлена на оптимизацию времени выполнения приложений на фреймворках аналитики больших данных.

Система изначально конструирует небольшой испытательный стенд из большой производственной системы, чтобы он мог генерировать релевантные образцы данных для лучшего обучения прогнозируемой модели в заданном временном ограничении. Кроме того, такой алгоритм создает набор образцов, которые могут обеспечить широкий охват многомерного пространства параметром конфигурации.

Алгоритм не является стандартным в области машинного обучения. Он базируется на выборке гиперкуба LHS (latin hypercube sampling) для дальнейшей генерации эффективных выборок в многомерном пространстве параметров, которые используются для выбора перспективных конфигураций в ограниченном пространстве, предлагаемых существующими лучшими конфигурациями.

4) *Quality Assurance for Big Data Application– Issues, Challenges, and Needs, 2016. Chuanqi Tao, Jerry Gao.* [4]

В данной работе авторы проводят исследования о том, каким образом необходимо осуществлять валидацию приложений обработки больших данных для обеспечения качества работы всей системы, возникающие проблемы и потребности при этом. Также обсуждаются факторы качества и проводится сравнение между традиционным тестированием и тестированием приложений обработки больших данных.

Авторы отмечают основной процесс тестирования, состоящий из следующих шагов:

1. Функциональное тестирование систем больших данных для алгоритмов, возможности обучения, специфичных для предметной области функций:
2. Функциональное тестирование систем больших данных для целостности системы, безопасности надежности;
3. Тестирование визуализации, набора данных, удобства использования;
4. Тестирование, связанное с временными аспектами: тестирование работы в реальном времени, тестирование в течение срока службы и т.д.

5) *Machine Learning and Big Data Processing: A Technological Perspective and Review, 2018. Roheet Bhatnagar.* [5]

Данная статья является обзорной и раскрывает роль алгоритмов и методов на основе машинного обучения в обработке и аналитике больших данных. Автор представляет обзор проблем, связанных с основными подходами машинного обучения: избыточность данных, зашумленность, неоднородный характер, дискретность, маркировка, несбалансированность, представление функций и их выбор.

Все эти проблемы необходимо учитывать при создании системы автоконфигурирования приложений анализа и обработки больших данных. В целом документ нацелен на предоставление как текущих практик, так и будущих направлений исследований в области обработки больших данных с использованием методов машинного обучения.

Основным наблюдениям является то, что статей по данной проблематике достаточно мало, а существующие опубликованы совсем

недавно. Основная причина заключается в том, что системы обработки больших данных усложняются в процессе роста вычислительных мощностей. Вследствие этого появляется все больше настраиваемых параметров и конфигураций, необходимых для стабильной и оптимальной работы.

Исследование методов автоматической настройки параметров – многообещающий, но достаточно сложный подход оптимизации производительности системы. Сейчас самим пользователям приходится решать вопрос с проблемой управления такими системами, но для этого не хватает инструментов для упрощения конфигурирования, а также достаточной информации о многочисленных параметрах, их корреляциях и зависимостях. Например, системы Hadoop и Spark имеют более сотни параметров, которые можно настраивать, однако их настройка является сложной задачей для человека, вследствие чего обычно принимают настройку по умолчанию, которая совершенно не является оптимальной для текущей задачи.

Поэтому основной задачей является реализация такой системы оптимизации выполнения Big Data приложений, которая будет учитывать все ключевые аспекты конфигурирования и предоставлять удобный интерфейс пользователю, либо будет полностью автономной.

1.2 Постановка формальной задачи определения оптимальных параметров

Задача прогнозирования оптимальных параметров для запускаемых процессов в системе Hadoop представляет собой решение оптимизационной задачи следующего содержания.

Пусть $p \in Params$, где $Params$ – множество входных параметров, системы Hadoop. Рассмотрим параметры следующих компонентов Hadoop: HDFS – распределенная файловая система для хранения файлов с возможностью потокового доступа к информации, Yarn – система для планирования заданий и управления кластером, Hbase – нереляционная, распределенная база данных, Jvm – виртуальная машина Java, исполняющая байт-код программ, а также ряд собственных параметров. Некоторые из этих параметров приведены в Приложении 1.

Собственными параметрами будем считать те, которые присущи нашей конкретной задаче, которую мы запускаем в Hadoop. Их перечень представлен в таблице 1.1.

Таблица 1.1 – Параметры экспериментальной задачи

Параметр	**Значения**	**Описание**
tableSize	654 Б - 2234021317 Б	Объем данных для обработки
tableRegions	1 - 1000	Количество регионов таблицы в hbase

В качестве целевого параметра будем рассматривать время выполнения задачи с текущей конфигурацией над полученным объемом данных в Hbase. В итоге, имеем формулу 1.1 для расчета множества продолжительностей выполнения задачи через функцию F, определяющую реальную зависимость от входных параметров.

$$D = durations = F\ (hdfsParams,\ yarnParams,$$
$$hbaseParams, javaParams,\ ownParams) = F(Params) \quad (1.1)$$

где: $hdfsParams$ – параметры настройки hdfs,

$yarnParams$ – параметры настройки yarn,

$hbaseParams$ – параметры настройки hbase,

$javaParams$ – параметры настройки java,

$ownParams$ – параметры, присущие нашей задаче.

Требуется найти оптимальные входные параметры конфигурации, при которых значение целевой метрики, а именно времени выполнения, будет минимальным. Пусть w – вектор обучаемых параметров модели. Минимизируем функцию ошибки модели, что представлено в формуле 1.2 через квадратичную функцию ошибки.

$$w^* = (w_0^*.., w_n^*) = \underset{w_0,..,w_n}{\operatorname{argmin}}(model - D)^2, \quad (1.2)$$

где $model$ – множество выходных значений, прогнозируемых моделью машинного обучения.

Найденные значения $w_0^*, .., w_n^*$ необходимы модели для предсказания длительности выполнения задачи для новых входных параметров конфигурации. Так модель предсказывает значения $model$ согласно формуле 1.3.

$$model = F_{model}(Params), \quad (1.3)$$

где F_{model} – функция, определяющая зависимость длительности выполнения задачи от входных параметров согласно обученной модели.

Но для поиска оптимальных значений входных $params$, при которых достигается минимальное время выполнения $duration$, полученная функция

прогнозирования модели F_{model} с известными (обученными) значениями $w_0^*, \ldots, w_n^*$, минимизируется относительно параметров $params$ с помощью численного метода множителей Лагранжа [6]. Формальная постановка задачи минимизации функции F_{model} нескольких переменных представлена формулой 1.4.

$$p_0^*, \ldots, p_n^* = \underset{p_0,\ldots,p_n}{\operatorname{argmin}} F_{model}(Params), \tag{1.4}$$

где $p_0^*, \ldots, p_n^*$ – оптимальные параметры запускаемой задачи в Hadoop, при которых прогнозируемое время выполнения минимально.

Основная проблема заключается в том, что найденные значения могут выходить за рамки допустимых диапазонов, поэтому необходимо учитывать ряд линейных ограничений в оптимизационном алгоритме. Основные ограничения каждого входного значения представлены формулой 1.5.

$$bounds = \begin{cases} p > 0, & p \in p^* \\ p < \max\_value, & p \in p^* \end{cases}, \tag{1.5}$$

где $\max\_value$ определяется для каждого параметра отдельно администратором системы.

1.3 Архитектура разрабатываемого комплекса

Архитектура разрабатываемого комплексного приложения запусков задач в системе Hadoop приведена на рис. 1.1.

На первом этапе необходимо собрать набор обучающих данных на основе запуска map-reduce задачи с различными конфигурациями Hadoop. Значения параметров этих конфигураций собирает Java приложение, которое подключается к отдельным компонентам через открытый Rest API. На вход

Java приложению подается дата, начиная с которой необходимо искать релевантные запуски.

Далее сгенерированный набор данных поступает в модель машинного обучения, архитектура которой описана в следующих главах. Модель обучается и предоставляет свои оптимальные обучаемые параметры $w_0^*, .., w_n^*$ функции нахождения времени выполнения задачи F_{model}. На основе полученных значений алгоритм определения оптимальных параметров конфигурации предоставляет параметры конфигурации Hadoop. Также предоставляется возможность проведения повторного сбора статистик и переобучения модели перед выполнением основной задачи.

Сам алгоритм встроен в поток выполнения задачи через Oozie Workflow, который применяет рекомендованные параметры перед выполнением самой задачи. В результате пользователю не приходится запускать этот алгоритм собственноручно, а задача будет выполняться с оптимальной конфигурацией.

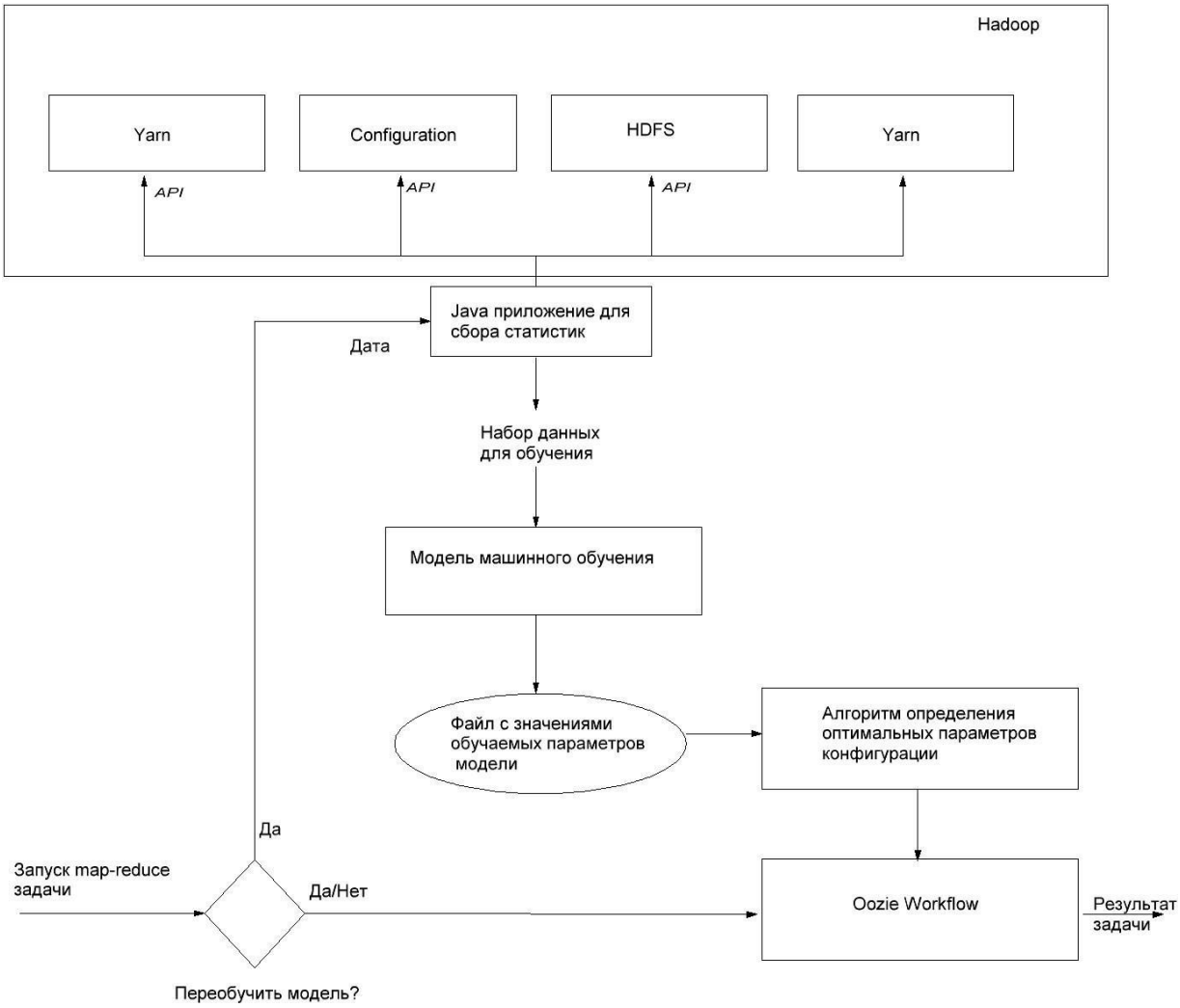

Рис. 1.1 – Архитектура комплекса автоматического конфигурирования параметров

Подробное описание работы компонентов архитектуры представлено в следующих разделах.

1.4 Java приложение для сбора статистики

Приложение реализовано на языке Java версии 8. В его основе лежит сбор статистики через REST API и различных библиотек с последующей генерацией файла формата csv, содержащего обучающие данные для модели машинного обучения. На вход приложению подается начальная дата, с

которой необходимо рассматривать запуски задач в Hadoop. Получение статистик осуществляется из: Yarn, Hdfs, Hbase, конфигурационных файлов.

В Yarn компонент Resource Manager предоставляет REST API, позволяющий пользователям получать информацию о кластере, о его статусе, метриках, информацию всех DataNode и о всех запущенных на них задачах. Для определения списка задач, начавшихся с определенной даты, используется http запрос, отправляемый на адрес http://<address>:<port>/ws/v1/cluster/apps/?startTimeBegin=<time>, где <address> - адрес расположения Resource Manager, <port> - порт подключения к ResourceManager, <time> - начальное время, с которого осуществляется поиск запущенных приложений в UNIX формате. Пример ответа представлен в Приложении 2. Полученный ответ обрабатывается, вследствие чего определяются искомые задачи через фильтрацию по типу задачи и паттерну названия. Для получения более подробных метрик используется запрос к http://<address>:<port>/ws/v1/cluster/apps/{appid}, где <appid> - идентификатор задачи, полученной из запроса к предыдущему адресу. В результате запросов к Yarn получаем перечень параметров, представленных в таблице 1.2.

Таблица 1.2 – Параметры, полученные из Yarn API

Параметр	**Описание**
finishedTime	Время окончания выполнения задачи
startTime	Время начала выполнения задачи
finalStatus	Статус задачи

Продолжение таблицы 1.2

Параметр	Описание
allocatedVcores	Сумма виртуальных ядер, выделенных работающим контейнерам задачи
allocatedMemory	Сумма памяти, выделенная для работающих контейнеров задачи

Конфигурации компонентов Hadoop прописаны в специальных файлах кластера: hdfs-site.xml, hbase-site.xml, mapred-site.xml, core-site.xml, yarn-site.xml. Их считывание происходит с помощью библиотеки для языка Java org.apache.hadoop.hadoop-common версии 3.3.0, которая предоставляет доступ ко всей конфигурации. Перечень полученных параметров этих файлов представлен в таблице 1.3. Представлен лишь ограниченный набор параметров, которые вносят наиболее значимый вклад в длительность выполнения задачи.

Таблица 1.3 – Параметры, полученные из xml файлов конфигурации

Параметр	Описание
dfs.blocksize	Размер блока в hdfs
dfs.replication	Фактор репликации
hbase.client.scanner.caching	Количество строк, выбираемое при вызове next() сканера hbase.
mapreduce.task.io.sort.factor	Количество потоков для одновременного объединения при сортировке файлов
mapreduce.task.io.sort.mb	Объем буферной памяти при сортировке файлов

Продолжение таблицы 1.3

Параметр	**Описание**
mapreduce.map.sort.spill.percent	Мягкое ограничение в буфере сериализации
mapreduce.job.reduces	Максимальное количество задач reduce
mapreduce.job.maps	Максимальное количество задач map
mapreduce.map.cpu.vcores	Количество ядер на задачу map
mapreduce.reduce.cpu.vcores	Количество ядер на задачу reduce
mapreduce.map.memory.mb	Максимальное количество памяти на задачу map
mapreduce.reduce.memory.mb	Максимальное количество памяти на задачу reduce

Библиотека hadoop-common также предоставляет интерфейс взаимодействия с базой данных Hbase, из которой получаем размер таблиц, а также количество их регионов.

Сопоставлять набор данных из конфигураций и hbase с запускаемой задачей необходимо через публикацию соответствующих значений в поток логирования работающего приложения, либо через их преждевременное снятие при проведении ряда экспериментальных запусков с целью сбора обучающих данных для модели.

В результате java-приложение получает для каждой выполняемой задачи весь набор параметров, при которых они были выполнены. Такой набор примеров записывается в выходной csv файл, пример которого представлен на рис. 1.2.

```
blocksize;replication;javaMemory;reduce.memory.mb;reduce.cpu.vcores;tableNumber;job.maps;parallelJobsLimit;
134217728;2;7168;3072;267;230;1073;2;109;20;2048;637083;1086;71;256;13;90;288;10099
402653184;3;5120;3072;274;237;1105;1;291;23;3072;297303;1161;65;256;14;90;258;2643
268435456;3;7168;2048;252;238;1149;3;143;22;4096;575375;1140;80;512;14;80;290;5545
134217728;2;5120;4096;265;249;1039;2;207;24;2048;674228;1012;72;256;13;90;176;10490
268435456;3;5120;3072;235;234;1125;1;181;27;4096;483084;1094;67;256;11;80;230;7010
134217728;3;7168;3072;137;245;1102;1;134;30;4096;454830;1125;68;512;10;90;135;7760
134217728;3;5120;2048;229;247;1057;1;190;30;2048;329425;1128;71;512;14;90;203;10225
402653184;3;7168;2048;275;233;1197;3;232;20;3072;701804;1053;74;256;11;90;186;7600
402653184;2;7168;4096;279;245;1096;3;124;28;3072;778179;1006;68;512;15;90;252;10153
268435456;3;7168;4096;215;234;1130;1;203;27;4096;111673;1076;66;256;10;90;127;2984
134217728;3;5120;4096;251;242;1026;2;149;23;2048;656802;1166;68;256;12;90;273;8622
268435456;2;7168;2048;102;245;1143;3;219;28;4096;382457;1108;76;512;12;80;163;10357
268435456;3;7168;4096;116;242;1119;3;116;26;2048;184787;1139;65;512;14;90;143;4449
402653184;2;5120;2048;258;249;1196;2;278;20;3072;194584;1107;79;512;10;90;193;2856
402653184;2;5120;4096;105;233;1147;3;213;22;3072;278682;1115;70;256;10;80;252;3534
268435456;2;5120;2048;115;250;1185;1;234;30;4096;91219;1144;65;512;11;80;250;1887
134217728;3;7168;3072;133;232;1051;3;158;24;4096;438618;1054;66;256;11;90;286;10393
402653184;2;7168;4096;293;249;1098;1;232;25;2048;231611;1025;75;256;13;90;255;2709
268435456;3;7168;4096;138;234;1008;3;201;23;4096;158769;1012;69;512;10;90;157;3280
402653184;2;5120;4096;140;236;1189;2;272;21;4096;178612;1018;70;256;14;90;107;5337
134217728;2;5120;4096;268;230;1088;1;168;25;3072;741589;1073;64;512;13;90;228;8889
134217728;3;5120;4096;230;238;1078;3;132;29;2048;318993;1094;78;256;12;90;180;10164
134217728;3;5120;4096;231;243;1139;1;118;20;4096;83227;1134;80;256;15;90;202;2309
268435456;3;7168;2048;245;235;1151;1;299;29;3072;202684;1064;67;512;15;80;131;5167
268435456;3;7168;3072;254;242;1005;3;169;26;3072;185018;1054;70;512;12;80;149;3970
268435456;2;7168;2048;203;246;1076;1;198;26;2048;524584;1142;78;512;12;90;209;10793
134217728;2;5120;3072;279;238;1011;1;132;25;4096;83386;1139;75;512;13;80;144;5192
134217728;3;5120;3072;207;235;1103;2;103;27;3072;660063;1169;67;256;13;80;263;9551
```

Рис.1.2 – Пример начала csv файла с обучающими данными

1.5 Модель машинного обучения

Набор обучающих данных, полученных из java-приложения для сбора статистики, на которых обучается модель машинного обучения, имеет огромный набор признаков. Этот набор признаков уменьшается для выделения наиболее значимых параметров запускаемой задачи.

Механизмом отбора признаков на основе их важности выбран ансамблевый алгоритм на основе дерева решений Extremely Randomized Trees [7]. Этот алгоритм работает путем создания большого количества необрезанных регрессионных деревьев из обучающего набора и дальнейшего усреднения их прогнозов в случае регрессионной задачи. В отличие от случайного леса, который использует жадный алгоритм для выбора оптимальной точки разделения, данный алгоритм выбирает точку разделения случайным образом, а также использует всю обучающую выборку для создания деревьев.

Таким образом, есть три основных гиперпараметра, которые нужно настроить в алгоритме: количество деревьев решений в ансамбле, количество входных функций, которые следует случайным образом выбрать и рассмотреть для каждой точки разделения и минимальное количество выборок, необходимых в узле для создания новой точки разделения. Случайный выбор точек разделения делает деревья решений в ансамбле менее коррелированными, хотя это увеличивает дисперсию алгоритма. Этому увеличению дисперсии можно противодействовать, увеличив количество деревьев, используемых в ансамбле.

Для реализации алгоритма Extremely Randomized Trees выбрана библиотека языка Python sklearn [8], предоставляющая класс sklearn.ensemble.ExtraTreesRegressor. Регрессор был обучен на наборе обучающих данных с целью выявления параметров, которые сильнее влияют на результат целевого показателя.

Модель машинного обучения представляет собой нейронную сеть. Данный подход позволяет не принимать во внимание второстепенные несистематические эффекты и процессы, которые происходят в системе Hadoop и не связаны с запущенной задачей. Нейросеть обучена с целью предсказания времени выполнения задачи по входным параметрам конфигурации Hadoop. Язык программирования нейронной сети – Python, используемая библиотека – Keras.

Рассматривается архитектура нейронной сети, представленная на рис. 1.3. Она содержит три скрытых полносвязных слоя с последовательным коэффициентом количества нейронов 3, 2, 1 от базового количества. Базовое количество нейронов скрытого слоя $base$ рассчитывается по формуле 1.6.

$$base = (features + targets + 100) * log_2(rows) \quad (1.6)$$

где: $features = 12$ – количество исследуемых параметров,

$targets = 1$ – количество целевых характеристик (время выполнения задачи),

$rows = 17000$ – количество строк в исходном наборе данных для обучения.

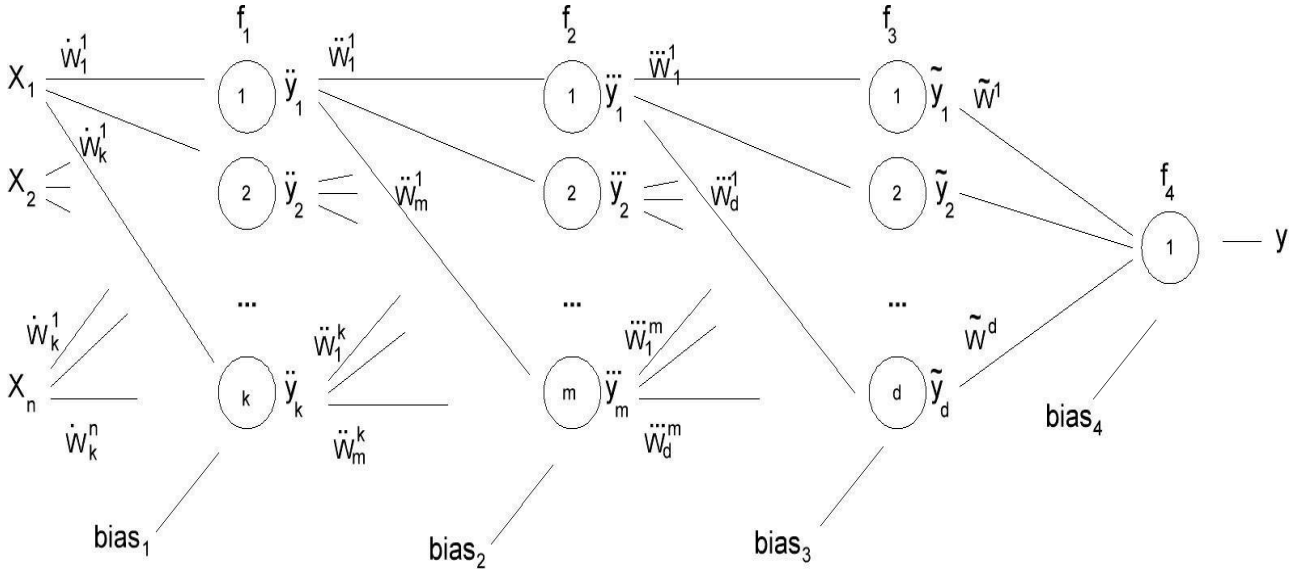

Рис. 1.3 – Архитектура нейронной сети

На рис. 1.3 входные параметры сети обозначены как $X_1..X_n$, где их количество $n = features$. Функции активации обозначены для каждого слоя $f_1..f_4$, которые определяем как функция Rectified Linear Unit [9]. Количество нейронов первого слоя – k, второго – m, третьего - d. Все соответствующие обучаемые параметры модели обозначены как $\dot{w}_1^1..\dot{w}_j^l$, $\ddot{w}_1^1..\ddot{w}_i^j$, $\dddot{w}_1^1..\dddot{w}_d^i$, $\tilde{w}..\tilde{w}$. Параметры смещения слоев обозначены как $bias_1..bias_4$.

В итоге математическая интерпретация построенной модели с учетом ее параметров w имеет вид, представленный формулой 7.

$$y = f_4\left(\sum_{r=1}^{d}(\tilde{w}\tilde{y}) + bias_4\right) =$$

$$= f_4\left(\sum_{r=1}^{d}\left(\tilde{w}\left[f_3\left(\sum_{i=1}^{m}(\ddot{w}^i\ddot{y}_i) + bias_3\right)\right]\right) + bias_4\right) =$$

$$= f_4\left(\sum_{r=1}^{d}\left(\tilde{w}\left[f_3\left(\sum_{i=1}^{m}\left(\ddot{w}^i\left[f_2\left(\sum_{j=1}^{k}(\ddot{w}^j\ddot{y}_j) + bias_2\right)\right]\right) + bias_3\right)\right]\right) + bias_4\right) =$$

(1.7)

$$= f_4\left(\sum_{r=1}^{d}\left(\tilde{w}\left[f_3\left(\sum_{i=1}^{m}\left(\ddot{w}^i\left[f_2\left(\sum_{j=1}^{k}\left(\ddot{w}^j f_1\left(\sum_{l=1}^{n}(\dot{w}^l x_l) + bias_1\right)\right) + bias_2\right)\right]\right) + bias_3\right)\right]\right) + bias_4\right)$$

После обучения нейронной сети на предоставленном наборе данных получили модель для предсказания времени выполнения Hadoop задачи по входному набору конфигурационных параметров.

1.6 Алгоритм определения оптимальных параметров конфигурации

Для поиска входных параметров, при которых значение целевой метрики будет минимально, полученная функция f_4 минимизируется относительно параметров $X_1..X_n$. с учетом всех обученных параметров модели $\dot{w}^1_1..\dot{w}^l_j$, $\ddot{w}^1_1..\ddot{w}^j_i$, $\ddot{w}^1_1..\ddot{w}^i_d$, $\tilde{w}..\tilde{w}$.

Для минимизации функции нейронной сети f_4 в условиях ограничений выбран алгоритм оптимизации последовательного квадратичного программирования SQP [10]. В текущих условиях некоторые входные параметры модели машинного обучения нельзя изменять, например, размер обрабатываемой таблицы. Поэтому в ограничениях следует задать для этого параметра одинаковое минимальное и максимальное значение, которое получено из API перед исполнением задачи.

Идеей алгоритма является последовательное решение задач квадратичного программирования, аппроксимирующих заданную оптимизацию. Решение в условиях ограничений сводится с методу множителей Лагранжа. Выбор алгоритма обусловлен тем, что квадратичные приближения гораздо точнее аппроксимируют исходную нелинейную функцию, что приводит к большей скорости сходимости по сравнению с линейными методами. Программная реализация этого метода также предоставляется пакетом Scipy.

В итоге алгоритм предоставляет оптимальный набор конфигурационных параметров Hadoop $p_0^*, .., p_n^*$ в условиях ограниченности ресурсов, задающихся непосредственно разработчиком, либо администратором перед встраиванием алгоритма в процесс запуска задач.

1.7 Внедрение в Oozie Workflow

Так как разработанный программный комплекс используется для настройки задачи, повторяющейся во времени, была выбрана система для планирования выполнения заданий в Hadoop - Oozie.

Для задач, которые должны быть выполнены последовательно, используется построение графа запуска с помощью Oozie Workflow. Это реализуется в виде ориентированных ациклических графов, задающих последовательность выполняемых действий и их правил.

С помощью Oozie Workflow разработаны workflow.xml и sub-workflow.xml файлы, последовательность действий которых представлены на рис. 1.4.

Файл workflow.xml состоит из этапов:

1. Запуск файла algoritm.py, который рассчитывает оптимальные параметры запуска задачи в Hadoop $p^*, \ldots p^*_n$ путем минимизации функции модели машинного обучения. Полученные значения передаются на следующий этап. Веса обученной нейронной сети считываются из файла weights.npy;
2. Запуск рассматриваемой задачи в системе Hadoop с полученными оптимальными значениями;
3. Запуск действий файла sub-workflow.xml. Необходимость выполнения данного этапа регулируется разработчиком.

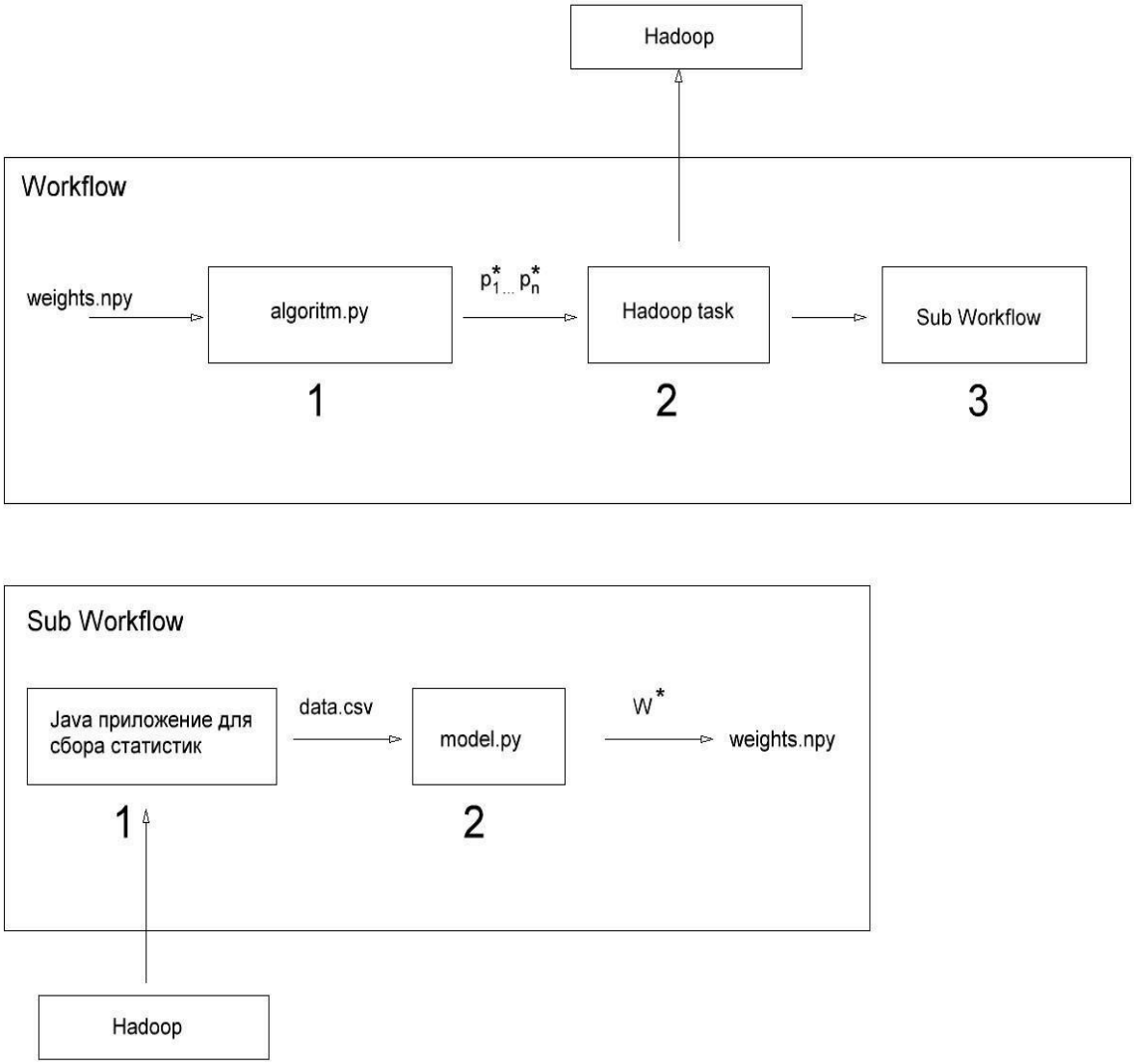

Рис. 1.4 – Последовательность выполнения задач в Oozie Worflow

Файл sub-workflow.xml состоит из этапов:

1. Запуск java-приложения для сбора статистик ранее запущенных задач за определенное время и генерации обучающего набора данных;
2. Запуск файла model.py, который содержит модель машинного обучения, заданной архитектуры, способную обучаться на предоставленном наборе данных для предсказания времени выполнения задач в Hadoop;
3. Запись обученных параметров w^* в файл weights.npy, для дальнейшего считывания алгоритмом расчета оптимальных параметров.

Таким образом, предоставляя планировщику задач Oozie Coordinator файл последовательности sub-workflow.xml на выполнение его с заданной периодичностью, реализован постоянный автоматический сбор статистики и поддержка актуальности полученных оптимальных параметров с течением времени. Это обуславливается тем, что модель машинного обучения переобучается на более актуальных данных при желании разработчика. Тем самым достигаются все требования поставленной задачи.

2. ПРАКТИЧЕСКАЯ ЧАСТЬ

2.1 Сбор данных для обучения

Демонстрация работоспособности системы приведена на наборе данных, загруженных в базу данных Hbase. В ней были созданы 250 таблиц, которые хранят данные различного объема, а также искусственно разбиты на различное количество регионов.

Для проведения вычислительных экспериментов был собран набор данных для обучения, содержащий 12 параметров, представленных в таблице 2.1 с их диапазоном значений. Количество строк набора 17000.

Таблица 2.1 – Параметры для обучения нейронной сети

Параметр	min	max	Описание
mapreduce.task.io.sort.factor	200	256	Количество потоков для одновременного объединения при сортировке файлов
mapreduce.task.io.sort.mb	200	256	Объем буферной памяти при сортировке файлов
max.reducers	2	10	Максимальное количество задач reduce
max.mappers	2	10	Максимальное количество задач map
mapreduce.map.cpu.vcores	1	4	Количество ядер на задачу map
mapreduce.reduce.cpu.vcores	1	4	Количество ядер на задачу reduce
mapreduce.map.memory.mb	512	1024	Максимальное количество памяти на задачу map

Продолжение таблицы 2.1

Параметр	min	max	Описание
mapreduce.reduce.memory.mb	512	1024	Максимальное количество памяти на задачу reduce
mapper.java.option	512	1024	Размер области heap java для задачи map
reducer.java.option	512	1024	Размер области heap java для задачи reduce
table.size	654	2234021317	Размер обрабатываемой таблицы
table.regions	1	1000	Количество регионов обрабатываемой таблицы hbase

Задаче на вход поступают все строки из двух таблиц базы данных Hbase. На этапе отображения каждой подзадаче поступает на вход строка таблицы, а также название самой таблицы. Происходит считывание идентификатора записи для дальнейшего определения на какой процесс reduce отправить текущую запись. На этап reduce поступают все строки по соответствующим идентификаторам. Таким образом, записей будет либо 1, либо 2. Далее происходит рекурсивное сравнение всех полей между записями с одинаковыми идентификаторами. Все различные поля записываются в файловую систему HDFS. Изначальная задача заканчивается, когда будут сравнены все строки двух таблиц. Для простоты анализа на вход каждой запускаемой задачи поступала одна таблица, которая сверялась сама с собой.

Использование алгоритма на основе дерева решений Extremely Randomized Trees определило важные входные параметры, представленные на рис. 2.1.

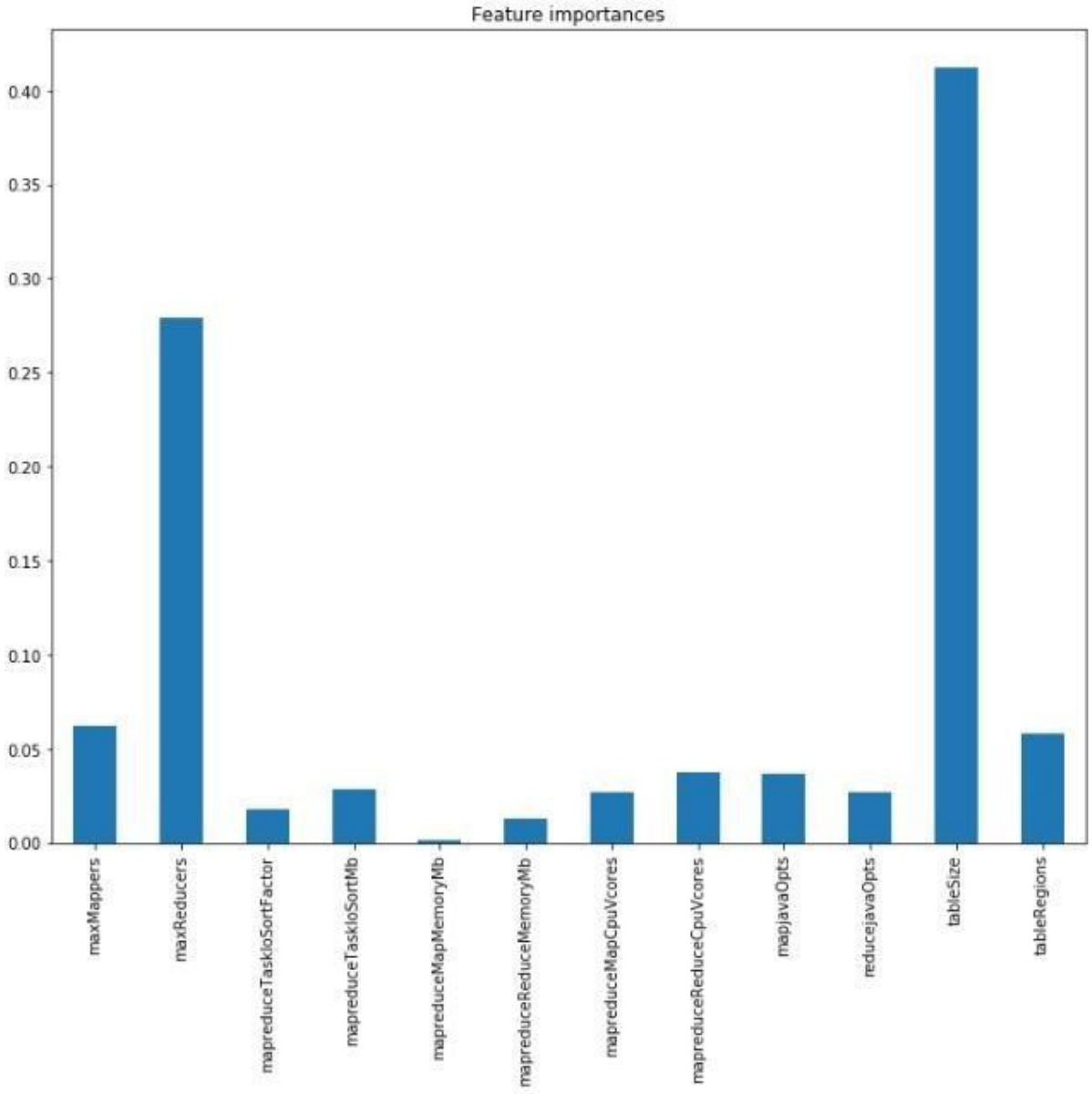

Рис. 2.1 – Важность входных параметров нейронной сети

Таким образом наиболее значимыми изменяемыми параметрами Hadoop среди полученного объема данных для обучения оказались максимальное количество задач отображения (map) и количество задач свертки (reduce). Параметры, связанные с количеством памяти, оказались менее важны, что объясняется достаточностью минимального значения памяти для запускаемой задачи. Вторыми по значимости параметрами являются количество виртуальных процессоров, выделяемых на этапы отображения и свертки. Стоит отметить, что параметр tableRegions можно менять путем запуска

определенного процесса для изменения количества регионов таблицы Hbase. При этом можно добиться улучшения производительности, но правка этого параметра выходит за рамки текущих исследований.

2.2 Обучение нейронной сети

Файл model.py содержит описание нейронной сети и процесс ее обучения. Первым этапом обучения является очистка данных, вследствие чего были убраны аномальные “выбросы”, одинаковые записи. Все остальные строки представляют собой разделенные запятой числовые значения, без пропусков, которые были нормализованы.

На рис. 2.2 представлены зависимости времени выполнения задачи от ключевых входных параметров. Видно, что увеличение количества задач отображения и свертки ускоряет время выполнения запускаемой в Hadoop задачи. Очевидной закономерностью является то, что с увеличением объема таблицы увеличивается время ее обработки. Но увеличение количества регионов таблицы влияет неоднозначно, что объясняется тем, что в обрабатываемых таблицах количество регионов устанавливается больше для объемных таблиц в целях исключения возникновения ошибок задержки получения информации.

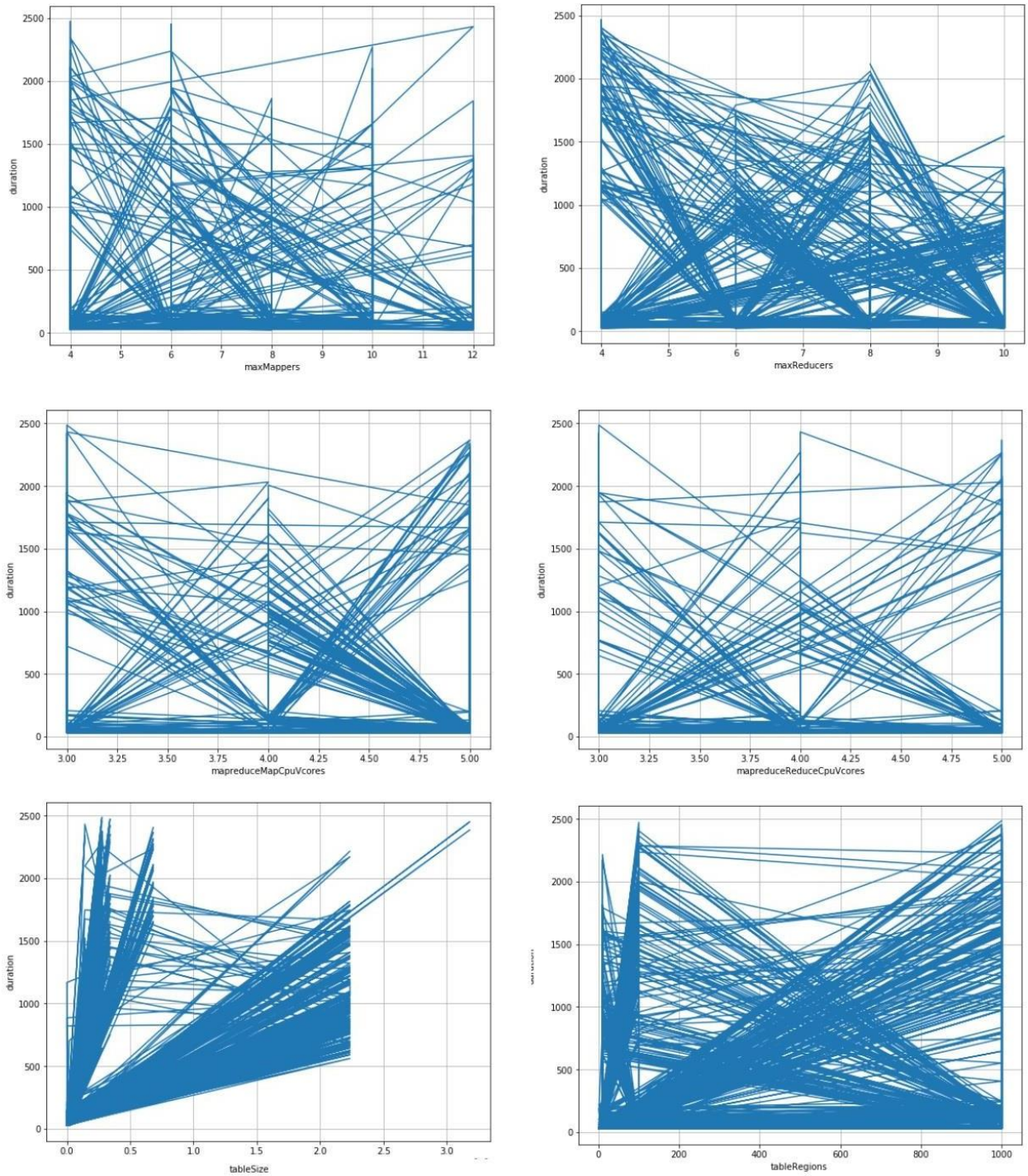

Рис. 2.2 – Длительность выполнения задачи в зависимости от входных параметров: а) от максимального количества задач отображения, б) от количества задач свертки, в) от размера таблицы, г) от количества разбиений (регионов) таблицы

Нейросеть имеет структуру, представленную на рис. 2.3, использует оптимизатор Adam [11] и функцию активации Relu. Обучается на наборе данных, составляющим 60% от полученного набора. Остальная часть распределена поровну между тестовой и валидационной выборкой.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 6872)              89336
_________________________________________________________________
dense_2 (Dense)              (None, 4581)              31485213
_________________________________________________________________
dense_3 (Dense)              (None, 2290)              10492780
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 2291
=================================================================
Total params: 42,069,620
Trainable params: 42,069,620
Non-trainable params: 0
_________________________________________________________________
```

Рис. 2.3 – Нейронная сеть в программном виде

Значения функции потерь, являющейся среднеквадратической ошибкой, во время обучения представлена на рис. 2.4.

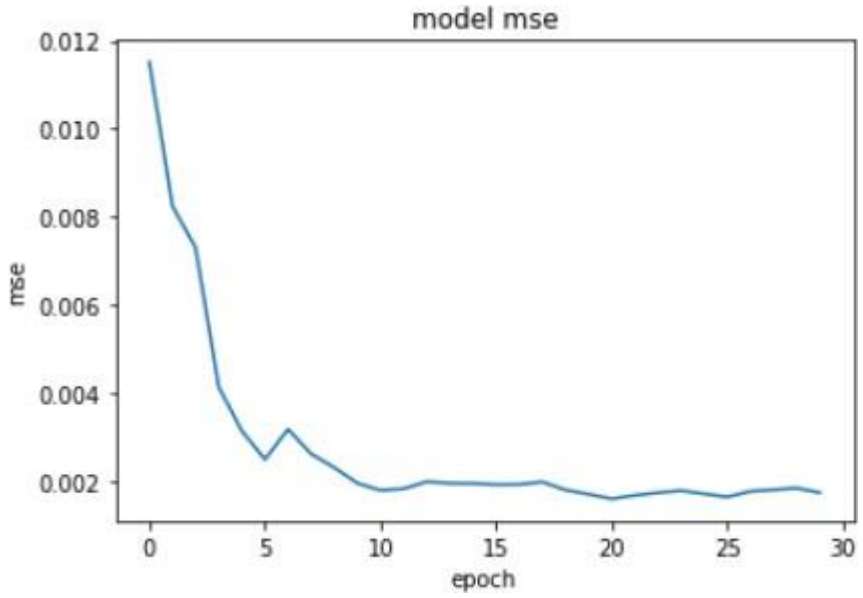

Рис. 2.4 – Среднеквадратическая ошибка во время обучения модели

Значения ошибок на тестовой выборке: среднеквадратическая ошибка 0.00134, среднеквадратическое отклонение 0.03658, средняя абсолютная ошибка 0.01205. Пример сравнения данных из исходного набора данных с результатом предсказания нейронной сети представлен на рис. 2.5.

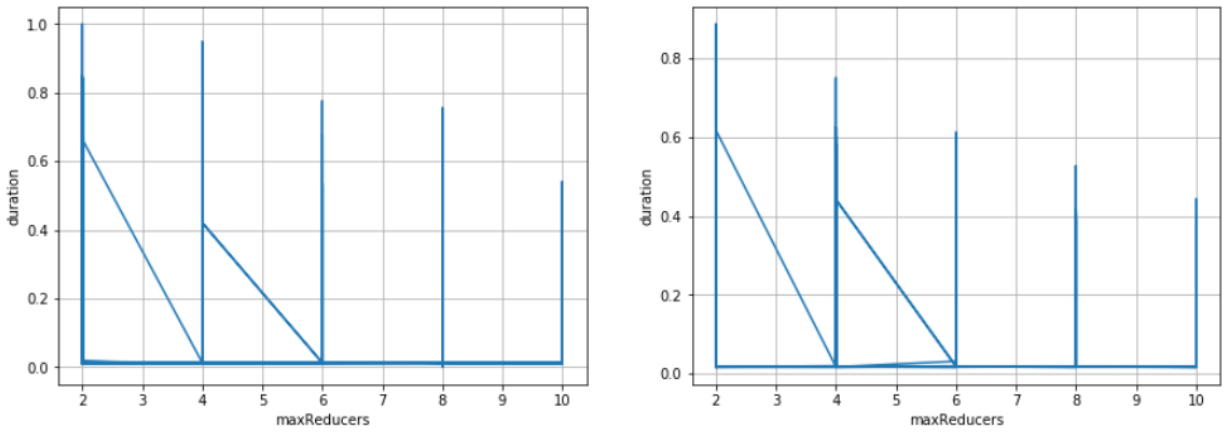

Рис. 2.5 – Сравнение длительности выполнения задачи для тестовой выборки исходного набора данных и предсказанных моделью значений от количества задач свертки

Веса нейронной сети для каждого слоя со значениями смещения сохраняются в отдельный файл weights.npy, который считывается файлом algoritm.py для расчета оптимальных значений.

2.3 Реализация алгоритма определения оптимальных параметров

Алгоритм определения оптимальных параметров, описанный в файле algoritm.py, использует модуль scipy.optimize для минимизации функции нейронной сети, представленной формулой 2.1.

$$F_{model} = relu((relu((relu((relu(x \cdot w1 + bias1)) \cdot w2 + bias2)) \cdot w3 + bias3)) \cdot w4 + bias4) \quad (2.1)$$

где: $relu$ – функция Rectified Linear Unit,

x – вектор начальных входных параметров

$w1..w4$ – матрицы весов соответствующих слоев нейронной сети

$bias1..bias4$ – векторы смещений соответствующих слоев нейронной сети

Для каждого параметра Hadoop определяются значения ограничений: минимальное и максимальное значение. Пример таких ограничений для таблицы размером 500000б и количеством регионов 200 представлен в листинге 2.1.

Листинг 2.1 – Пример ограничений входных параметров

```
min_values = {'maxMappers': [2],\
              'maxReducers': [2], \
              'mapreduceTaskIoSortFactor': [10], \
              'mapreduceTaskIoSortMb': [200], \
              'mapreduceMapMemoryMb': [512],\
              'mapreduceReduceMemoryMb': [512], \
              'mapreduceMapCpuVcores': [1], \
              'mapreduceReduceCpuVcores': [1], \
              'mapjavaOpts': [512],\
              'reducejavaOpts': [512], \
              'tableSize': [500000], \
              'tableRegions': [200] \
             }
max_values = {'maxMappers': [15],\
              'maxReducers': [15], \
              'mapreduceTaskIoSortFactor': [500], \
              'mapreduceTaskIoSortMb': [512], \
              'mapreduceMapMemoryMb': [2048],\
              'mapreduceReduceMemoryMb': [2048], \
              'mapreduceMapCpuVcores': [5], \
              'mapreduceReduceCpuVcores': [5], \
              'mapjavaOpts': [1024],\
              'reducejavaOpts': [1024], \
              'tableSize': [500000], \
              'tableRegions': [200] \
             }
```

Результат минимизации функции при таких ограничениях - 36,1с. Для проверки работоспособности алгоритма проведена проверка на тестовом наборе данных, а именно сравнение длительности выполнения задач без оптимизации с длительностью выполнения предложенной алгоритмом. Результаты представлены на рис. 2.6.

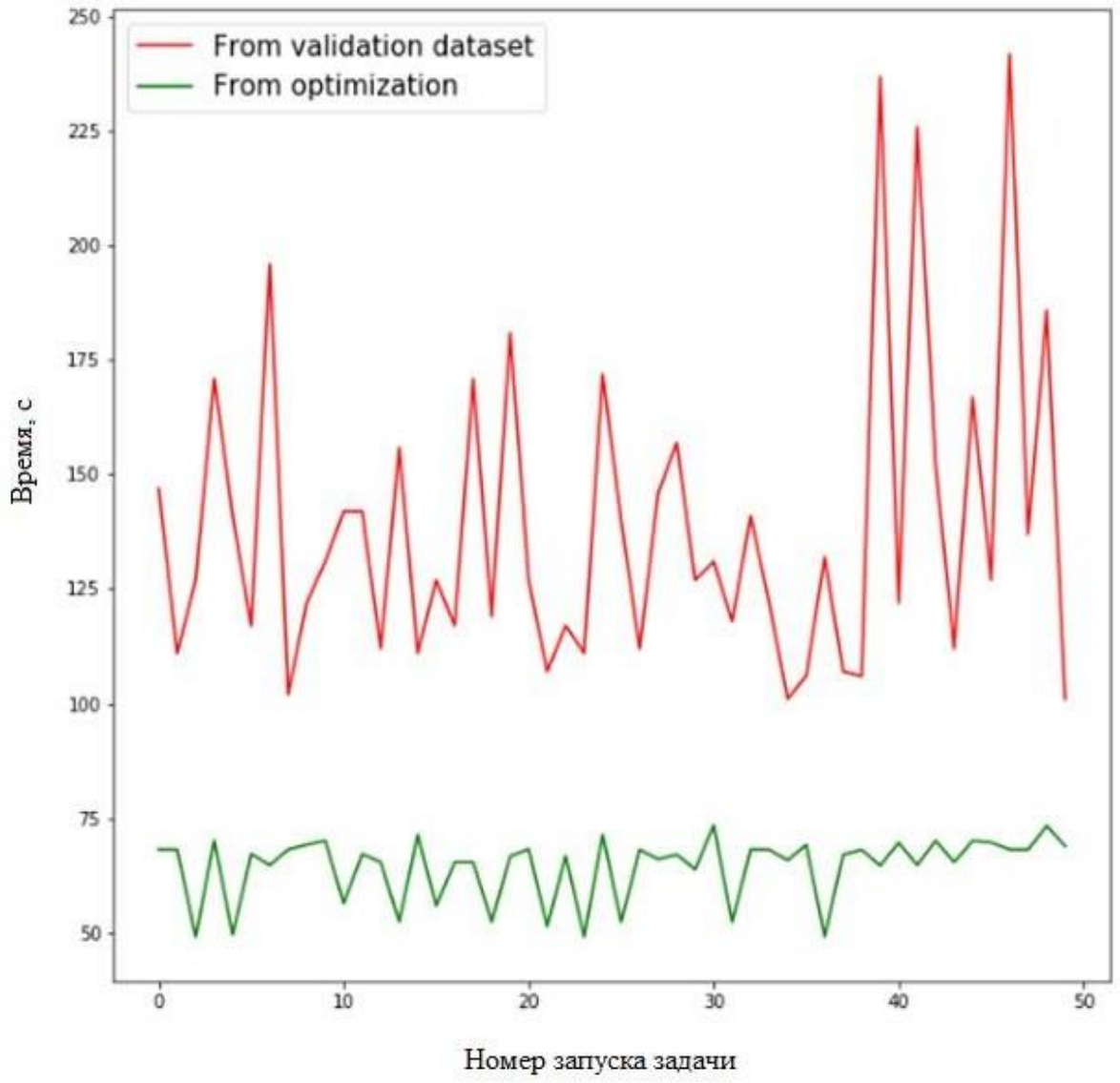

Рис. 2.6 – Сравнение длительности выполнения задачи для тестовой выборки исходного набора данных с длительностью после оптимизации параметров для нескольких запусков

Из полученного графика видно, что алгоритм предлагает параметры Hadoop, которые ускоряют выполнение задачи по сравнению с запусками с неоптимальными параметрами.

Для заключительного тестирования подсчитано медианное значение ускорения выполнения задачи в системе Hadoop между запуском с параметрами из тестовой выборки собранного набора данных и

оптимизированными параметрами для соответствующих таблиц. Медианное значение ускорения составило ~ 52.4%.

2.4 Реализация внедрения в Oozie Workflow

Для запуска задач в автоматическом режиме была разработана последовательность этапов, которые необходимы для запуска регулярной задачи в системе Hadoop на основе планировщика Oozie. Последовательность описана в двух xml файлах: workflow.xml и sub-worflow.xml, представленных на рис. 1.4. Листинг 2.2 отражает действия файла workflow.xml.

Листинг 2.2 – Содержимое файла workflow.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<workflow-app xmlns="uri:oozie:workflow:0.4" name="Run application workflow">
    <start to="algoritm" />
    <action name="algoritm">
        <shell xmlns="uri:oozie:shell-action:0.1">
            <job-tracker>${clusterJobtracker}</job-tracker>
            <name-node>${clusterNamenode}</name-node>
            <exec>python</exec>
                <file>algoritm.py</file>
                <arg>${pathToWeights}</arg>
                <arg>${tableSize}</arg>
                <arg>${tableRegions}</arg>
            <capture-output />
        </shell>
        <ok to="JavaAction" />
        <error to="fail" />
    </action>
    <action name="JavaAction">
        <java>
            <main-class>${mainClass}</main-class>
                <arg>${wf:actionData('algoritm')['maxMappers']}</arg>
                <arg>${wf:actionData('algoritm')['maxReducers']}</arg>
                <arg>${wf:actionData('algoritm')['mapreduceTaskIoSortFactor']}</arg>
                <arg>${wf:actionData('algoritm')['mapreduceTaskIoSortMb']}</arg>
                <arg>${wf:actionData('algoritm')['mapreduceMapMemoryMb']}</arg>
                <arg>${wf:actionData('algoritm')['mapreduceReduceMemoryMb']}</arg>
                <arg>${wf:actionData('algoritm')['mapreduceMapCpuVcores']}</arg>
                <arg>${wf:actionData('algoritm')['mapreduceReduceCpuVcores']}</arg>
                <arg>${wf:actionData('algoritm')['mapjavaOpts']}</arg>
                <arg>${wf:actionData('algoritm')['reducejavaOpts']}</arg>
            <capture-output />
        </java>
        <ok to="end" />
        <error to="fail" />
    </action>
     <decision>
          <switch>
               <case to=”sub-workflow”>
                    ${runSubWorkflow}
               </case>
               <default to=”end”/>
          </switch>
     </decision>
     <action name="sub-workflow">
        <sub-workflow>
            <app-path>${subWorkflowAppPath}</app-path>
            <configuration>
                <property>
                    <name>dateFrom</name>
                    <value>${coord:formatTime(coord:dateOffset(coord:nominalTime(), 0,
                            ‘DAY’), “yyyyMMdd”)}</value>
                </property>
            </configuration>
        </sub-workflow>
        <ok to="end"/>
        <error to="kill"/>
    </action>

```

```
59:       <kill name="fail">
60:           <message>Job failed, error
61:               message[${wf:errorMessage(wf:lastErrorNode())}]</message>
62:       </kill>
63:       <end name="end" />
64:   </workflow-app>
```

Процесс начинается с этапа algoritm, который отвечает за расчет оптимальных параметров задачи путем запуска файла algoritm.py. На вход этот этап получает путь до файла, хранящего весовые значения функции нейронной сети, объем обрабатываемой таблицы, количество регионов таблицы. Результатом работы является запись в стандартный поток вывода значений оптимальных параметров настройки запускаемой задачи.

Следующим этапом является запуск основного java приложения, которое необходимо выполнить в Hadoop. На вход оно получает все оптимальные настойки, рассчитанные на предыдущем шаге, а также основной запускаемый класс. Параметры, с которыми запускается задача, логируются стандартными средствами с целью их дальнейшего получения приложением для сбора статистики.

В зависимости от переданного oozie флага runSubWorkflow будет запущен дочерний процесс sub_workflow.xml, представленный листингом 2.3. На вход он получает значение даты, начиная с которой необходимо осуществлять сбор статистик для генерации csv файла с обучающим набором данных модели машинного обучения. В случае любых ошибок процесс заканчивается этапом fail.

Листинг 2.3 – Содержимое файла sub_workflow.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<workflow-app xmlns="uri:oozie:workflow:0.4" name="Run learning model sub workflow">
    <start to="algoritm" />
    <action name="JavaGenerateData">
        <java>
            <main-class>${mainClass}</main-class>
                <arg>${pathToDataset}</arg>
                <arg>${dateFrom}</arg>
            <capture-output />
        </java>
        <ok to="model" />
        <error to="fail" />
    </action>

     <action name="model">
        <shell xmlns="uri:oozie:shell-action:0.1">
            <job-tracker>${clusterJobtracker}</job-tracker>
            <name-node>${clusterNamenode}</name-node>
            <exec>python</exec>
                <file>model.py</file>
                <arg>${pathToDataset}</arg>
                <arg>${pathToWeights}</arg>
            <capture-output />
        </shell>
        <ok to="end" />
        <error to="fail" />
    </action>

    <kill name="fail">
        <message>Job failed, error
            message[${wf:errorMessage(wf:lastErrorNode())}]</message>
    </kill>
    <end name="end" />
</workflow-app>
```

Процесс sub_workflow.xml начинается с запуска java приложения для сбора статистики, которое получает на вход дату, с которой осуществляется поиск запусков, путь до csv файла, в который необходимо записывать данные обучения нейронной сети.

Следующим шагом является запуск этапа model. Этап получает на вход путь до данных для обучения, а также путь до файла, в которых необходимо записать весовые коэффициенты обученной модели. Этот шаг запускает файл model.py, который переинициализирует нейронную сеть и обучает ее предсказывать длительность исполнения задачи. В итоге имеем файл с обновленными весами для дальнейшего более точного определения

оптимальных параметров. В случае любых ошибок процесс заканчивается этапом fail.

В итоге нейронная сеть встроена в процесс запуска map-reduce задачи с целью обучения и предоставления своих обученных параметров алгоритму, который рассчитывает по ним оптимальные параметры изначально запускаемой задачи. Так все компоненты работают согласованно, последовательно, это дает возможность не производить дополнительные действия разработчикам для периодичного запуска задач обработки данных в Hadoop.

3. ЗАКЛЮЧЕНИЕ

В работе представлен подход к построению программного обеспечения на языках Java и Python, позволяющего производить автоматическую оптимальную настройку системы Hadoop под запускаемые на ней задачи на основе нейросетевого моделирования. Были разработаны компоненты системы для сбора данных, для обучения построенной нейронной сети, для оптимизации параметров с учетом их граничных значений. Также разработан автоматизированный процесс, позволяющий производить расчет и подстановку оптимальных параметров перед запуском самой задачи.

3.1 Оценка и достоверность результатов

Результат проверки оптимизированных параметров показывает ускорение выполнения тестовой map-reduce задачи на 52.4%. При этом важнейшими параметрами являются максимальное количество задач отображения (map) и количество задач свертки (reduce). Основным заключением является то, что представленный подход позволяет ускорить работу обработки данных за счет правильной настройки параметров запуска, но само значение ускорения зависит от множества факторов. Например, собранный набор данных имеет малую долю записей для действительно больших таблиц, на которых разумнее проводить эксперименты. Также для сбора эффективных данных требуется большое количество времени.

Особое внимание следует уделить анализу параметров обучающей выборки. Изначально набор параметров был выбран эмпирическим путем, но для максимальной эффективности подхода необходимо найти ряд параметров, оказывающие большее влияние на целевую метрику. Для поиска таких параметров следует обратиться к руководствам администратора Hadoop.

Также можно добиться еще большего уменьшения времени выполнения задачи путем определения ограничивающих значений. Например, в проведенном эксперименте было выбрано достаточно малое возможное количество задач отображения и свертки.

Таким образом, для встраивания разработанного комплекса в рабочий процесс разработчикам понадобится определить перечень изменяемых параметров Hadoop, а также их граничные значения. После этого нужно сделать ряд запусков для сбора обучающих данных, которые будут в дальнейшем автоматически использоваться для обучения нейронной сети.

Основным недостатком разработанного комплекса является то, что обученные веса нейронной сети будут бесполезны в случае полного изменения задачи, запускаемой в Hadoop. То есть данный подход применим для приложений, которые запускаются периодично без изменений логики обработки данных на различных объемах, либо с изменением незначительной части логических операций.

3.2 Дальнейшее развитие

В рамках дальнейших исследований предлагается включить в процесс запуска задач с оптимальными параметрами шаг, при котором будет производится расчет параметров для выполнения за указанное количество времени. Например, в случае запуска задачи на объемах данных, поступивших в систему за день, возможна ситуация, когда объем информации больше обычного, поэтому требуется увеличить ресурсы для выполнения задачи за обычное время. Найденные значения параметров должны бать предложены для подтверждения разработчику (даже при их выходе за установленные границы), вследствие чего принимается решение об их подстановке в Hadoop.

Предлагается провести эксперименты для различного типа задач, так как эксперименты проводились только для map-reduce задачи, имеющую шаг отображения, перемешивания, свертки.

Для исправления основного недостатка, заключающегося в привязанности нейронной сети к конкретной задаче, предлагается добавить параметры, характеризующие запускаемую задачу, чтобы нейросеть научилась их “классифицировать”. Результатом такой идеи будет универсальная модель машинного обучения, способная предсказывать длительность исполнения разного рода задач, на которых был собран обучающий набор данных. Имея достаточное количество задач различных типов, станет возможным анализировать время исполнения новых поступающих задач, которые имеют ряд особенностей, присущих ранее только единственному типу задач.

Учитывая большое влияние параметра, определяющего количество регионов, на которые разбита таблица, становится целесообразно изменять это значение. Его изменение возможно производить перед запуском задачи с помощью использования Hbase API. Полученное значение можно использовать на постоянной основе для соответствующих таблиц, чтобы минимизировать ошибки, связанные с задержкой получения информации из них при многочисленных подключениях.

Наконец, необходимо провести сравнение между подходом нейросетевого моделирования и стандартными методами машинного обучения в контексте поставленной задачи.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Herodotos Herodotou Yuxing Chen Jiaheng Lu. A Survey on Automatic Parameter Tuning for Big Data Processing Systems // ACM Computing Surveys. April 2020.

[2] Á.B. Hernández, M.S. Perez, S. Gupta, V. Muntés-Mulero, Using machine learning to optimize parallelism in big data applications, Future Generation Computer Systems (2017), http://dx.doi.org/10.1016/j.future.2017.07.003

[3] Bao Liang, Liu Xin, Chen Weizhao. Learning-based Automatic Parameter Tuning for Big Data Analytics Frameworks // CoRR. 2018. Ò. abs/1808.06008. URL: http://arxiv.org/abs/1808.06008.

[4] Zhang Pengcheng, Cao Wennan, Muccini Henry. Quality Assurance Technologies of Big Data Applications: A Systematic Literature Review // CoRR. 2020. Ò. abs/2002.01759. URL: https://arxiv.org/abs/2002.01759.

[5] When Machine Learning Meets Big Data: A Wireless Communication Perspective / Yuanwei Liu, Suzhi Bi, Zhiyuan Shi [è äð.] // CoRR. 2019. Ò. abs/1901.08329. URL: http://arxiv.org/abs/1901.08329.

[6] Nie Jiawang, Wang Li, Ye Jane [è äð.]. A Lagrange Multiplier Expression Method for Bilevel Polynomial Optimization. 2020.

[7] Konstantinov Andrei V., Utkin Lev V. Gradient boosting machine with partially randomized decision trees // CoRR. 2020. Ò. abs/2006.11014. URL: https://arxiv.org/abs/2006.11014.

[8] Scikit-learn: Machine Learning in Python / Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort [è äð.] // CoRR. 2012. Ò. abs/1201.0490. URL: http://arxiv.org/abs/1201.0490.

[9] Agarap Abien Fred. Deep Learning using Rectied Linear Units (ReLU) // CoRR. 2018. Ò. abs/1803.08375. URL: http://arxiv.org/abs/1803.08375.

[10] Pelley Carwyn, Innocente Mauro Sebasti_an, Sienz Johann. Combining Particle Swarm Optimizer with SQP Local Search for Constrained Optimization

Problems // CoRR. 2021. Ò. abs/2101.10936. URL: https://arxiv.org/abs/2101.10936.

[11] Balles Lukas, Hennig Philipp. Follow the Signs for Robust Stochastic Optimization // CoRR. 2017. Ò. abs/1705.07774. URL: http://arxiv.org/abs/1705.07774.

[12] Бобряков А.С., Нейросетевое моделирование для автоматической настройки задач в системах обработки больших данных // XLVII Гагаринские чтения // Москва 2021

ПРИЛОЖЕНИЯ

Приложение 1

Перечень некоторых параметров hadoop

mapreduce.job.committer.setup.cleanup.needed	Количество потоков для слияния одновременно при сортировке файлов. Определяет количество открытых дескрипторов файлов.
mapreduce.task.io.sort.factor	Общий объем буферной памяти, используемой при сортировке файлов, в мегабайтах. По умолчанию каждому потоку слияния присваивается 1 Мбайт, что должно минимизировать поиск.
mapreduce.task.io.sort.mb	Мягкий предел в буфере сериализации. После достижения, поток начнет разливать содержимое на диск в фоновом режиме.
mapreduce.map.sort.spill.percent	Позволяет потоку монитора задач следить за потреблением одного диска заданиями. Установив это значение, задача завершится неудачей, если оно будет достигнуто ~~значение~~.

mapreduce.job.local-fs.single-disk-limit.bytes	Предел дискового пространства.
mapreduce.job.dfs.storage.capacity.kill-limit-exceed	
mapreduce.job.maps	Количество задач map по умолчанию для каждого задания.
mapreduce.job.reduces	Количество задач reduce по умолчанию для каждого задания. Обычно устанавливается значение 99% емкости сокращения кластера, так что в случае сбоя узла сокращение все еще может быть выполнено в одном reducer.
mapreduce.job.running.map.limit	Максимальное количество одновременных задач map на задание.
mapreduce.job.running.reduce.limit	Максимальное количество одновременных задач reduce на задание.
mapreduce.job.max.map	Ограничивает количество задач map, разрешенных для каждого задания. Нет предела, если это значение отрицательно.
mapreduce.job.reducer.preempt.delay.sec	Порог (в секундах), после которого неудовлетворенный

	запрос map запускает упреждение редуктора.
mapreduce.job.reducer.unconditional-preempt.delay.sec	Порог (в секундах), после которого неудовлетворенный запрос map запускает принудительное упреждение редуктора независимо от ожидаемого запаса
mapreduce.job.max.split.locations	Максимальное количество местоположений блоков для хранения для каждого разделения для расчета локальности.
mapreduce.job.split.metainfo.maxsize	Максимально допустимый размер файла для разделения.
mapreduce.reduce.maxattempts	Максимальное количество попыток на задачу reduce.
mapreduce.reduce.shuffle.fetch.retry.timeout-ms	Значение таймаута в мс для задач, чтобы повторить попытку извлечения снова, в случае, когда происходит какой-то сбой.
mapreduce.reduce.shuffle.parallelcopies	Количество параллельных передач по умолчанию, выполняемых reduce во время фазы копирования (shuffle).
mapreduce.reduce.shuffle.connect.timeout	Максимальное количество времени для попыток

	подключиться к удаленному узлу.
mapreduce.shuffle.listen.queue.size	Длина очереди прослушивания сервера.
mapreduce.map.memory.mb	Объем памяти, запрашиваемый планировщиком для каждой задачи map. Если это не указано или не является положительным, то оно выводится из mapreduce.map.java.opts и mapreduce.job.heap.memory-mb.ratio. Если java-opts также не заданы, мы устанавливаем его равным 1024.
mapreduce.map.cpu.vcores	Количество виртуальных ядер для запроса от планировщика для каждой задачи map.
mapreduce.reduce.memory.mb	Объем памяти для запроса от планировщика для каждой задачи reduce. Если это не указано или не является положительным, то оно выводится из mapreduce.reduce.java.opts и mapreduce.job.heap.memory-mb.ratio. Если java-opts также

	не заданы, мы устанавливаем его равным 1024.
mapreduce.reduce.cpu.vcores	Количество виртуальных ядер, запрашиваемых планировщиком для каждой задачи reduce.

Приложение 2

Пример http ответа от Yarn на запрос данных о запущенных задачах


```
{
 "apps":
 {
  "app":
  [
   {
    "id": "application_1476912658570_0002",
    "user": "user2",
    "name": "word count",
    "queue": "default",
    "state": "FINISHED",
    "finalStatus": "SUCCEEDED",
    "progress": 100,
    "trackingUI": "History",
    "trackingUrl": "http://host.domain.com:8088/cluster/app/application_1476912658570_0002",
    "diagnostics": "...",
    "clusterId": 1476912658570,
    "applicationType": "MAPREDUCE",
    "applicationTags": "",
    "priority": -1,
    "startedTime": 1476913457320,
    "finishedTime": 1476913761898,
    "elapsedTime": 304578,
    "amContainerLogs":
"http://host.domain.com:8042/node/containerlogs/container_1476912658570_0002_02_000001/user2",
    "amHostHttpAddress": "host.domain.com:8042",
    "allocatedMB": 0,
    "allocatedVCores": 0,
    "runningContainers": 0,
    "memorySeconds": 206464,
    "vcoreSeconds": 201,
    "queueUsagePercentage": 0,
    "clusterUsagePercentage": 0,
    "preemptedResourceMB": 0,
    "preemptedResourceVCores": 0,
    "numNonAMContainerPreempted": 0,
    "numAMContainerPreempted": 0,
    "logAggregationStatus": "DISABLED",
    "unmanagedApplication": false,
    "appNodeLabelExpression": "",
    "amNodeLabelExpression": "",
    "resourceRequests": [
    {
       "capability": {
          "memory": 4096,
          "virtualCores": 1
       },
       "nodeLabelExpression": "",
       "numContainers": 0,
       "priority": {
          "priority": 0
       },
       "relaxLocality": true,
       "resourceName": "*"
    },
```

```
        {
          "capability": {
            "memory": 4096,
            "virtualCores": 1
          },
          "nodeLabelExpression": "",
          "numContainers": 0,
          "priority": {
            "priority": 20
          },
          "relaxLocality": true,
          "resourceName": "host1.domain.com"
        },
        {
          "capability": {
            "memory": 4096,
            "virtualCores": 1
          },
          "nodeLabelExpression": "",
          "numContainers": 0,
          "priority": {
            "priority": 20
          },
          "relaxLocality": true,
          "resourceName": "host2.domain.com"
        }
      ]
    }
  }
```