МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
имени М.В.ЛОМОНОСОВА
НАУЧНО-ИССЛЕДОВАТЕЛЬСКИЙ ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР

На правах рукописи

Шайхисламов Денис Ильгизович

Исследование и разработка методов для сравнительного анализа суперкомпьютерных приложений на основе технологий интеллектуального анализа данных

Специальность 2.3.5.
Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей

Диссертация на соискание учёной степени
кандидата физико-математических наук

Научный руководитель:
кандидат физико-математических наук
Воеводин Вадим Владимирович

Москва — 2025

Оглавление

Стр.

Введение

Суперкомпьютеры являются важным инструментом для решения научных и прикладных задач в различных областях, таких как физика, геология, метеорология, химия, медицина и многих других. Использование суперкомпьютеров необходимо из-за вычислительной сложности некоторых задач, решение которых на обычных компьютерах может занять дни, месяцы или даже годы. Однако эффективно использовать суперкомпьютерные системы очень непросто как из-за чрезвычайно сложной архитектуры этих систем, так и из-за сложности создания высокопроизводительных параллельных программ. Для достижения высокой производительности требуются специализированные знания и опыт в разработке эффективных приложений для суперкомпьютеров, которыми зачастую не обладают пользователи суперкомпьютера из других областей науки.

Обеспечение высокой эффективности работы суперкомпьютера требует постоянного контроля его состояния и поведения. Для этого используется специализированное системное программное обеспечение, которое собирает и анализирует разнообразные данные как о работе отдельных программных и аппаратных компонент суперкомпьютера (вычислительных узлов, служебных серверов, файловой системы, менеджера ресурсов и т.д.), так и о выполнении пользовательских приложений. При этом анализ производительности выполняемых на суперкомпьютере приложений очень важен, поскольку именно производительность приложений является одним из ключевых факторов, влияющих на эффективность работы суперкомпьютерной системы в целом.

В качестве входных данных для изучения производительности пользовательских приложений чаще всего используется информация о загрузке и поведении вычислительных узлов, собираемая во время выполнения этих приложений. Для получения подобной информации используются различные системы мониторинга (такие как Zabbix, Nagios, Collectd или DiMMon), которые могут собирать данные от самых разных источников на вычислительном узле — операционная система, аппаратные процессорные датчики, сетевая карта, графический ускоритель и т.д. Также важно получать от менеджера ресурсов информацию о запуске приложений. Широкий спектр собираемых данных и их источников предоставляет системным администраторам обширный объем раз-

нообразной полезной информации, который, однако, невозможно в полной мере обработать вручную. Поэтому в настоящее время все более важным становится вопрос разработки методов на основе технологий интеллектуального анализа данных для оценки состояния суперкомпьютеров. Эти методы позволят администраторам проще и эффективнее выявлять и устранять проблемы, которые могут снижать качество функционирования суперкомпьютеров. Примерами подобных проблем являются низкая производительность пользовательских заданий, неоптимальная конфигурация системного или прикладного ПО или неэффективная работа менеджера ресурсов, что приводит как к замедлению выполнения пользовательских расчетов, так и к простою вычислительных ресурсов.

За последние десятилетия наблюдается значительный рост применения методов интеллектуального анализа данных в различных областях, поскольку они позволяют выявлять закономерности в больших объемах данных и использовать их для решения практических задач с высокой точностью. Примерами таких областей являются машинное зрение (распознавание объектов, сегментация изображений), медицина (диагностика заболеваний, разработка лекарственных средств), геология (прогнозирование месторождений полезных ископаемых, моделирование качества воды), естественная обработка языка (распознавание речи, перевод и генерация текста) и многие другие. Активное развитие методов интеллектуального анализа данных повышает их применимость и в сфере высокопроизводительных вычислений и суперкомпьютеров. Современные суперкомпьютеры собирают огромное количество информации о работе своих компонент и запускаемых приложений, которая может быть использована для анализа качества их функционирования, однако ручной анализ этих данных зачастую затруднителен или даже невозможен из-за их большого объема. Методы интеллектуального анализа данных помогают решать эту проблему; в частности, они уже применяются для предсказания времени выполнения приложений или анализа состояния инженерной инфраструктуры.

Еще одной задачей, для решения которой можно применять методы интеллектуального анализа данных, является поиск схожих приложений, выполняющихся на суперкомпьютере, на что и направлена данная работа. В данном случае схожими будем называть приложения, которые обладают одними и теми же важными свойствами, связанными с поведением приложения во время его выполнения. Практика показывает, что если приложение A

обладает некоторыми свойствами (например, использует определенный программный пакет), и при этом оно схоже с приложением Б, то в ряде случаев можно считать, что приложение Б также обладает этими свойствами. Само определение схожести меняется в зависимости от того, какие свойства или особенности приложений рассматриваются, и это будет влиять на то, как в данной работе будет определяться оценка расстояния и построенные на ее основе методы выявления схожих приложений. Разработка методов обнаружения схожих приложений позволит анализировать новые задания, опираясь на результаты анализа изученных ранее приложений: проводить кластеризацию заданий, прогнозировать их поведение, выявлять аномальные запуски, выявлять проблемы с производительностью и т.д. Это значительно упрощает оценку поведения и эффективности выполнения приложений для пользователей и администраторов суперкомпьютеров.

Целью данной диссертационной работы является исследование и разработка методов на основе технологий интеллектуального анализа данных для решения задачи выявления схожих суперкомпьютерных приложений, а также реализация и применение разработанных методов на практике для решения актуальных для администраторов и пользователей задач, таких как определение используемых программных пакетов в приложениях или кластеризация приложений по их поведению.

Для достижения поставленной цели было необходимо решить следующие **задачи**:

1. Разработать и экспериментально исследовать методы выявления схожих суперкомпьютерных приложений с использованием технологий интеллектуального анализа данных.
2. Разработать алгоритмы для исследования потока суперкомпьютерных заданий на основе предложенных методов выявления схожих приложений.
3. Реализовать и апробировать программные решения для предложенных алгоритмов с целью исследования реального потока суперкомпьютерных заданий.

Научная новизна: В представленном исследовании предложены новые методы выявления схожих суперкомпьютерных приложений на основе технологий интеллектуального анализа данных, демонстрирующие высокую точность и эффективность. Эти методы основаны на анализе как статических данных

(исполняемых файлов приложений), так и динамического поведения заданий во время их выполнения. На основе этих методов предложены новые подходы к решению различных актуальных задач: обнаружение используемых программных пакетов и их версий в приложениях; кластеризация пользовательских заданий для определения групп пользователей, решающих похожие задачи; выявление “сбойных” групп заданий для их последующего детального анализа с целью выявления проблем с производительностью; предсказание оценок качества использования суперкомпьютерных ресурсов; обнаружение аномального поведения заданий на основе исторических данных. Предложенные методы могут быть также использованы для решения других важных задач, таких как прогнозирование времени выполнения заданий или создание рекомендательных систем, помогающих пользователям оптимизировать производительность своих параллельных программ.

Практическая значимость работы состоит в создании программного решения, позволяющего с высокой точностью выявлять схожие суперкомпьютерные приложения, а также в реализации вышеуказанных способов применения методов поиска схожих приложений с целью анализа потока заданий. Данное программное решение активно используется на суперкомпьютере петафлопсного уровня производительности Ломоносов-2, где показало свою применимость на практике и позволило администраторам данной вычислительной системы существенно продвинуться в исследовании качества ее работы. Данное решение может применяться и на других вычислительных системах.

Методология и методы исследования. При получении результатов диссертационной работы использовались элементы теории вероятностей, методы интеллектуального анализа данных и математической статистики. При выполнении программной реализации методов выявления схожих суперкомпьютерных приложений использовались методы объектно–ориентированного анализа и проектирования, классические алгоритмы и структуры данных.

Основные положения, выносимые на защиту:

1. Два разработанных подхода, основанные на анализе статической информации (исполняемых файлов приложения) и динамических данных о поведении задания (данных от системы мониторинга), позволяют решать задачу выявления схожих суперкомпьютерных приложений. По результатам апробации данные методы показывают высокое качество работы.

2. Подходы к выявлению схожих суперкомпьютерных приложений позволяют создать методы для решения задач обнаружения используемых программных пакетов и их версий в приложениях, кластеризации приложений, а также предсказания оценок качества использования суперкомпьютерных ресурсов. Экспериментальное исследование полученных решений на потоке заданий, запускаемых на суперкомпьютере Ломоносов-2, демонстрирует их работоспособность и применимость на практике.

Достоверность полученных результатов обеспечивается масштабной апробацией разработанных и реализованных методов анализа приложений на суперкомпьютере Ломоносов-2, обоснованием принятых решений по их разработке и внедрению.

Апробация работы. Представленные в работе результаты докладывались на следующих международных и российских конференциях:

1. XXVIII Байкальская Всероссийская конференция с международным участием, г. Иркутск, Россия, 29 июня - 8 июля 2023
2. Международная конференция «Parallel Processing and Applied Mathematics 2022», г. Гданьск, Польша, 8-11 сентября 2022
3. Международная конференция «Международная научная конференция студентов, аспирантов и молодых учёных «Ломоносов-2022», г. Москва, Россия, 20,21 апреля 2022
4. Международная конференция «Суперкомпьютерные дни в России 2021», г. Москва, Россия, 28 сентября 2021
5. Международная конференция «Параллельные вычислительные технологии (ПаВТ) 2021», Онлайн, Россия, 30 марта 2021
6. Международная конференция «Международная научная конференция студентов, аспирантов и молодых учёных «Ломоносов-2020», г. Москва, Россия, 10-27 ноября 2020
7. Международная школа-конференция «Интеллектуальный анализ Больших данных и распределенные системы», г. Будва, Hotel Splendid, Черногория, 1 октября 2019

Публикации. Основные положения и выводы диссертационного исследования в полной мере изложены в 4 научных работах, в том числе в 4 публикациях в рецензируемых научных изданиях, определенных п. 2.3 «По-

ложения о присуждении ученых степеней в Московском государственном университете имени М.В.Ломоносова».

Глава 1. Обзор существующих методов обнаружения схожих приложений

Целью данной диссертационной работы является исследование и разработка методов на основе технологий интеллектуального анализа данных для решения задачи выявления схожих суперкомпьютерных приложений, а также реализация и применение разработанных методов на практике для решения актуальных для администраторов и пользователей задач, таких как определение используемых программных пакетов в приложениях или кластеризация приложений по их поведению. И первым шагом исследования является обзор существующих решений для выявления схожих суперкомпьютерных приложений. Существующие решения разделяются на две группы: выявление схожих приложении на основе статических данных о приложении, как, например, исходному коду или исполняемым файлам, или на основе динамических данных, как, например, данных о производительности приложений, выходным данным, лог файлам и т.д.

1.1 Обзор методов на основе статических данных

Статический анализ подразумевает анализ приложения без его запуска, например, путем анализа исходного кода/исполняемых файлов приложения. Существуют множество работ по анализу исходного кода с целью выявления схожих программ, в основном с целью выявления факта плагиата у работ студентов, но также для выявления уязвимостей в коде. По данному направлению есть работа с обширным обзором, представленная в [1], в котором рассматривается задача сравнения программных реализаций в разных ракурсах (полное совпадение, изменение имен переменных и функций, отличие кода, но схожесть результата работы и т.д.) и какие в индустрии имеются решения данной задачи. Способы анализа в данном случае делятся на 2 варианта: с учетом семантики программного кода и без. Примером анализа без учета семантики является сравнение кодов как просто текстовых документов, применение которого показаны в [2], [3]. Если рассматривать методы анализа с учетом семантики, то одним

из основных методов является построение абстрактного синтаксического дерева (AST) из исходного кода приложения и последующего его анализа. Анализ заключается в поиске схожих поддеревьев у двух AST деревьев, что является вычислительно сложной задачей. Примерами такого подхода являются [4], [5]. Похожим подходом является графовый анализ исходного когда, который заключается в построении Program Dependence Graph (PDG). PDG представляет собой ориентированный граф, где каждая вершина представляет собой операцию, а ребра показывают зависимость операций по данным или управлению. Не обошлось и без методов машинного обучения, в которых на вход подаются как различные параметры исходного кода (количество строк, переменных, функций и т.д.), так и вышеупомянутые AST деревья, PDG и т.д., что позволяет методам машинного обучения "выбирать" то, что важно для выделения схожих кодов. Примеры предоставлены в [6], [7], [8].

Для выделения схожих приложений можно также использовать и исполняемые или объектные файлы. Они также обладают информацией о структуре программы, что позволяет их использовать для сравнения. В большинстве случаев сначала исполняемый файл преобразовывается в код ассемблера с помощью программного инструмента, называемого дизассемблер. Далее можно использовать методы выделения схожих приложений на основе исходного кода, как, например, с помощью AST деревьев и графовых алгоритмов [9], [10], [11], [12], [13]. Несмотря на то, что данный способ анализа позволяет обнаружить программы и алгоритмы, реализации которых, даже если сильно отличаются, решают схожие задачи, у данного способа есть большой недостаток: высокая вычислительная сложность. Это обусловлено тем, что почти каждый этап анализа является вычислительно сложной задачей — дизассемблирование, построение AST деревьев или PDG из дизассемблированного кода, и непосредственно сравнение полученных деревьев или графов.

Существуют также методы, включающие в себя анализ только характеристик исполняемого файла, например, имен функций и переменных, которые используются в данном файле. Пример такого анализа показан в [14], в которой используются методы машинного обучения в статическом анализе для обнаружения схожих приложений. В этой статье авторы сравнивают приложения для прогнозирования их дальнейшего энергопотребления с использованием исторических данных. Анализ основан на так называемых символах, извлеченных из

исполняемых файлов — элементах таблицы символов исполняемого файла, которые соответствуют используемым имен функций и переменных.

1.2 Обзор методов, использующих данные о динамике выполнения приложения

Данный вид анализа представляет собой анализ динамики выполнения приложений. Под динамикой выполнения будем понимать совокупность динамических характеристик, описывающих активность и характер использования вычислительных ресурсов системы во время выполнения определенного приложения. Данные о динамических характеристиках предоставляются системой мониторинга в режиме реального времени. Для каждого приложения собираются такие характеристики, как загрузка CPU, GPU, сети InfiniBand, количество промахов в кэш-память L1 или LLC в секунду и т. д. Каждая динамическая характеристика представляет собой временной ряд, в котором представлены показатели динамической характеристики во время исполнения задания. Таким образом, каждое задание представляет собой многомерный временной ряд, из-за чего в динамическом анализе задания будут считаться схожими, если описывающие их многомерные временные ряды схожи между собой. Пример похожих по поведению приложений показан на рисунке 1.1. На нем показаны динамические характеристики двух запущенных приложений, представленных разными типами линий. Видно, что общее поведение очень похоже друг на друга, хотя есть заметные локальные различия.

Если рассматривать задачу поиска схожих суперкомпьютерных заданий с помощью динамических характеристик, то работ по данному направлению не очень много. В [15] и [16] авторы обнаруживают схожие фазы выполнения приложения, используя методы кластеризации. Метрики «IPC» (среднее число выполненных инструкций за такт) и «Instructions retired» собираются во время выполнения приложений для группировки по разным всплескам активности CPU, относящимся к последовательным вычислениям между MPI коммуникациями. В данной работе авторы показывают, что только этих двух характеристик достаточно для разделения фаз приложения на кластеры, но такой подход неприменим при использовании данных системы мониторинга: большин-

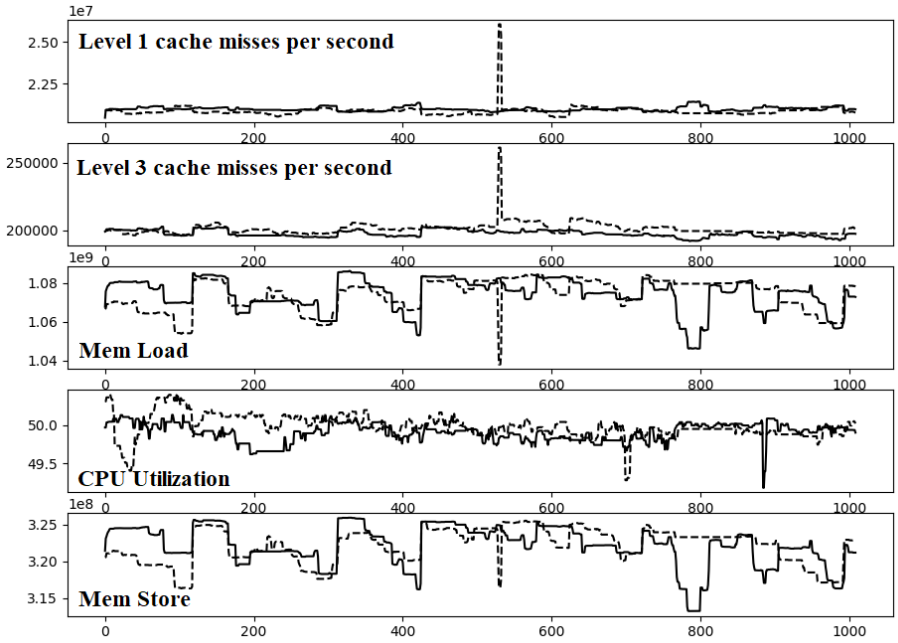

Рисунок 1.1 — Значения динамических характеристик во время исполнения приложений, схожих по поведению. Первое приложение представлено сполшной линией, а второе — пунктирной.

ство систем мониторинга группирует данные не по всплескам активности CPU, а по временным интервалам (обычно от 1 минуты до 1 часа). Следовательно, суть подхода в таком случае неприменима.

Но из-за того, что поиск схожих заданий сводится к сравнению многомерных рядов, то можно применить традиционные методы их анализа. При поиске таких методов, были выявлены следующие группы: Dynamic Time Warping и его вариации, методы структурного анализа и нейросетевые подходы.

1.2.1 Применение Dynamic Time Warping для сравнения временных рядов

Dynamic Time Warping (DTW, [17]) — один из самых популярных и эффективных методов оценки расстояния между временными рядами. Алгоритм находит оптимальное соответствие для точек во временном ряду, которые имеют схожее поведение, но не выровнены по времени. То есть для выбранной функции расстояния f между точками временного ряда алгоритм DTW для временного ряда X и Y выдает такую последовательность пар (x_i, y_j), что сумма

$f(x_i, y_j)$ минимальна. При сравнении временных рядов стандартной функцией расстояния является евклидово расстояние. Поскольку сложность наивного алгоритма DTW является квадратичной ($O(NM)$), для ускорения его работы были предложены различные варианты: FastDTW [18], PrunedDTW [19], SparseDTW [20] и т.д., которые вводят набор ограничений для значительного сокращения количества необходимых операций.

Одним из примеров использования DTW является работа [21]. В этой статье авторы используют модификацию DTW для расчета расстояния между многомерными временными рядами. Они разбивают исходный временной ряд на интервалы, превращая каждый в одну точку, значительно сокращая длину временного ряда. Это было сделано для дальнейшего сокращения времени работы DTW.

DTW — довольно старый и простой, но эффективный метод. Однако у DTW есть свои недостатки: часто используемое евклидово расстояние очень чувствительно к масштабированию и сдвигам. В связи с этим рассматриваются и другие функции расстояния, решающие подобные задачи. Например, в [22] авторы предлагают сравнивать не сами точки, а приближенные производные. В [23] авторы предлагают анализировать временной ряд не как последовательность точек, а как последовательность векторов (точка-точка), а затем использовать косинусное сходство как функцию расстояния между векторами. Это делает вышеупомянутые алгоритмы устойчивыми к смещениям по оси Y, и эти методы можно использовать и для многомерных временных рядов, поскольку предлагаемые функции расстояния менее чувствительны к многомерности, чем евклидово расстояние. Но эти методы имеют недостатки, такие как чувствительность к кратковременным колебаниям, что приводит к необходимости “сглаживания” временного ряда перед применением любого из вышеперечисленных методов.

1.2.2 Сравнение временных рядов на основе их структурных особенностей

Другим вариантом для решения задачи сравнения временных рядов является структурный анализ. Эти методы основаны на наборе так называемых

примитивов, или базовых фигур, которые можно использовать для описания частей временного ряда (например, постоянной, прямой, синуса и т.д.), а затем временной ряд преобразуется в последовательность примитивов для дальнейшего анализа. Подробная информация об этом методе приведена в [24]. Анализ можно проводить как с помощью теории формальных грамматик, так и с помощью методов статистического анализа, используя полученную последовательность символов/примитивов в качестве вектора признаков.

1.2.3 Применение нейронных сетей для сравнения временных рядов

Существует несколько работ по обнаружению сходства временных рядов с помощью нейронных сетей. В [25] авторы предлагают использовать сиамскую нейронную сеть, которая дает вероятность того, насколько похожи временные ряды. Метод основан на принципе двух рекуррентных сетей с одинаковыми параметрами (веса, смещения и т.д.), получающих в качестве входных данных временные ряды для дальнейшего сравнения. Основная задача рекуррентных сетей состоит в том, чтобы обнаруживать особенности и закономерности во временном ряду, важные для дальнейшего сравнения, которые затем подаются в качестве входных данных для нейронной сети прямого распространения. Данная сеть прямого распространения в дальнейшем решает, похожи ли эти временные ряды или нет.

Нейронные сети также могут использоваться для уменьшения размерности, после чего к полученным рядам применяются традиционные методы сравнения временных рядов. Такие нейронные сети называются автокодировщиками. Архитектура таких сетей включает так называемый слой узкого места с меньшим количеством нейронов, чем количество измерений во входном векторе. Сама нейронная сеть предназначена для воссоздания исходного сигнала на своем выходе, что позволяет проводить обучение без учителя, а в процессе обучения сеть учится обнаруживать общие черты и закономерности во временном ряду. Пример использования кодировщика можно увидеть в [26], где после уменьшения размерности временного ряда результат используется как вход для другой нейронной сети, которая вычисляет матрицу соответствия элементов

временного ряда. Эта информация используется для расчета расстояния между временными рядами.

Все вышеперечисленные методы, использующие нейронные сети для сравнения временных рядов, теоретически подходят для решения нашей задачи, но их использование очень затруднено из-за отсутствия большого количества размеченных приложений, доступных для обучения.

Другой пример использования автокодировщика для сравнения временных рядов показан в [27]. В этой статье авторы использовали рекуррентные автокодировщики для получения более компактного представления многомерного временного ряда, а затем использовали стратифицированное локально-чувствительное хеширование для получения вектора фиксированной длины (хэш). Затем они использовали эти хэш-значения для сравнения и классификации с использованием алгоритма ближайшего соседа (1NN). Метод показал хорошие результаты в прогнозировании острых гипотензивных эпизодов, но авторы статьи не показали, как они сравнивают полученные хэши для временных рядов, что является очень важной частью этого алгоритма.

1.3 Выводы и задачи предстоящего исследования

При выборе потенциального направления исследования в области анализа статических данных необходимо учитывать то, какие данные для анализа нам доступны. У администраторов чаще есть доступ к исполняемым и объектным файлам, нежели к исходному коду приложений. Также стоит учитывать скорость обработки заданий из-за большого числа запускаемых заданий на суперкомпьютере, что исключает семантический анализ исполняемых файлов. Остается способ анализа с использованием статических характеристик исполняемых файлов. У данного способа есть недостаток относительно других способов анализа статических данных — зачастую обфускация исходного кода с легкостью обходит такой способ анализа и не позволяет найти схожие приложения, но данная особенность важна при анализе, например, программ студентов для выявления плагиата, а на суперкомпьютере пользователям обычно нет смысла намеренно усложнять свой код, потому данный способ анализа и был выбран.

В дальнейшем будем считать, что задания статически похожи, если их исполняемые файлы используют схожие имена функций и переменных.

При рассмотрении методов анализа динамических данных о приложении, необходимо учесть плюсы и минусы каждого из направлений. Методы сравнения временных рядов на основе их структурных особенностей зачастую не учитывают абсолютные значения ряда, что нам не подходит, так как абсолютные значения играют очень большую роль в том, как воспринимается то или иное задание. Даже если графики загрузки CPU у двух заданий схожи, но значения отличаются в 2 раза — их нельзя считать схожими. По этой причине данный подход на практике сменили такие методы анализа временных рядов, как DTW и нейронные сети, поскольку они показывают хорошие результаты за счет простоты как с точки зрения реализации, так и вычислительной сложности.

Применение нейронных сетей для сравнения временных рядов выглядит хорошим направлением для исследования. Но есть и большой недостаток — нейронные сети требуют большой обучающей выборки для получения достойного качества работы. В данном случае обучающая выборка должна будет состоять из пар заданий с информацией о том, схожи они или нет. Как будет показано далее, для тестирования будут использоваться более 600 пар вручную размеченных заданий, и с их помощью можно оценить качество работы методов, но для обучения нейронных сетей этого недостаточно. Из-за этого, было принято решение не использовать данное направление. Но стоит отметить, что применение нейронных сетей возможно в другом аспекте анализа — уменьшения размерности динамических данных на основе архитектуры автокодировщика.

Несмотря на то, что Dynamic Time Warping (DTW) — достаточно старый метод, он все еще является одним из основных способов сравнения временных рядов и является стандартом в данной области. Зачастую новые методы сравнения временных рядов сравнивают именно с DTW, и DTW показывает хорошее качество работы. Учитывая недостатки вышеупомянутых методов и такие достоинства DTW, как устойчивость к временным сдвигам, а также возможность сравнения временных рядов различной длины и применения различных функций расстояния между точками временных рядов, делает DTW отличным кандидатом для решения задачи обнаружения схожих приложений на основе динамических данных.

Глава 2. Разработка метода обнаружения схожих приложений на основе статических данных

2.1 Разработка метода на основе модели Doc2Vec

Первым выбранным видом оценки расстояния между заданиями является статический анализ. В нем задействована только статическая информация, такая как исходный код или, в нашем случае, исполняемые файлы приложения. Приложения могут вести себя по-разному в зависимости от параметров их запуска, поэтому данный вид анализа можно использовать только для выявления групп приложений, схожих по структуре кода. При рассмотрении одного пользователя этот метод позволяет определить различные классы заданий, которые он выполняет. С точки зрения разных пользователей, такой анализ может показать, например, какие пользователи работают над одними и теми же задачами или используют одни и те же пакеты/библиотеки приложений, что также является полезной информацией о работе суперкомпьютера.

Поскольку метод, описанный в [14], теоретически подходит для решения нашей задачи, было принято решение его адаптировать для наших целей. В рамках этого метода сначала извлекается информация из исполняемых файлов приложений с помощью стандартной утилиты ОС Linux “nm”, чтобы получить имена функций и переменных (в дальнейшем называемые термами). Однако во время компиляции имена функций и переменных зачастую преобразуются компилятором в уникальное внутреннее представление. Для получения исходных имен мы использовали опцию -C у nm. Также необходимо отметить, что некоторые термы, которые часто встречаются в исполняемых файлах и поэтому не добавляют полезной информации, были исключены из рассмотрения. Мы исключили термы, которые встречаются в более чем 90% исполняемых файлов, такие как `main`, `__libc_start_main`, `memcpy`, `__frame_dummy_init_array_entry` и т.д. Всего было исключено около 30 часто встречающихся термов.

Далее необходимо преобразовать полученные наборы термов в сущности, которые можно легко сравнивать между собой. Для этого была использована нейросетевая модель Paragraph Vector [28]. Она является расширением

известной модели Word2vec [29], которая применяется для создания векторных представлений слов. Модель Paragraph Vector также известна как Doc2Vec, и в дальнейшем данные термины будут использоваться взаимозаменяемо.

Paragraph Vector позволяет описывать векторами фиксированной длины не только отдельные слова, но и целые документы. Существуют два режима работы данной модели: Paragraph Vector с распределенной памятью (PV-DM) и с распределенным набором слов (PV-DBOW). Основное различие между ними заключается в том, что PV-DM пытается подобрать такой вектор документа, чтобы по контексту слова максимально точно предсказать следующее слово, в то время как PV-DBOW пытается подобрать такой вектор документа, чтобы максимально точно предсказать случайное слово из документа. Иными словами, в PV-DM учитывается порядок слов, в PV-DBOW — нет, из-за чего для решения задачи выделения схожих приложений было принято решение использовать PV-DBOW, так как термы, извлеченные из исполняемых файлов, не упорядочены.

Для применения модели Doc2Vec необходимо пройти следующие шаги:

- Обучение
  - Преобразование слов и документов в обучающем наборе в вектора фиксированный длины с помощью метода one-hot encoding (работа метода описана далее).
  - Обучение нейронной сети с архитектурой, показанной на рисунке 2.1, для получения матрицы весов D и W. Матрица весов D отвечает за преобразование вектора one-hot encoding для документа из обучающего набора в вектор фиксированной длины; матрица весов W отвечает за предсказание слов по вектору документа.
- Векторизация новых исполняемых файлов
  - Преобразование анализируемого документа и слов в нем в вектора фиксированный длины с помощью метода one-hot encoding.
  - Обучение нейронной сети с архитектурой, показанной на рисунке 2.2, для получения матрицы весов D'. Матрица весов W совпадает с шагом обучения, фиксируется и не изменяется на этом шаге.
  - По матрице весов D' получаем вектор фиксированной длины для нового документа.

Как было упомянуто выше, первым шагом является обучение модели Doc2Vec. Для этого требуется набор уникальных документов. Пусть количество документов в обучающем наборе равно C, а количество уникальных слов во всех документах — V. Первым этапом является построение словаря с применением метода one-hot encoding, который работает следующим образом.

Каждому слову W ставится в соответствие его порядковый номер от 1 до V (например, k). Слову W ставится в соответствие вектор длины V, в котором все элементы равны 0, кроме индекса порядкового номера слова (то есть k) — в данной позиции значение равно 1. Данный вектор будем обозначать как E_W.

Аналогично словам, документам также ставится в соответствие вектор длины C. Каждый документ из обучающей выборки получает порядковый номер от 1 до C, и соответствующий ему вектор имеет значение ноль на всех позициях, кроме его порядкового номера в обучающем наборе — в данной позиции значение равно единице. Следовательно, каждое слово и документ рассматривается как вектор длины V и C соответственно, которые используются для обучения модели. В процессе обучения также фиксируется длина вектора, в который преобразуется документ — N.

При обучении Doc2Vec имеет простую архитектуру нейронной сети, показанную на рисунке 2.1. Функция активации скрытого слоя — линейная, выходного слоя — softmax. При обучении обучающий набор преобразуется следующим образом: для каждого документа d и слова в этом документе w строится пара $\{E_d : E_w\}$. В процессе обучения нейронная сеть подбирает веса в матрицах D и W1 таким образом, чтобы на основе вектора документа d максимально точно предсказывать слова из этого документа. Обучение осуществляется с использованием стохастического градиентного спуска.

После обучения, получается финальный набор матриц весов D и W1, но возникает вопрос — как их использовать для получения вектора фиксированной длины для нового документа? Для этого используется архитектура нейронной сети, показанная на рисунке 2.2. Матрица весов W1 заимствуется из обученной нейронной сети и фиксируется, в то время как матрица весов D' будет изменяться в процессе обучения данной сети. Поскольку новый документ является единственным при повторном обучении, его one-hot encoding представляет собой вектор длиной 1: [1]. Для каждого слова w из нового документа создаются пары $\{[1] : E_w\}$, которые используются в процессе обучения новой сети. После

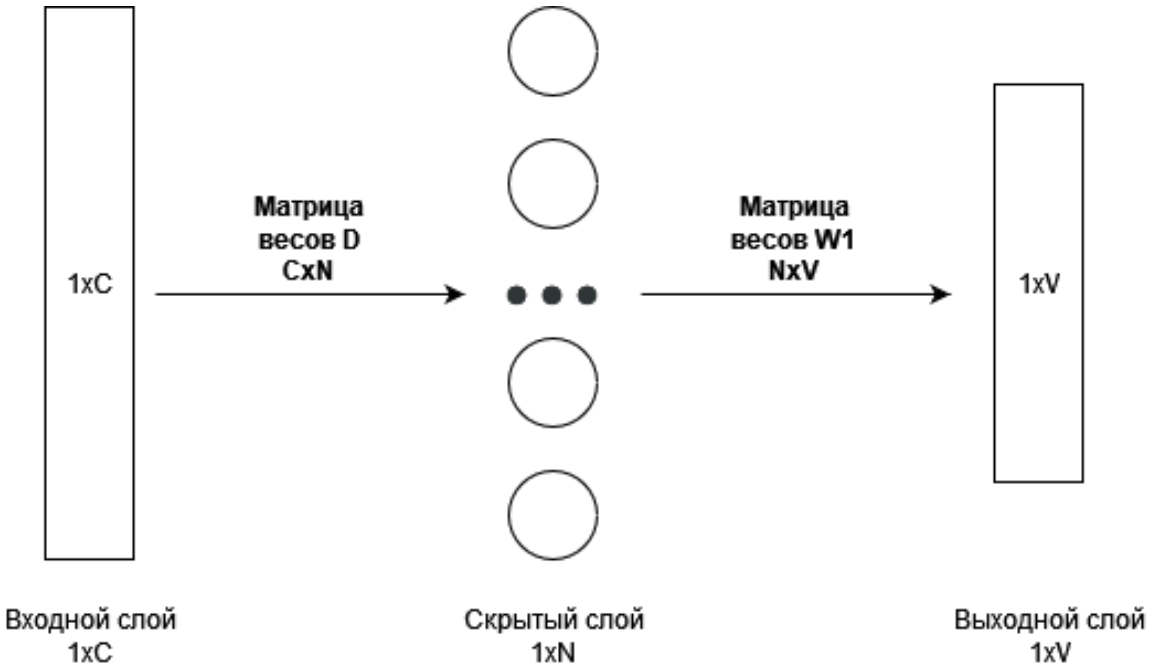

Рисунок 2.1 — Используемая архитектура нейронной сети PV-DBOW для обучения.

завершения обучения сети матрица весов D' используется в качестве вектора фиксированной длины, описывающего данный документ.

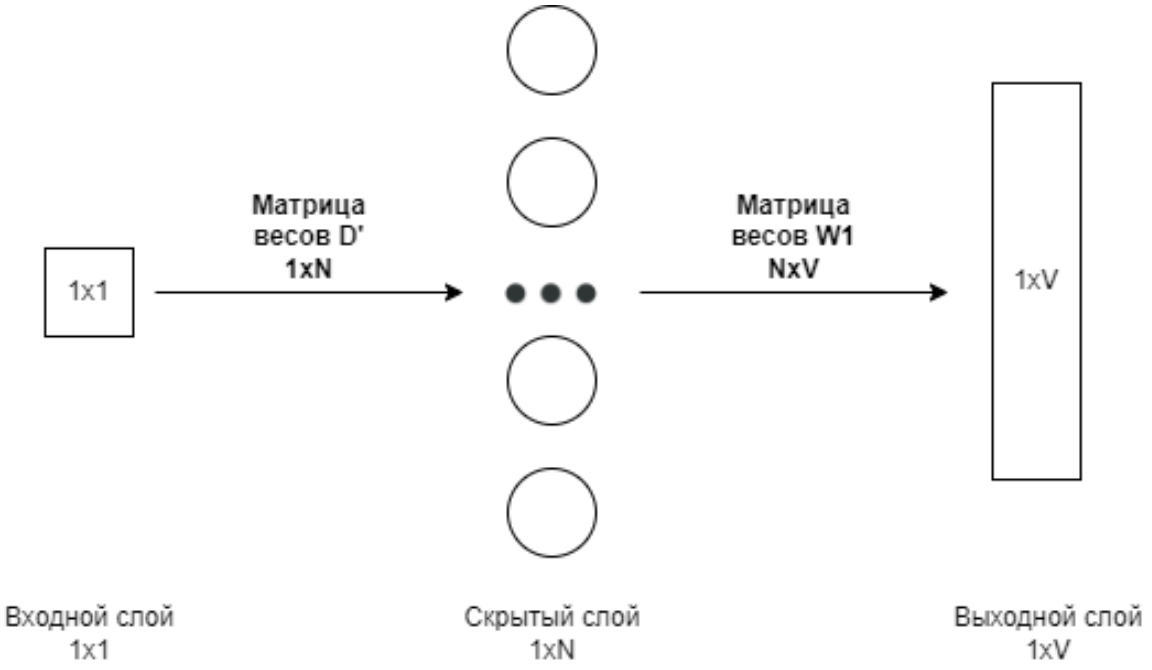

Рисунок 2.2 — Используемая архитектура нейронной сети PV-DBOW для получения вектора нового документа.

В данной работе был использован программный пакет gensim [30] для Python, который предоставляет модель PV-DBOW. Модель PV-DBOW была обучена с использованием данных из более чем 7000 уникальных исполняемых

файлов, собранных за более чем 2 года работы суперкомпьютера “Ломоносов-2” и включающих исполняемые файлы более 300 пользователей.

После получения векторного представления фиксированной длины для каждого документа используется косинусное расстояние в качестве меры для сравнения приложений по формуле 2.1, где A и B — вектора фиксированной длины, а θ — угол между ними. Косинусное расстояние измеряет косинус угла между векторами и показывает, насколько они похожи. Косинусное расстояние было выбрано из-за того, что оно лучше подходит чем, например, Евклидово расстояние, для метода Word2Vec, который используется в модели Doc2Vec. У Word2Vec есть свойство, при котором более часто встречающиеся слова имеют более длинные вектора, но при этом направление вектора практически не меняется. Евклидово расстояние сильно зависит от длины векторов, в то время как косинусное сходство зависит только от их направления, поэтому в данной работе оно лучше подходит для сравнения приложений.

$$\text{cosine distance}(A, B) = 1 - \cos(\theta) = 1 - \frac{A \cdot B}{\|A\|_2 \|B\|_2} \tag{2.1}$$

Таким образом, был предложен метод для преобразования исполняемых файлов приложений в вектора фиксированной длины, которые можно использовать для их последующего сравнения с помощью косинусного расстояния. Это позволит для новых, неизученных приложений найти схожие с ним приложения из исторических данных.

2.2 Алгоритм статического метода обнаружения схожих приложений

Предлагается использовать следующий алгоритм работы метода (он также проиллюстрирован на рисунке 2.3):

- Получение исполняемых файлов приложения.
- Применение утилиты "nm"для извлечения имен используемых функций и переменных (термов) из исполняемых файлов.
- Применение обученной модели Doc2Vec на извлеченных термах для получения вектора фиксированной длины.

– Сравнение полученного вектора с векторами заданий из исторических данных с помощью косинусного расстояния.

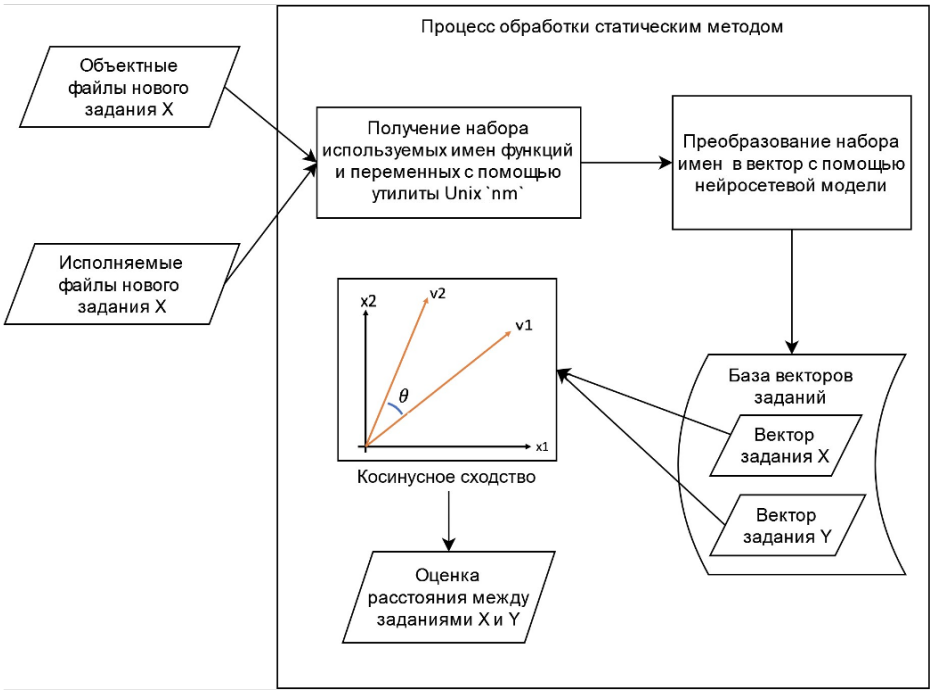

Рисунок 2.3 — Предлагаемый алгоритм статического метода выделения схожих приложений.

2.3 Оценка точности предложенного метода на реальных данных

Для проверки работоспособности и точности полученной модели было решено использовать следующий подход:

– вручную выделить кластеры на основе сходства между заданиями;
– использовать разработанный метод статического анализа для получения оценки расстояния между каждой парой заданий;
– провести кластеризацию задания, используя полученные расстояния;
– сравнить ручную и полученную кластеризации.

Для проверки корректности работы статического метода выделения схожих заданий было принято решение вручную кластеризовать задания пользователей, но для этого использовалось одно допущение. Пути к исполняемым файлам часто включают названия как решаемых задач, так и их версий. Версиями могут выступать как очередная итерация модификации исходного кода

разработчиком, так и изменение версии используемой программной библиотеки. В связи с этим приложения считались схожими, если их пути совпадают, но версии разные. Таким образом, было отобрано 132 приложения, которые вручную были разделены на 17 различных кластеров.

После получения ручной разметки, было произведено их попарное сравнение с помощью метода статического анализа. Это позволило получить оценку расстояния для каждой пары заданий, что дальше использовалось для проверки качества работы метода.

Так как полученная ручная разметка представляет собой результаты работы методы кластеризации, для проверки качества было целесообразно использовать существующие оценки качества методов кластеризации. Для проведения автоматической кластеризации был выбран метод иерархической агломерационной кластеризации из пакета sklearn для Python [31]. Метод основан на том, что изначально все элементы принадлежат разным кластерам, после чего, в зависимости от расстояния между элементами, некоторые кластеры сливаются в один. Поэтому в данном методе кластеризации не требуется указывать количество кластеров, так как оно автоматически определяется на основе расстояния между элементами, что является одним из основных причин выбора данного алгоритма.

Для сравнения полученных кластеров был выбран скорректированный индекс Рэнда (Adjusted Rand Index, ARI) [32]. Это мера того, насколько близки два варианта кластеризации друг к другу, где значение 1 соответствует точному совпадению, а значение 0 соответствует случайному распределению кластеров. Для вычисления ARI необходимо определить понятие матрицы сопряженности, пример которой показан в таблице 1, где X_i — это группа элементов i в результате кластеризации X, Y_j — группа элементов j в результате кластеризации Y, и n_{ij} — количество элементов, входящие как в группу X_i, так и в Y_j. На основе данных значений высчитывается значение ARI, формула которой показана в 2.2.

$$ARI = \frac{\sum_{ij} \binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] \Big/ \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right] - \left[\sum_i \binom{a_i}{2} \sum_j \binom{b_j}{2}\right] \Big/ \binom{n}{2}} \tag{2.2}$$

Одним из самых главных параметров метода иерархической кластеризации является порог расстояния между элементами. Смысл данного порога в том, что если расстояние между элементами меньше данного порога, то они

Таблица 1 — Матрица сопряженности для двух результатов кластеризаций: X и Y

	Y_1	Y_2	...	Y_s	суммы
X_1	n_{11}	n_{12}	...	n_{1s}	a_1
X_2	n_{21}	n_{22}	...	n_{2s}	a_2
...	...	...	...	...	...
X_r	n_{r1}	n_{r2}	...	n_{rs}	a_r
суммы	b_1	b_2	...	b_s	

считаются принадлежащими одному кластеру, иначе — нет. В зависимости от данного порога меняется результат кластеризации, а значит и значение ARI. На рисунке 2.4 показан график изменения значения ARI в зависимости от порогового значения. Наилучшим показателем ARI стало 0.79, что считается высоким показателем для ARI. Стоит отметить, что почти все задания были сгруппированы корректно, за исключением четырех кластеров, каждый из которых разделился на два отдельных кластера. Такое поведение можно объяснить ошибкой в векторном описании приложения или ошибкой в допущении при ручной кластеризации. Ошибкой векторного описания приложения будем называть случаи, когда несильно отличающиеся входные данные преобразуются в сильно отличающиеся выходные векторы. Данный феномен может возникать, если во время обучения модель Doc2Vec учитывает некоторые термы с большим весом, чем это требуется на самом деле, и отсутствие/присутствие данных термов во входных файлах может вести к сильному изменению в результате работы метода.

Стоит отметить, что на графике изменения ARI отчетливо видно “плато” для диапазона порогового значения от 0.16 до 0.3. Возникает вопрос — не целесообразно ли использовать пороговое значения из данного плато при дальнейшем использовании метода? Однако на практике было обнаружено, что при выборе порогового значения из диапазона от 0.1 до 0.3, качество работы статического метода было наилучшим при значениях близким к 0.11. Это подтверждалось на задачах обнаружения использования программных пакетов, предсказания качества использования суперкомпьютерных ресурсов и кластеризации приложений, которые будут описаны в следующих разделах.

В заключение отметим, что метод статического анализа применим для обнаружения похожих приложений на суперкомпьютере “Ломоносов-2” и может

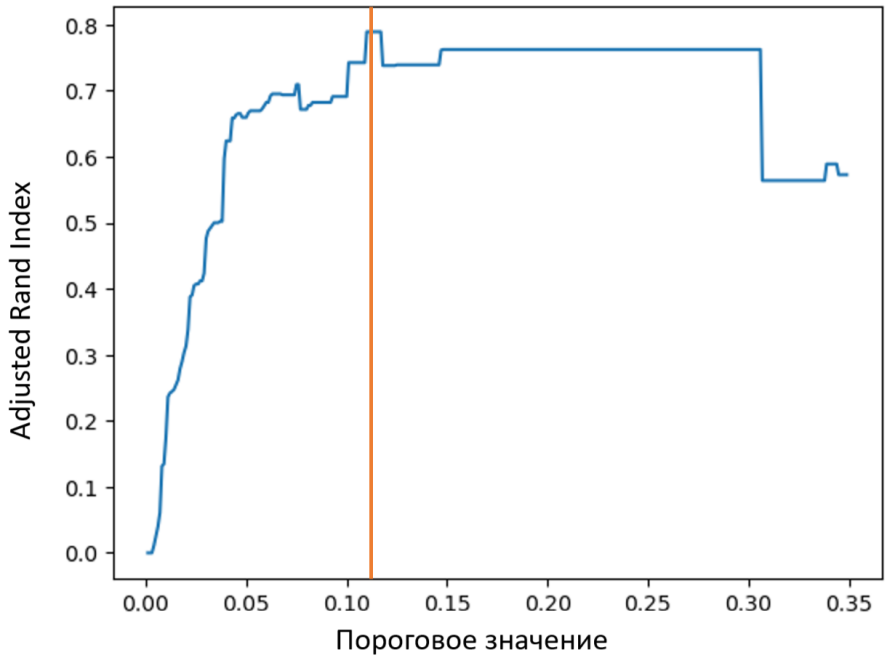

Рисунок 2.4 — График изменения ARI от порогового значения.

быть использован для выделения различных групп приложений как в пределах одного пользователя, так и в пределах нескольких пользователей. Однако он не может объяснить более тонкие различия в отдельных классах приложений в связи с отсутствием информации о том, как приложения себя ведут во время исполнения с определенными параметрами запуска или входными данными, в связи с чем также был предложен второй метод обнаружения на основе динамических данных, который позволяет решить эту проблему.

Глава 3. Разработка метода обнаружения схожих приложений с применением данных о динамике выполнения

3.1 Входные данные

Следующим видом анализа является динамический анализ, т.е. анализ поведения приложения во время его выполнения на суперкомпьютере, который может нивелировать недостаток метода, описанного в предыдущем разделе: несмотря на отличные результаты в обнаружении разных классов приложений, статический метод не может различить изменения, которые вносятся за счет разных параметров запуска или входных данных, а также состоянием вычислительной среды.

Суперкомпьютеры собирают огромное количество информации во время исполнения приложения, которые могут отражать различные изменения в его поведении. На суперкомпьютере “Ломоносов-2” работает система мониторинга, которая агрегирует данные с различных датчиков вычислительных узлов. Показания датчиков описывают различные аспекты того, как работает приложение: например, интенсивность работы с памятью или сетью InfiniBand, загруженность центральных процессоров и графических ускорителей, и так далее. Агрегированные показания определенного датчика в дальнейшем будем называть значениями динамической характеристики. Агрегация происходит с заданной периодичностью (в случае суперкомпьютера “Ломоносов-2” — каждую минуту), благодаря чему для каждой динамической характеристики формируется временной ряд, где каждая точка – минутные агрегированные данные по показателям датчика.

В данной работе используются следующие входные данные о заданиях, собираемые для каждого вычислительного узла:

- Количество промахов в кэш-память первого уровня в секунду
- Количество промахов в кэш-память третьего уровня в секунду
- Загрузка CPU
- Количество полученных байт в секунду по сети Infiniband
- Количество полученных пакетов в секунду по сети Infiniband
- Количество отправленных байт в секунду по сети Infiniband

- Количество отправленных пакетов в секунду по сети Infiniband
- Load average (среднее число активных процессов на узел)
- Количество полученных байт в секунду по сети файловой системы
- Количество полученных пакетов в секунду по сети файловой системы
- Количество отправленных байт в секунду по сети файловой системы
- Количество отправленных пакетов в секунду по сети файловой системы
- Загрузка GPU
- Количество обращений в память GPU в секунду
- Загрузка памяти GPU

Указанные динамические характеристики позволяют достаточно полно описать динамику работы приложения во время его исполнения на суперкомпьютерном комплексе.

3.2 Разработка метода на основе алгоритма Dynamic Time Warping

После получения данных о задании, возникает вопрос — как сравнивать временные ряды для обнаружения схожих приложений? Перед ответом на данный вопрос необходимо выделить те требования, которые были посчитаны важными, а именно:

- Алгоритм сравнения должен уметь анализировать временные ряды различной длины. Данное требование исходит из того, что на суперкомпьютерах запускаются приложения с различным итоговым временем исполнения, и необходимо это учитывать. Есть разные подходы, которые позволяют привести различные временные ряды к одинаковой длине (например, с помощью аппроксимации промежуточных значений).
- Алгоритм должен быть устойчив к локальным временным сдвигам. Зачастую у приложений существуют фазы исполнения, которые от запуска к запуску будут иметь разное время исполнения, и таким образом части временного ряда могут быть сдвинуты, иметь меньшую или большую длину и т.д. Такие случаи необходимо правильно обрабатывать. На рисунке 3.1 приведен пример динамических характеристик для двух схо-

жих заданий, у которых как длительность, так и начало фазы падения активности отличаются.

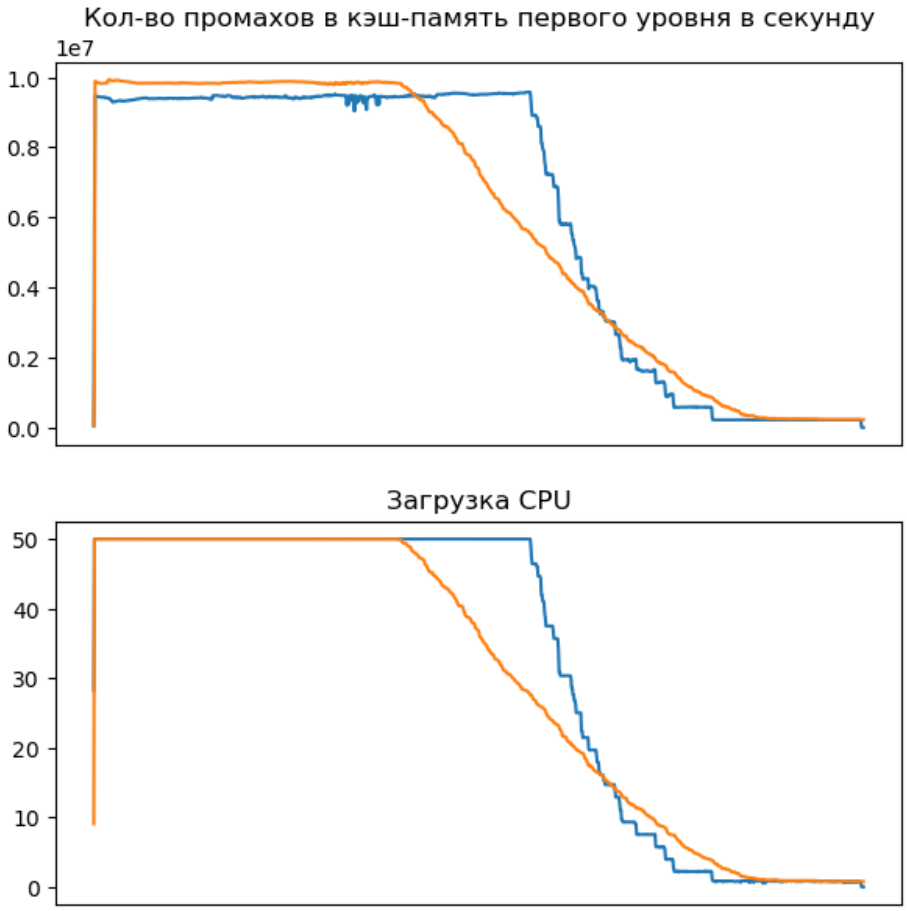

Рисунок 3.1 — Графики изменения динамических характеристик схожих заданий во время их исполнения. Задание 1 выделено оранжевым цветом, задание 2 — синим.

В области сравнения временных рядов наиболее популярными методами являются евклидово расстояние, динамическая трансформация временной шкалы и методы на основе образов/форм. Для расчета евклидова расстояния необходимо, чтобы временные ряды имели одинаковую длину, и формула показана в 3.1, где A и B — временные ряды длиной n, A_i и B_i — это i-е точки временных рядов A и B соответственно. Данный метод чувствителен к временным сдвигам, и это можно увидеть на примере из рисунка 3.1: из-за разного момента начала фаз исполнения, а также длительности фазы поэлементное расстояние в местах расхождений будет велико. А как было сказано ранее — поведение идентичных запусков приложения (с одинаковым набором параметров, идентичной версией приложения и входными данными) может сильно отличать-

ся из-за локальных сдвигов, что делает евклидово расстояние неприменимым для решения задачи выделения схожих суперкомпьютерных приложений.

$$ED(A, B) = \sqrt{\sum_{i=1}^{n}(A_i - B_i)^2} \tag{3.1}$$

Еще одним способом сравнения временных рядов является использование методов на основе образов/форм. Данные методы сначала преобразуют исходные временные ряды в другое представление, которые в дальнейшем сравниваются. Примерами являются: преобразование Фурье; перевод непрерывной динамической характеристики в дискретную; преобразование временного ряда в последовательность заранее определенных типовых форм (например, прямая линия или форма ступеньки [24]). В нашем случае, преобразование Фурье не подходит из-за того, что временные сдвиги в рядах могут сильно влиять на результаты работы метода, а временные сдвиги происходят часто; преобразование в последовательность типовых форм не подходит в связи с тем, что в данных методах теряется информация об абсолютных значениях временных рядов, что является очень важной составляющей данных о производительности приложений.

Преобразование непрерывной динамической характеристики в дискретную является одним из основных способов анализа временных рядов. Идея состоит в том, что высчитываются статистические характеристики временных рядов, которые в дальнейшем анализируются. Примерами статистических характеристик являются среднее значение, стандартное отклонение, коэффициент ассиметрии и эксцесса, перцентили и т.д. В случае многомерных рядов, статистические характеристики высчитываются для каждого временного ряда.

В контексте данной работы, под анализом понимается сравнение с целью выявления факта схожести временных рядов. Данный анализ можно проводить с помощью методов машинного обучения. Так как нам важно то, схожи временные ряды или нет, для решения данной задачи подходят методы классификации. В роли классов будет выступать факт схожести (класс А) и факт различия (класс Б), то есть классификатор будет определять, схожа пара многомерных временных рядов или нет. Для этого необходимо произвести обучение классификатора на обучающем наборе, который состоит из пар заданий и их класса (схожие или нет). Количество необходимых данных в обучающем наборе очень сильно зависит от используемого метода классификации. Например,

для линейных методов классификации может быть достаточно десятков примеров для каждого из классов. А для нелинейных методов, как метод случайных деревьев или нейронных сетей, как правило, необходимо существенно больше информации для получения классификатора с высоким качеством работы. Данная проблема усугубляется с увеличением размерности входных данных, так как при маленьком размере обучающего набора и высокой размерности входных данных может произойти переобучение — феномен, когда классификатор начинает хорошо классифицировать элементы из обучающего набора, но при классификации новых, ранее не встреченных элементов, начинает делать существенные ошибки. В данной работе используются 15 динамических характеристик, и даже при использовании пяти статистических характеристик на каждую динамическую характеристику в итоге получается 75 характеристик. По нашим оценкам, для обучения классификатора с хорошей точностью необходимо минимум 1000 элементов в обучающей выборке, а также дополнительные 20% в тестовой выборке. За неимением такого рода информации о заданиях, обучающую и тестовую выборки необходимо собирать вручную, что очень трудозатратно. Также сложно предсказать итоговое качество работы классификатора на основе статистических данных временных рядов. В связи с данными рисками, было принято решение не исследовать данное направление.

Другим популярным способом сравнения временных рядов является алгоритм динамической трансформации временной шкалы (DTW). Идея алгоритма в том, чтобы получить оптимальное соответствие между точками временных рядов согласно формуле 3.2, где B — временной ряд длиной n, C — временной ряд длиной m, π — это последовательность пар индексов временных рядов B и C $((i_0, j_0),...,(i_{K-1},j_{K-1}))$ длиною K с условиями:

- $i_0 = j_0 = 0$
- $i_{K-1} = n - 1, j_{K-1} = m - 1$
- $i_{k-1} <= i_k <= i_{k+1}$
- $j_{k-1} <= j_k <= j_{k+1}$

В нашем случае d — это евклидово расстояние между точками временных рядов.

$$DTW(B, C) = \min_{\pi \in \mathcal{A}(x,x')} \sum_{(i,j)\in\pi} d(B_i, C_j) \tag{3.2}$$

Пример работы метода DTW показан на рисунке 3.2. На рисунке демонстрируется изменение динамических характеристик тех же заданий, что были

показаны на рисунке 3.1, но после оптимального сопоставления точек методом DTW. Можно заметить, что метод DTW удалось практически идеально сопоставить фазу падения активности у заданий, что минимизирует среднее расстояние между временными рядами.

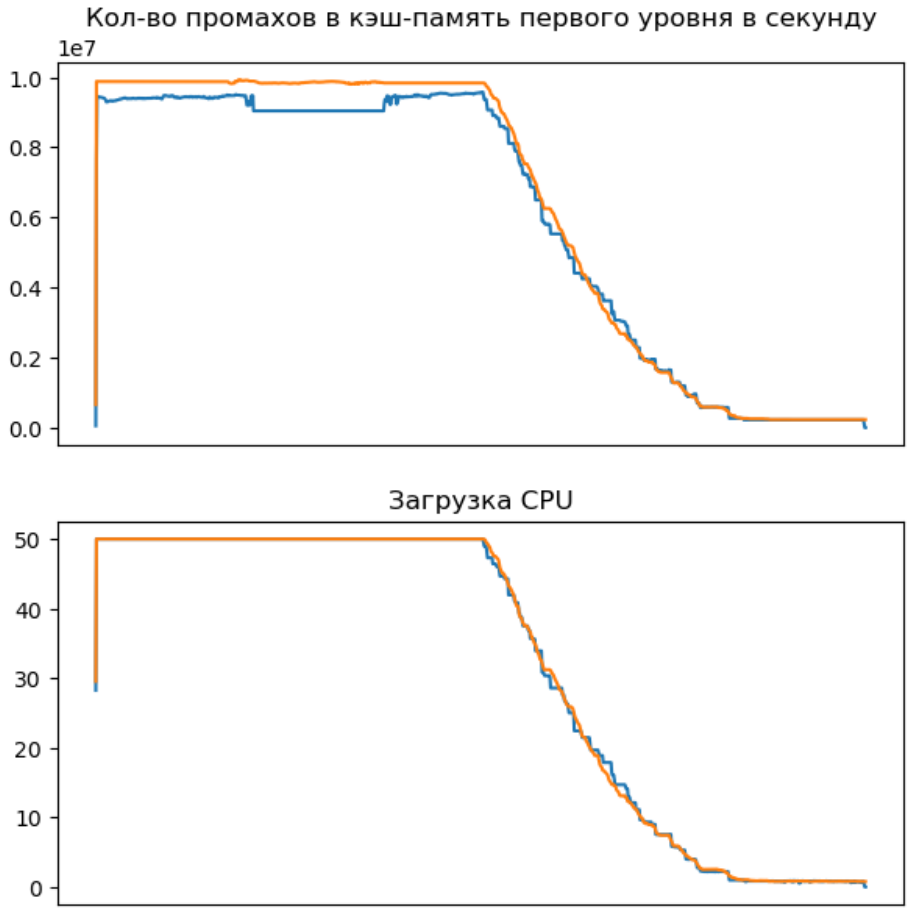

Рисунок 3.2 — Графики изменения динамических характеристик схожих заданий после работы алгоритма DTW.

В статьях, использующих DTW как метод сравнения временных рядов, длина временного ряда часто фиксируется ([21], [33]). В нашем случае длина временного ряда может существенно варьироваться, что является проблемой: расстояние, возвращаемое алгоритмом DTW, зависит от длины сравниваемого временного ряда, что делает невозможным сравнение $DTW(A, B)$ и $DTW(A, C)$ для рядов A, B и C, если их длины различны. Чтобы решить эту проблему, после получения оптимального соответствия между рядами π длиною K, мы используем формулу 3.3 для расчета расстояния между временными рядами:

$$dist(A, B) = \frac{DTW(A, B)}{K}, \tag{3.3}$$

которая показывает среднее расстояние между точками временных рядов после получения оптимального соответствия. Это значение и используется для оценки схожести двух заданий.

Таким образом, предлагается использовать следующий алгоритм сравнения заданий на основе их динамических данных:

- Извлечение данных системы мониторинга.
  - Для каждого из сравниваемых заданий a и b из БД системы мониторинга извлекаются данные о динамике их работы — временные ряды A и B.
- Проводится предварительная обработка временных рядов A и B. Данный этап детально будет рассмотрен в следующем разделе.
  - Обработка пропущенных значений.
  - Приведение динамических характеристик к одному диапазону.
  - Понижение влияния определенных характеристик.
- Вычисление оценки расстояния между заданиями a и b с помощью формулы 3.3.

3.3 Предобработка данных

Одним из самых важных этапов при использовании методов интеллектуального анализа данных является этап предварительной обработки данных. Источниками входных данных могут быть: люди, данные из сенсоров в инженерных помещениях, логи пользовательских приложений и т.д., и собираемая информация зачастую несравнима между собой. Как пример рассмотрим используемую в данной работе систему мониторинга суперкомпьютера Ломоносов-2, которая собирает информацию о динамике работы приложения во время его исполнения. И многие из динамических характеристик совершенно не сопоставимы друг с другом. Какие-то характеристики имеют известные пределы своих значений (например, загрузка CPU/GPU варьируется от 0 до 100), но другие — нет. Например, зачастую очень непросто оценить практически достижимую максимальную частоту кэш-промахов. И для более эффективного анализа желательно приводить такие характеристики к одному диапазону значений.

Также источники данных могут быть ненадежными. Одной из частых проблем во входных данных является наличие отсутствующих значений. Если рассматривать используемую в данной работе систему мониторинга, это означает отсутствие данных за какой-то период работы приложения, и необходимо решить, что делать в такой ситуации — игнорировать этот промежуток, или же заполнить отсутствующие значения на основе доступной информации.

От качества предобработки данных зависит итоговое качество работы предлагаемых методов, из-за чего этому уделяется особое внимание.

3.3.1 Обработка пропущенных значений

Важным аспектом предварительной обработки временных рядов является обработка пропущенных значений. В нашем случае система мониторинга собирает и агрегирует данные о работе приложений с определенной периодичностью (раз в минуту на суперкомпьютере “Ломоносов-2”), поэтому под пропущенными значениями временных рядов понимается отсутствие данных динамических характеристик в указанные периоды времени. Для демонстрации различных методов обработки пропущенных значений рассмотрим пример с функцией синуса с пропущенными значениями, изображенным на рисунке 3.3.

Существуют различные методы обработки отсутствующих значений во временных рядах. Одним из простейших методов является игнорирование пропущенных данных, при котором временные метки с отсутствующими или NaN значениями удаляются из ряда. Этот подход приводит к искусственному “схлопыванию” временного промежутка, исключая его из анализа. Пример данного подхода показан на рисунке 3.4.

Другими популярными методами являются заполнение пропущенных значений предыдущими или последующими значениями. Основная идея таких методов заключается в замене отсутствующих значений на ближайшее доступное значение до или после пропуска. Пример работы данных методов показан на рисунке 3.5.

Еще одним из популярных способов заполнения пропущенных данных является интерполяция. Интерполяция предполагает использование имеющихся данных временного ряда для заполнения пропущенных значений, при этом

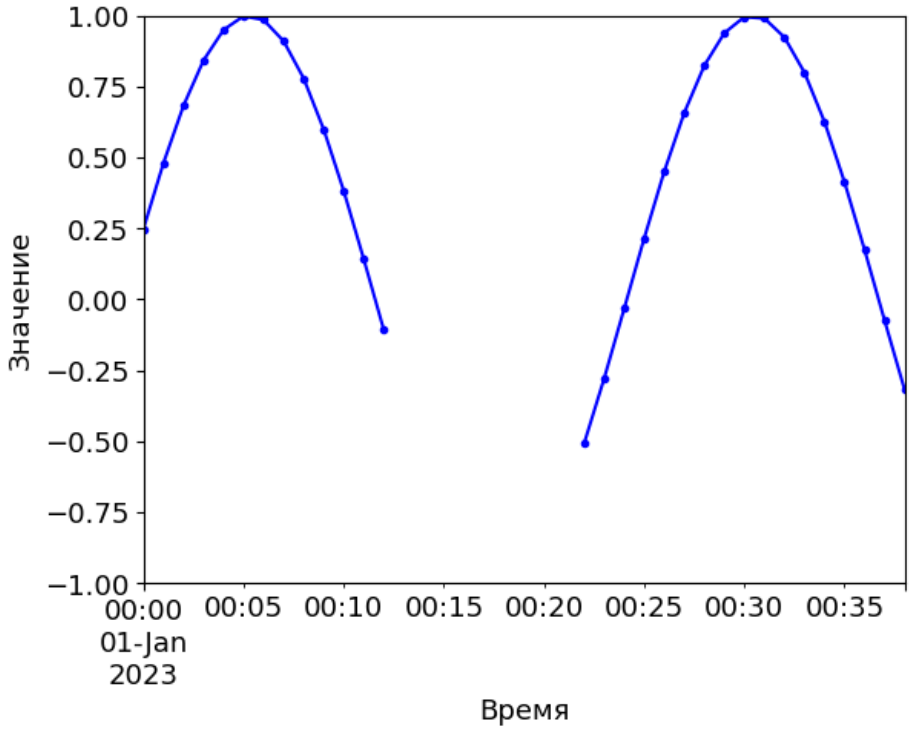

Рисунок 3.3 — Пример с пропущенными значениями на основе функции синуса.

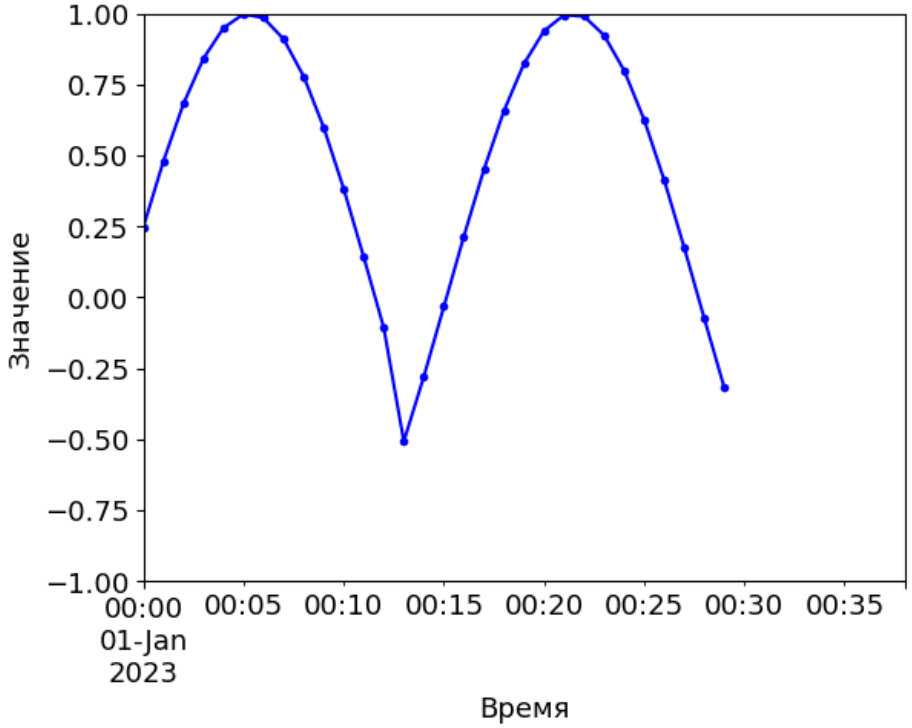

Рисунок 3.4 — Пример игнорирования пропущенных данных.

между соседними элементами временного ряда проводится кривая, которая аппроксимирует пропущенные значения. Заполнение предыдущими и последующими значениями также может рассматриваться как форма интерполяции, где кривая является постоянной. Однако существуют более сложные и точные методы интерполяции, такие как линейная интерполяция. В данном методе для

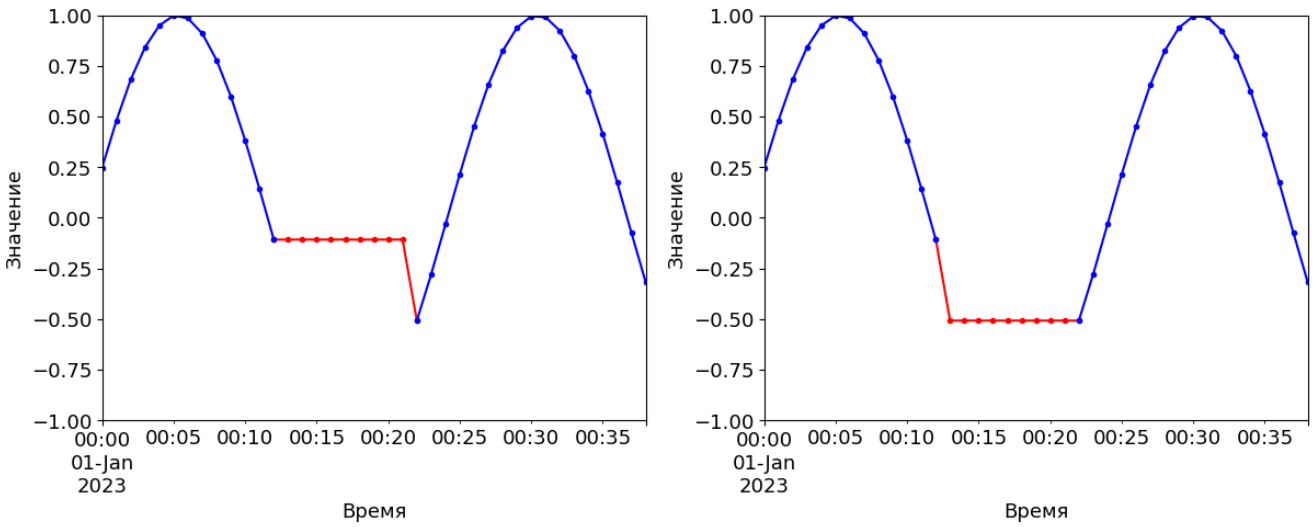

Рисунок 3.5 — Пример заполнения пропущенных значений с помощью первого доступного предыдущего и последующего значений соответственно.

двух соседних точек $(t_i,\ y_i)$ и $(t_{i+1},\ y_{i+1})$ временного ряда строится линейная кривая по формуле 3.4, и все промежуточные значения для временных меток t_k заполняются согласно данной формуле. Пример линейной интерполяции приведен на рисунке 3.6.

$$f(t) = \frac{y_{i+1} - y_i}{t_{i+1} - t_i} * (t - t_i) + y_i \tag{3.4}$$

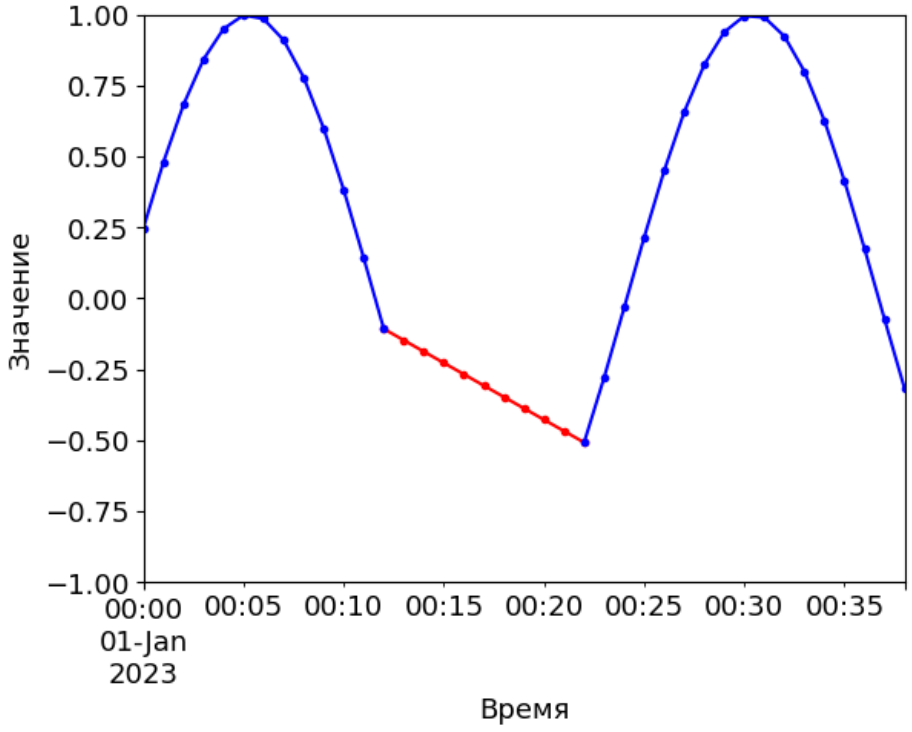

Рисунок 3.6 — Пример линейной интерполяции данных.

Более сложный способ интерполяции данных представляет собой кусочную полиномиальную интерполяцию при помощи сплайнов. Линейная интерполяция не обеспечивает гладкость кривой, то есть дифференцируемость в каждой точке временного ряда как самого ряда, так и его производной. Данная методика позволяет в определенных случаях достаточно реалистично заполнить пропуски в данных, как, например, на рисунке 3.7. Детали реализации данного метода не представляются в рамках данного исследования, однако основная идея заключается в разбиении временного ряда на отрезки, на которых данные аппроксимируются полиномами высокой степени. Кроме того, устанавливаются дополнительные условия, такие как непрерывность производных первого или второго порядка на протяжении всего временного ряда, что обеспечивает гладкость кривой.

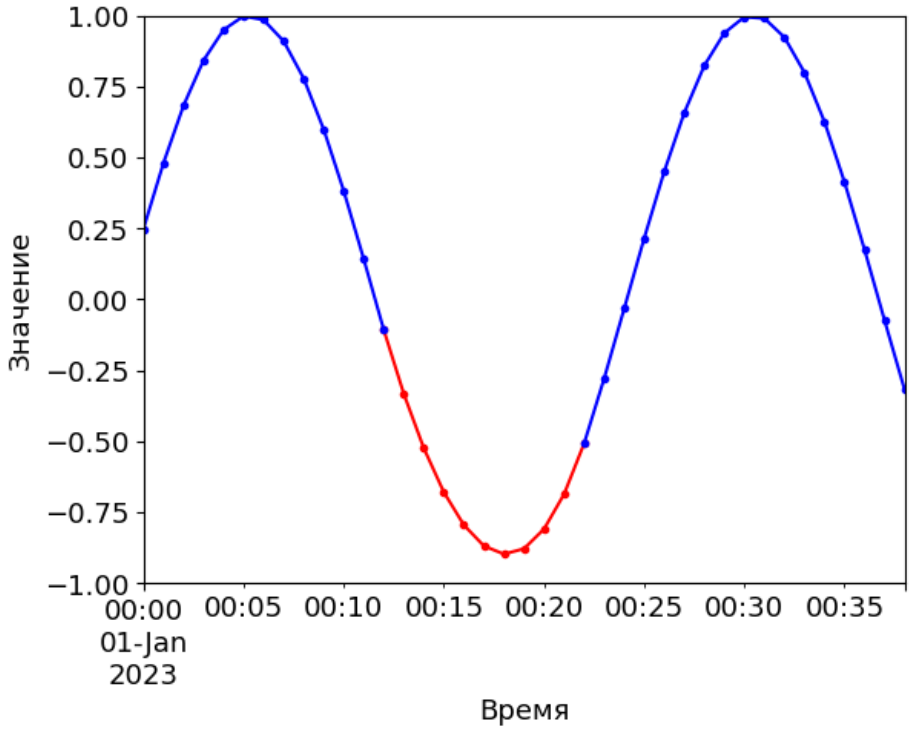

Рисунок 3.7 — Пример кусочной полиномиальной интерполяции.

Выше были описаны возможные способы обработки пропущенных значений, но возникает вопрос — какой из данных методов лучше подходит для решения поставленной задачи на данных используемой системы мониторинга? На рисунке 3.8 приведены данные по количеству промахов в кэш-память первого уровня в секунду у реального задания суперкомпьютера “Ломоносов-2” с пропущенными значениями. Данное поведение достаточно типичное для задания с отсутствующими данными, поэтому рассмотрим, как разные методы

заполняют пропущенные данные (результаты приведены на рисунке 3.9). Игнорировать данные нежелательно, так как тогда теряется информация в тех промежутках, в которых отсутствуют данные, что влияет как на общую картину поведения приложения, так и на длительность работы рассматриваемого приложения.

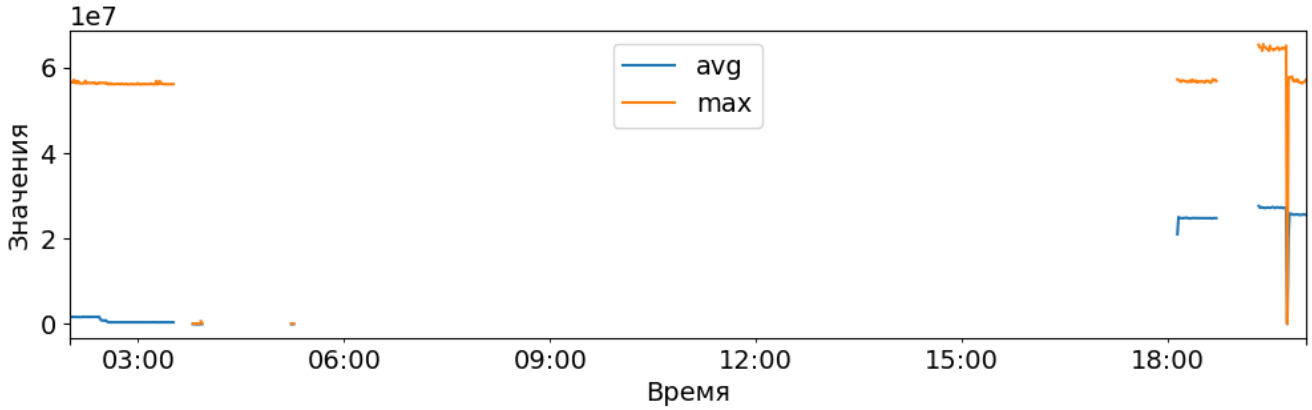

Рисунок 3.8 — Пример данных реального задания суперкомпьютера Ломоносов-2 с пропущенными значениями.

Заполнение пропусков первым доступным предыдущим или последующим значением может как завысить, так и преуменьшить динамические характеристики в интервалах с отсутствующими значениями. Линейная и кусочная полиномиальная аппроксимация показали себя наиболее подходящими методами заполнения пропусков, хотя результаты их работы схожи, и нет четкой разницы между ними. В результате было принято решение использовать линейную интерполяцию в дальнейшем исследовании.

3.3.2 Варианты преобразования значений характеристик

Как описывалось ранее, система мониторинга предоставляет данные о разного рода характеристиках работы приложения, и данные характеристики не равны по абсолютным значениям. Например, загрузка CPU всегда варьируется от 0 до 100 процентов, в то время как среднее количество промахов в кэш-память последнего уровня варьируется от 0 до теоретического максимума (десятки, а то и сотни миллионов), который меняется в зависимости от аппаратных характеристик системы. Также, не всегда заранее неизвестен

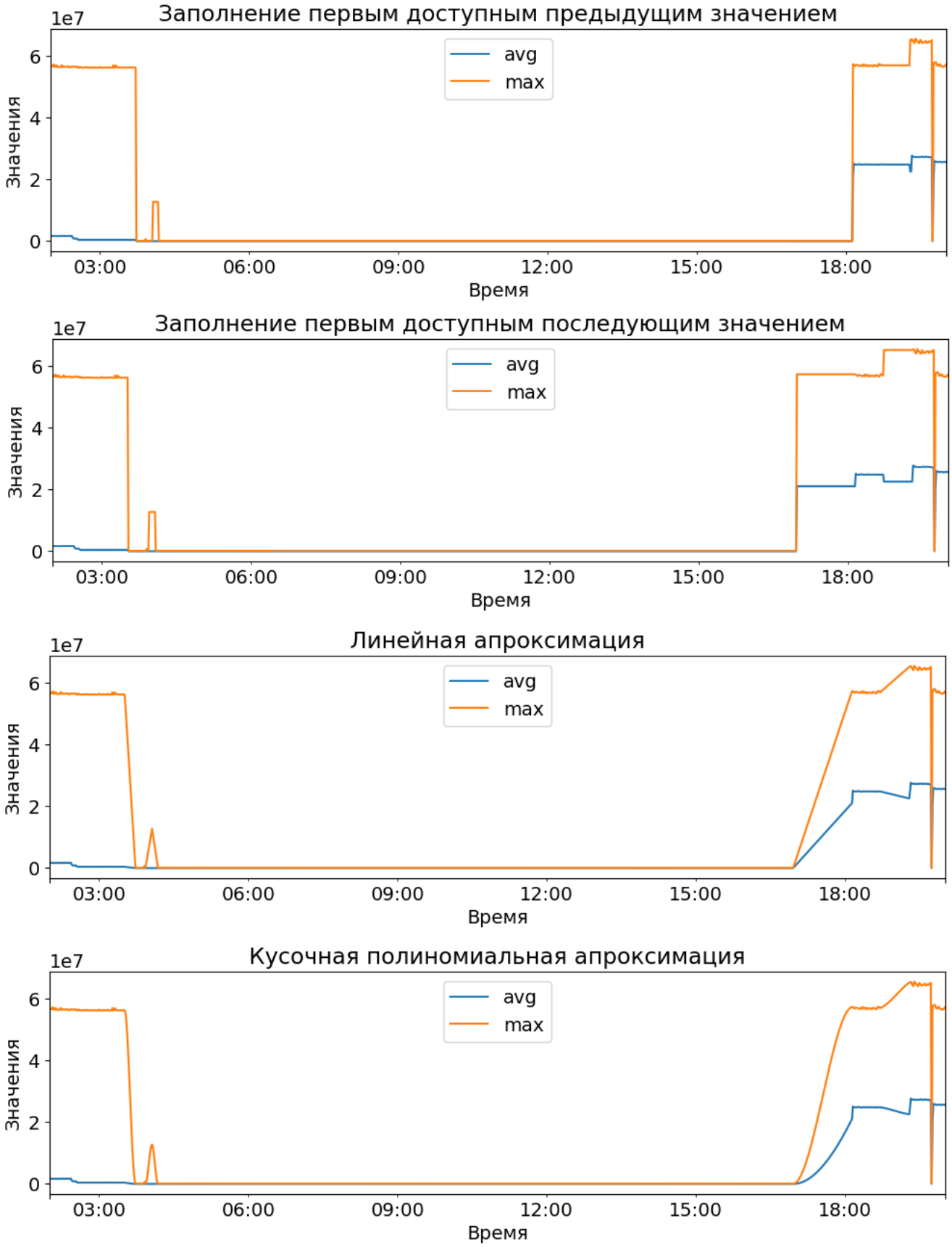

Рисунок 3.9 — Пример работы методов апроксимации данных на примере данных задания суперкомпьютера “Ломоносов-2”.

возможный диапазон значений для корректного масштабирования, ибо к теоретическому максимуму зачастую можно приблизиться только с помощью специализированных бенчмарков, а в реальных приложениях достигаемый максимум может быть существенно меньше. Поэтому, когда мы считаем расстояние между точками временных рядов, нам необходимо преобразовать динамические характеристики таким образом, чтобы их влияние на общее расстояние было примерно одинаковым. Стоит отметить, что значения всех динамических характеристик больше или равно 0.

Самыми известными видами преобразования входных данных являются минмаксная нормализация и стандартизация. Минмаксная нормализация работает так, чтобы данные после преобразования попадали в интервал $[a, b]$. Зачастую выбираемые интервалы равны [0, 1] и [-1, 1]. Формула данного преобразования показана на (3.5). x_{min} и x_{max} могут иметь значение как теоретического минимума и максимума (например, 0 и 100 для загрузки CPU), так и минимальные и максимальные значения, которые наблюдались для данного показателя на практике. Данное преобразование обычно применяется при отсутствии аномальных выбросов в данных, и когда данные имеют равномерное распределение.

$$\widetilde{x_i} = \frac{x_i - x_{min}}{x_{max} - x_{max}} * (b - a) + a \tag{3.5}$$

Формула стандартизации, или Z-score нормализации, показана в (3.6). $\overline{X}$ — это среднее значение характеристики, а σ — стандартное отклонение. Данные показатели считаются для используемого набора данных, и то, как распределены данные, очень сильно влияет на итоговое значение. Суть преобразования — сделать так, чтобы после преобразования данных среднее значение было равно 0, а дисперсия — 1. Данное преобразование хорошо подходит для данных с аномальными выбросами, а также для данных с нормальным распределением.

$$\widetilde{x_i} = \frac{x_i - \overline{X}}{\sigma} \tag{3.6}$$

Но если посмотреть на распределения значений, то видно, что у некоторых характеристик большинство значений расположены в очень узком промежутке у нуля, и это практически не дает нам различить данные значения между собой. Одним из выходов в данной ситуации является Log1p преобразование, формула которой показана в 3.7. Log1p — это использование логарифмической

шкалы вместо равномерной, из-за чего расстояния между маленькими значениями увеличиваются, а между большими — уменьшаются. Результат применения данного преобразования показан на рисунках 3.10 и 3.11. Видно, что после преобразования распределения значений стало близкими к нормальному, и таким образом мы можем дополнительно применить стандартизацию для унификации диапазона значений относительно других характеристик.

$$\widetilde{x_i} = log(1 + x_i) \tag{3.7}$$

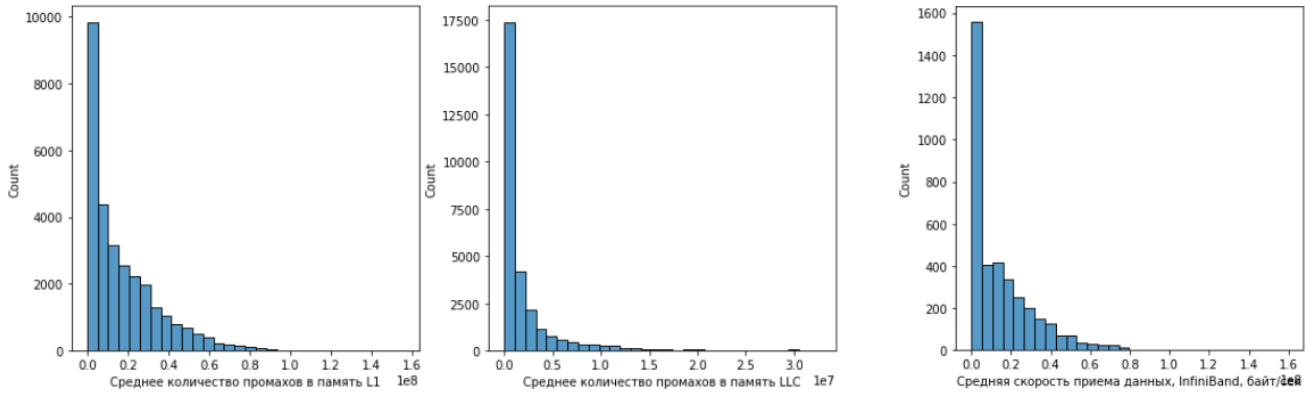

Рисунок 3.10 — Распределение значений динамических характеристик до преобразования.

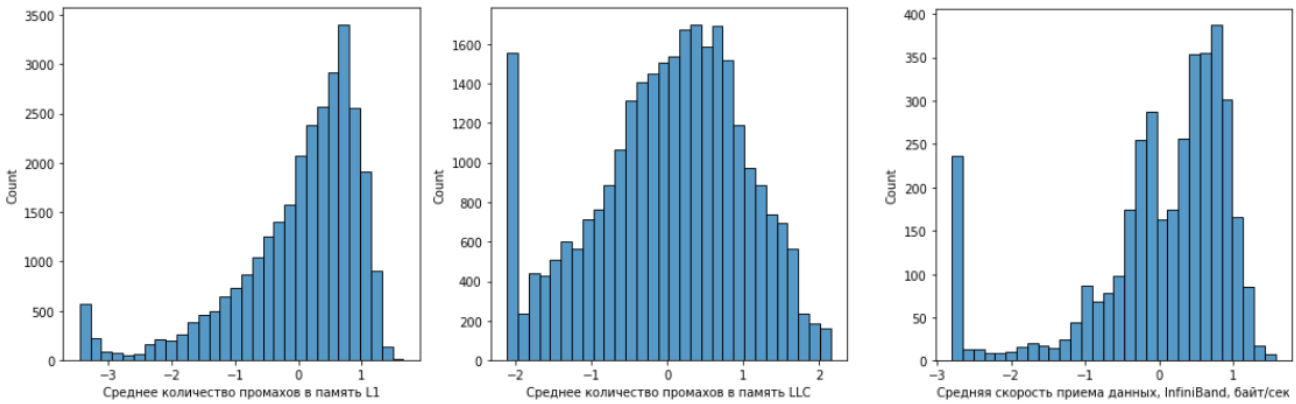

Рисунок 3.11 — Распределение значений динамических характеристик после log1p преобразования.

Рассмотрев указанные варианты предобработки данных, возникает вопрос: к каким динамическим характеристикам применить какой из вариантов, чтобы это дало наилучшие результаты? Это и стало первым шагом в подборе параметров преобразования данных.

Подбор методов преобразования данных и их параметров. Как упоминалось ранее, от выбора правильного метода преобразования данных очень сильно зависит итоговое качество работы методов интеллектуального анализа данных. Но для начала процесса подбора этих методов, необходимо иметь размеченную выборку заданий, для которых известно, схожи они или нет. Эта информация необходима для получения некоторого стартового значения точности, с которой в дальнейшем будут сравниваться результаты подбора. Точность в данном случае будем воспринимать как отношение правильно выделенных схожих и не схожих пар заданий к общему числу пар. Точность определяется с помощью порогового значения: если оценка расстояния меньше порогового значения, то считаем задания схожими между собой. Пороговое значение подбирается перебором значений от 0.01 до 1 с шагом 0.01 для получения максимальной точности.

Для начала были размечены $\sim$200 пар заданий по принципу схожести их поведения. Количество пар схожих и не схожих заданий было одинаково для корректной оценки точности. Считать точность до первой итерации подбора параметров преобразования данных было бессмысленным из-за того, что те динамические характеристики, у которых абсолютные значения были значительно выше, чем у остальных (как, например, количество промахов в кэш-память 1-го уровня), определяли схожесть заданий.

Первой итерацией было применение просто минмаксной нормализации на всех динамических характеристиках. Это позволит получить первое значение точности, от которой мы будем отталкиваться в будущем, несмотря на то, что для некоторых характеристик данные преобразование совсем не подходит из-за единичных выбросов. На данной итерации была получена точность 0.6.

Следующим шагом был подбор метода преобразования для каждой из характеристик. В большинстве случаев тип применяемого преобразования напрашивается сам собой: например, загрузка CPU и loadavg зачастую ограничены 100 и количеством логических ядер соответственно, из-за чего можно применить минмакс нормализацию, а логарифмическое преобразование количества пересылаемых байт по сети Infiniband позволяет легче отличить более низкие показатели данной характеристики. Но после применения логарифмического преобразования данные все равно необходимо преобразовать, чтобы они были в диапазоне, близком с другими динамическими характеристиками. На данном этапе количество рассматриваемых методов ограничено: необходи-

мо проверить, применять логарифмическое преобразование или нет, и потом применить минмакс нормализацию или стандартизацию. Это четыре варианта на одну характеристику, и для 16 используемых характеристик получается $4 * 10^9$ возможных вариантов. Полным перебором рассмотреть все варианты не представляется возможным. Поэтому мы применили логарифмическое преобразование на характеристики, для которых это имеет смысл — количество промахов в кэш-память 1-го и 3-го уровней, количество пересылаемых байт и пакетов по сети Infiniband, количество пересылаемых байт и пакетов с файловой системы. С учетом этого, количество вариантов для перебора опускается до 65536, что мы и сделали. После перебора всех вариантов, точность у лучшего варианта составила 0.77. Стоит отметить, что данный вариант использует только стандартизацию, т.е. получается, что даже для жестко ограниченных данных, таких как загрузка CPU, стандартизация оказалась лучше, чем нормализация.

3.3.3 Методы уменьшения размерностей

Следующим шагом стало рассмотрение методов уменьшения размерности точек временных рядов, т.е. количества значений, которые характеризуют поведение приложения в конкретный момент времени. Методы уменьшения применяются для преобразования векторов большой размерности в векторы меньшей размерности, в которых сохраняются значимые свойства исходных векторов. Уменьшение размерности эффективно используется для преодоления "проклятия размерности", в которой решаемая задача сильно усложняется, если исходные данные имеют высокие размерности (верно для задач машинного обучения, кластеризации и т.д.). Это применимо также к функциям расстояния. Например, для функции Евклидова расстояния верно то, что при увеличении размерности точек расстояния между ними становятся примерно одинаковыми. То есть, если $dist_{max}(x_i)$ — это расстояние от x_i до самой дальней точки x_k, а $dist_{min}(x_i)$ – расстояние от x_i до ближайшей точки x_m, то верно следующее утверждение:

$$\lim_{d \to \infty} \frac{dist_{max}(x_i) - dist_{min}(x_i))}{dist_{min} x_i} \to 0, \tag{3.8}$$

т.е. становится все сложнее и сложнее различить точки между собой. Данная проблема распространяется и на DTW, так как в нем используется евклидово расстояние. Данная проблема только начинает проявляться при размерностях больше 10, что является одной из главных причин проверить методы уменьшения размерностей.

Наиболее часто используемым методом уменьшения размерности является метод главных компонент (PCA). Суть PCA в том, чтобы найти такие подпространства меньшей размерности, в ортогональной проекции на которые разброс данных (то есть среднеквадратичное отклонение от среднего значения) максимален. Результатом работы PCA являются подобранные подпространства меньшей размерности, на которые исходные данные можно спроецировать и таким образом получить преобразование в меньшую размерность.

Другой рассматриваемый способ заключается в использовании нейросетевой модели, которая имеет архитектуру автокодировщика. Автокодировщик представляет собой архитектуру с “узким горлышком”, как показано на рисунке 3.12. Размерность слоя “узкого горлышка” должна быть обязательно меньше размерности входных данных. Данную нейронную сеть можно воспринимать как регрессионную модель, которая пытается воссоздать исходные данные, т.е. в нашем случае при обучении на вход подается точка временного ряда, а на выходе результат сравнивается с этой же точкой временного ряда для корректировки. Но встает вопрос — в чем смысл такого обучения? Весь смысл в слое “узкого горлышка”, и что его размерность меньше размерности входных данных. Если бы размерность этого слоя была равна или больше размерности входных данных, то нейронной сети ничего не надо было делать и просто возвращать значения входных данных в выходном слое. Но из-за того, что размерность меньше, нейронной сети необходимо выявлять во входных данных только нужные свойства и особенности, которые позволили бы максимально близко восстановить исходные данные по неполным данным. После обучения нейронной сети, можно отбросить часть декодировщика, и использовать часть кодировщика для уменьшения размерности.

Применение методов уменьшения размерности. Для уменьшения размерности мы решили протестировать как автокодировщик, так и метод главных компонент (PCA). Отметим, что новые признаки, полученные после уменьшения размерности, также нуждаются в стандартизации для дальнейшего

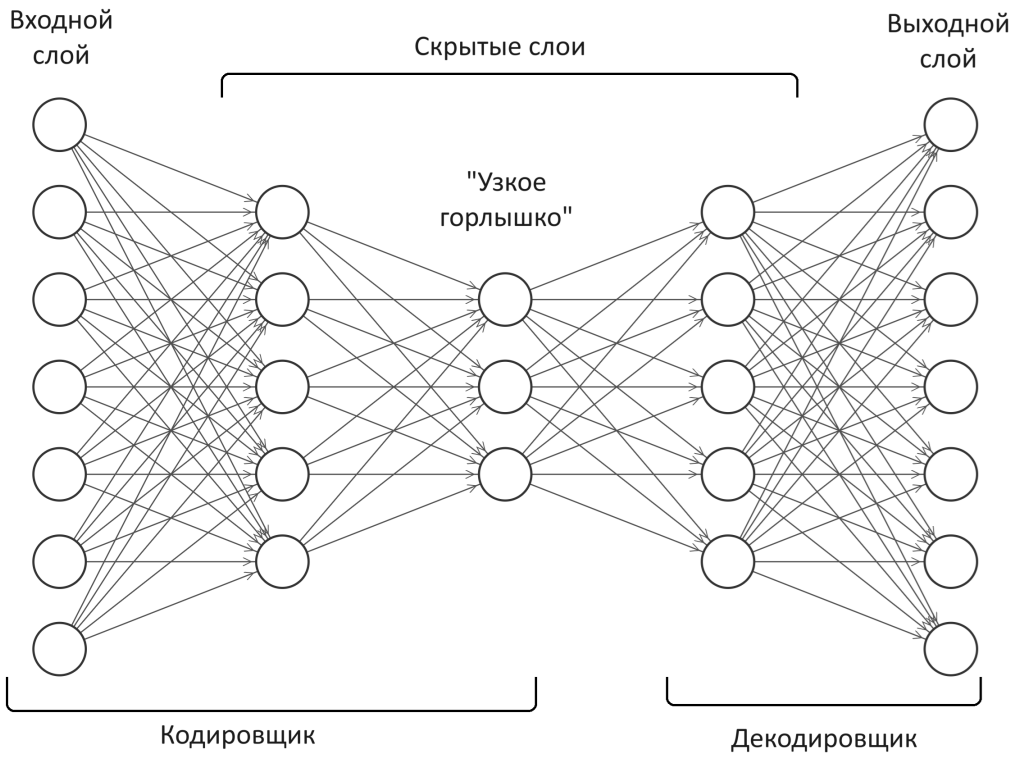

Рисунок 3.12 — Схема архитектуры нейронной сети автокодировщика.

сравнения, поскольку они могут иметь разную амплитуду, а признаки могут вносить очень неравномерный вклад в меру расстояния между точками.

Были рассмотрены два варианта уменьшения размерности для каждого из методов — уменьшение размерности с исходных 15 характеристик до 3 и 5. Данные значения были выбраны из соображения уменьшить количество характеристик в 5 и 3 раза соответственно.

Для обучения как PCA, так и автокодировщика использовались данные системы мониторинга за полгода работы суперкомпьютера. Для обучения очень важно, чтобы данные были сбалансированные, то есть не было случаев, когда один тип поведения по количеству перебивает все остальные. Для решения данной проблемы было предпринято следующее:

– Каждая точка (показания динамических характеристик в определенный момент времени) была преобразована при помощи подобранных методов, описанных в предыдущем подразделе. Это позволит нам их сравнивать между собой.
– Из обучающей выборки выделяются только те точки, которые не схожи между собой. Схожесть определяется с помощью порогового значения,

значение которого определилось в предыдущем подразделе при подсчете наилучшей точности.

– Выделенные точки временных рядов и будут новой обучающей выборкой.

После обучения методов, были получены готовые модели, которые можно применять для уменьшения размерности. Но, как и с исходными данными, результат уменьшения размерности необходимо преобразовать, чтобы каждая новая размерность имела равный вклад в оценку расстояния. Так как стандартизация показала себя лучшим образом в предыдущем подразделе, было принято решение и здесь использовать этот метод преобразования. Результат работы методов показан в таблице 2. Как можно заметить, PCA показывает близкие к исходным результатам. Автокодировщик же показывает результаты сильно лучше, чем остальные два метода, и в дальнейшем планировалось использовать данный метод. В случае с автокодировщиком также можно заметить, что увеличение размерности хоть и немного, но влияет на результат работы в лучшую сторону.

Таблица 2 — Сравнение результатов работы динамического метода с и без использования методов уменьшения размерности

	Автокодировщик		PCA		Без уменьшения размерности
	разм. 3	разм. 5	разм. 3	разм. 5	
Точность	0.9	0.92	0.83	0.86	0.85

В результате, имеется 2 способа применения динамического метода — с и без использования методов уменьшения размерности, но из-за более высокого качества работы метода с автокодировщиком предпочтение дается именно ему. Но несмотря на это, все же существуют пары заданий, в которых метод некорректно определил схожесть, и данные случаи необходимо рассмотреть.

3.3.4 Выяснение причин некорректного определения схожести

Для дальнейшего улучшения точности необходимо было определить, почему в некоторых случаях динамический метод некорректно определял или не

определял схожесть пар заданий. Такой анализ неприменим к методам уменьшения размерности, так как зачастую данные методы используются как черный ящик, и мы мало можем влиять на вывод данных методов.

При анализе ошибок динамического метода без уменьшения размерности было выявлено, что некоторые характеристики, несмотря на стандартизацию, все равно имеют слишком большой вклад в итоговое расстояние между точками, из-за чего оценка расстояния DTW зачастую сильно завышается. Это относится к таким характеристикам, как загрузка CPU и GPU, а также значение load average. Стало понятно, что у этих характеристик необходимо вручную понижать влияние с помощью масштабирования — преобразования вида $\widetilde{x_i} = x_i * C$, где C — фиксированное значение (константа). Данное преобразование необходимо для того, чтобы уменьшить или увеличить расстояние между точками определенных динамических характеристик. Таким образом, на данном шаге делался подбор таких C_i для каждой из проблемных динамических характеристик.

Сам процесс подбора параметров был таким:

- Ручное изменение коэффициентов C_i в зависимости от тех пар заданий, на которых динамический метод некорректно определил схожесть.
- Вычисление точности динамического метода с новыми коэффициентами.
- Анализ пар заданий, у которых вновь некорректно была определена схожесть.

После достаточно малого числа итераций, был достигнут некоторый потолок по точности, улучшить который не удавалось путем изменения коэффициентов C_i. Итоговая точность составила 0.95, что близко к результатам автокодировщика. Коэффициент для загрузки CPU равен 0.4, для загрузки GPU и LoadAVG — 0.33.

Следующий необходимый шаг — проверка модели на отсутствие переобучения. Переобучение — феномен, когда чрезмерная оптимизация метода на обучающем наборе приводит к тому, что на другом наборе данных метод работать будет сильно хуже. Для дополнительной проверки качества работы были размечены дополнительные 100 пар заданий, и на этих 100 парах заданий точность составила 0.93. Это показывает, что метод хорошо переносится на задания, отличные от обучающей выборки, и может использоваться на практике.

В конце стоит отметить еще один момент. Последнее преобразование с применением масштабирования существенно улучшило точность работы метода, поэтому возник вопрос в актуальности применения метода автокодировщика. После его переобучения выяснилось, что результаты по точности улучшились незначительно. В связи с чем было принято решение не использовать в дальнейшем автокодировщик, так как он не только не дает значительного прироста по точности, но также является “черным ящиком”, что сильно усложняет дальнейшую интерпретацию результатов его вывода.

Глава 4. Приложения разработанных методов обнаружения схожих приложений в практике работы суперкомпьютерного центра

4.1 Предсказание метрик производительности приложений

В настоящее время сотрудники Научно-исследовательского вычислительного центра МГУ разрабатывают систему оценок для анализа качества использования ресурсов суперкомпьютера [A4][34]. Эти оценки необходимы для предварительной оценки производительности приложений и выявления задач с низкой эффективностью, требующих внимания. Последующий детальный анализ приложения для выявления причин снижения производительности и способов их устранения предполагается осуществлять с использованием имеющихся средств анализа, таких как профилировщики и отладчики.

Оценки были разработаны для 6 типов ресурсов — центральный процессор ($\text{score}_{\text{cpu}}$), память ($\text{score}_{\text{mem}}$), сеть MPI ($\text{score}_{\text{mpi}}$), сеть ввода-вывода ($\text{score}_{\text{fs}}$), графический процессор ($\text{score}_{\text{gpu}}$) и его память ($\text{score}_{\text{gpu mem}}$), и были предложены конкретные формулы для их расчета. Для получения необходимых данных, требуемых для вычисления оценок, использовалась ранее упоминаемая система мониторинга. Однако для вычисления $\text{score}_{\text{cpu}}$ (4.1) и $\text{score}_{\text{mem}}$ (4.2) необходимо собирать дополнительные характеристики (другие данные с процессорных датчиков), для чего потребовалось использовать режим мультиплексирования, т.е. переключения между наборами собираемых характеристик. Данный режим привнес дополнительные накладные расходы, что в среднем увеличило время исполнения заданий на 2.78% [35].

$$\mathbf{score_{cpu}} = 100 * (1 - \frac{uops\_retired.retire\_slots}{2 * cpu\_clk\_unhalted.thread\_any}) \tag{4.1}$$

$$\begin{aligned}\mathbf{score_{mem}} = (&min(cpu\_clk\_unhalted.thread,\\ &cycle\_activity.stalls\_ldm\_pending) +\\ &resource\_stalls.sb) \,/\, cpu\_clk\_unhalted.thread\end{aligned} \tag{4.2}$$

Такие дополнительные накладные расходы, хотя и не критически важны, все же ощутимы. В связи с этим было принято решение запускать сбор данных

с дополнительных датчиков только для каждого третьего задания. Это позволяет снизить общие накладные расходы до уровня менее 1%. Однако возникает вопрос оценки эффективности использования ресурсов суперкомпьютера для остальных двух третей заданий. Для решения данной проблемы предлагается использовать методы выделения схожих приложений.

4.1.1 Описание предлагаемого метода предсказания

Для решения данной задачи было принято решение использовать методы выделения схожих суперкомпьютерных заданий. Основная идея схожа с решением задачи выделения программных пакетов: используется база знаний, в который включены задания с известными значениями оценок, и все новые задания сравниваются с заданиями из базы знаний.

Таким образом, предлагается использовать следующий алгоритм действий:

- Собирается база знаний, в которую включаются ранее выполнившиеся задания, для которых известны значения оценок.
- При появлении нового задания, для которого оценки не были посчитаны, выполняется следующее:
  - Проводится поиск схожих заданий в базе знаний с помощью статического метода.
  - При нахождении схожих заданий с помощью статического метода, среди них проводится поиск схожих заданий с помощью динамического метода с более грубым порогом (шаг А).
  - При отсутствии схожих заданий, выделенных с помощью статического метода, проводится поиск схожих заданий в базе знаний с помощью динамического метода с более строгим порогом (шаг Б).

Идея использования шагов А и Б заключается в предположении, что при нахождении статически схожих заданий вероятность того, что в результате динамического анализа несхожие задания будут ошибочно определены как похожие, мала, и поэтому есть возможность использовать более грубый порог, чтобы получить больше заданий для сравнения без потери точности.

После применения данного вышеуказанного алгоритма, будет получен набор заданий, которые схожи с новым. Для каждого схожего задания известны значения оценок, и здесь возникает вопрос: как использовать данные оценки для предсказания? Было рассмотрено 3 варианта: вычисление среднего значения, медианы и средневзвешенного значения softmax, формула для которого показано в 4.3.

$$\text{Softmax score}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} * x_i \tag{4.3}$$

4.1.2 Настройка и предварительная апробация предложенного метода

Одними из важных параметров работы предложенного метода являются пороговые значения для статического и динамического методов, по которым определяется схожесть двух заданий. При апробации описанных ранее методов выделения схожих заданий нами были определены данные пороги, и в данной задаче они хорошо подходили, из-за чего новые пороги не подбирались.

Для апробации предложенного метода необходимо определить, что в нашем случае считается хорошим качеством работы метода. Из-за того, что данная задача предсказания оценок — это задача регрессии, будут использоваться метрики оценки качества работ задач регрессии. Были выбраны 2 метрики — средняя абсолютная ошибка (MAE) и среднеквадратичная оценка(MSE). Данные метрики — самые распространенные метрики оценки качества регрессионных методов, так как с их помощью можно хорошо сравнить результаты работы разных вариантов регрессионных моделей. Формулы метрик представлены в 4.4 и 4.5, где y_j —- предсказанное, а $\widehat{y}_i$ — истинное значение. Обе метрики принимают положительные значения, и чем ближе значение к 0, тем лучше. MAE показывает то, на сколько в среднем регрессионная модель ошибается в предсказаниях, и при подсчете данной метрики все ошибки имеют одинаковый вес. Этот показатель хорошо демонстрирует качество работы метода, но у него есть недостаток: если небольшой процент предсказаний будут иметь большое значение ошибки, значение метрики MAE не будет сильно меняться. Для этого мы также рассматриваем ошибку MSE, в которой величина ошибки возводится в квадрат. Это означает, что предсказания с большим показателем ошибки

будут вносить большой вклад в итоговое значение метрики, и получается, что чем больше значение метрики, тем больше процент заданий с большим показателем ошибки. Стоит отметить, что изначально задача для метода предсказаний оценок ставилась так, чтобы средняя ошибка составляла <10 (Это диктуется свойствами самих оценок).

$$\mathbf{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \widehat{y}_j| \tag{4.4}$$

$$\mathbf{MSE} = \frac{1}{n}\sum_{j=1}^{n}(y_j - \widehat{y}_j)^2 \tag{4.5}$$

Для первичной оценки качества работы метода были рассмотрены все задания с известными значениями оценок за 4 месяца работы суперкомпьютера Ломоносов-2. В данный период число таких заданий составило ∼1500. Идея в том, что для каждого задания делаются предсказания оценок, которые затем сравниваются с истинными значениями, полученными при работе задания. При предсказании производился поиск схожих заданий среди всех доступных заданий с оценками (каждый с каждым).

Для оценки качества работы было предложено проанализировать 3 подхода на основе алгоритма, который был описан в предыдущем разделе:

– Комбинированный подход №1, в котором используются оба шага (А и Б).
– Комбинированный подход №2, в котором используется только шаг А.
– Только динамический метод, в котором используется только шаг Б.

Результат работы вышеуказанных трех подходов и трех вариантов агрегации показаны в таблице 3. На данном этапе изучалась только оценка $score_{cpu}$. При рассмотрении трех подходов, можно заметить, что использование только динамического метода позволяет получить предсказание оценок для большего процента заданий, но все показатели MAE и MSE сильно хуже, чем у других двух версий. Ключевой показатель — это процент заданий с высоким показателем ошибки, и у версии с только динамическим анализом процент таких заданий составляет 8-10%, что недопустимо при использовании в реальной системе. Если рассматривать комбинированные подходы, то можно заметить, что комбинированный подход №2 имеет показатели MSE и MAE чуть хуже, чем у

Таблица 3 — Данные по метрикам оценки качества работы метода для трех версий алгоритма и трех вариантов агрегации результатов, $score_{cpu}$

Подход	% заданий с предсказанной оценкой	MAE	MSE	% заданий с AE больше 10
Комбинированный подход №1, Softmax	56.62	1.39	7.42	0.65
Комбинированный подход №2, Softmax	33.06	1.66	8.12	0.47
Только динамический анализ, Softmax	80.53	3.71	37.92	8.01
Комбинированный подход №1, median	56.62	1.35	7.88	0.95
Комбинированный подход №2, median	33.06	1.68	9.09	0.71
Только динамический анализ, median	80.53	3.89	50.96	8.9
Комбинированный подход №1, average	56.62	1.48	8.06	0.65
Комбинированный подход №2, average	33.06	1.77	8.79	0.47
Только динамический анализ, average	80.53	4.12	43.81	9.61

комбинированного подхода №1, но имеет чуть меньший процент заданий с высоким показателем ошибок. Но данные отличия настолько незначительные, что можно считать их практически идентичными с точки зрения оценок качества работы, и в таком случае выбор оптимального подхода строится исключительно на том, какой подход может предсказать больше оценок. И в данном случае подход №1 явно берет вверх, поскольку смог предсказать на 70% больше заданий, чем подход №2. Это хорошо видно на рисунке 4.1, где каждый столбец соответствует определенному интервалу разницы между истинными и прогнозируемыми оценками, а его высота показывает, сколько заданий попадают в указанный интервал разницы. Комбинированные подходы имеют очень схожее распределение заданий с большим показателем ошибок, но подход №1 имеет сильно больше заданий с точными предсказаниями. В связи с этим было принято решение использовать комбинированный подход №1 в дальнейших тестах.

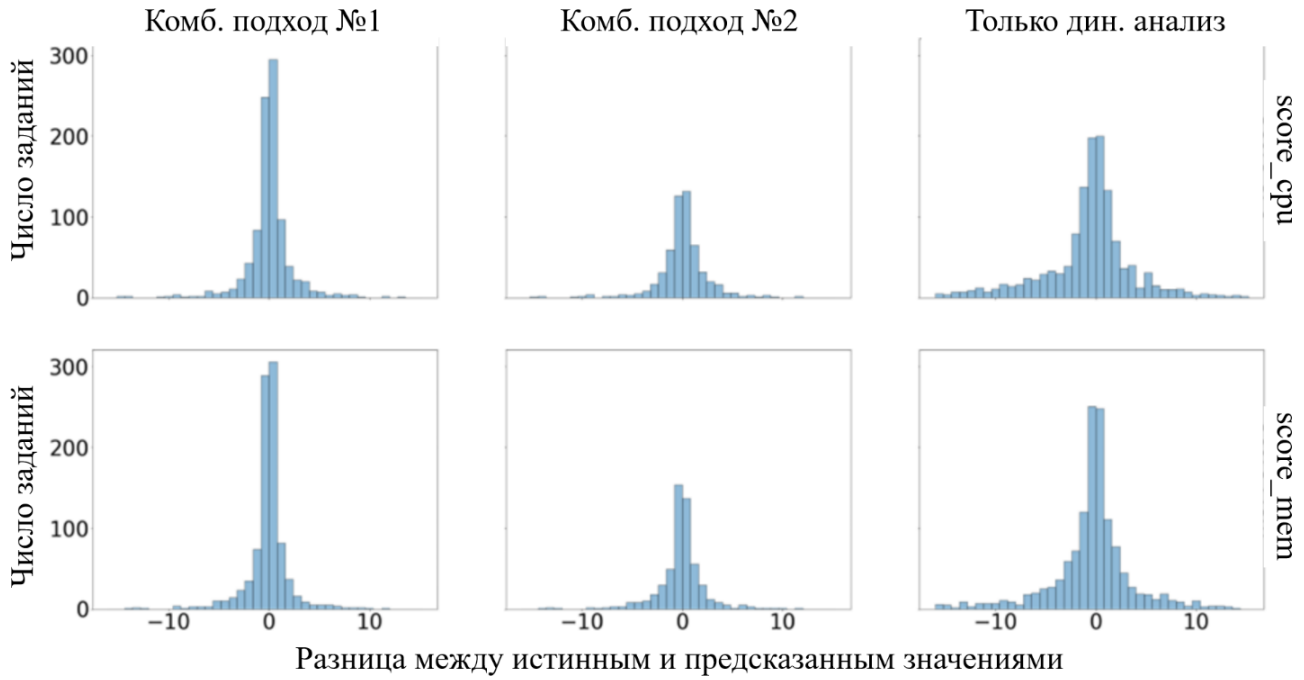

Рисунок 4.1 — Гистограмма распределения числа заданий с полученной разницей между истинным и предсказанным значениями оценок $score_{cpu}$ и $score_{mem}$, для трех рассмотренных подходов

Если рассматривать варианты агрегации, то можно также выделить явный лучший вариант. Будем рассматривать значения только для комбинированного подхода №1. Видно, что вариант со средним значением имеет худшие показатели MSE и MAE среди всех трех вариантов. Версия с медианой имеет лучший показатель MAE. Но явно выделяется версия с softmax: он имеет

лучшие показатели по MSE и % заданий с высоким показателем ошибок, и незначительно хуже показатель метрики MAE, чем у версии с медианой. Из-за этого было принято решение использовать в дальнейшем для агрегации вариант с softmax функцией. Но несмотря на это, все равно стоит отметить, что разница между вариантами не так велика. Это хорошо видно на рисунке 4.2, где для трех вариантов агрегации показано распределение разницы между предсказанными и истинными значениями оценок для заданий.

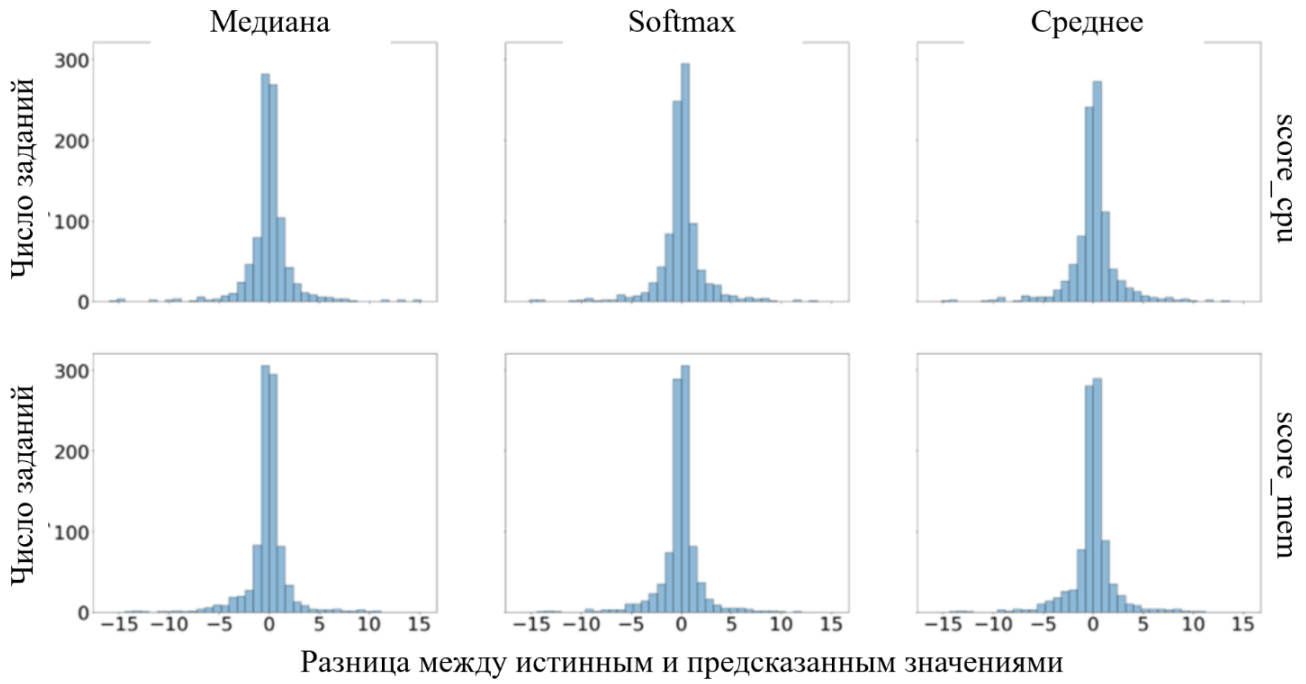

Рисунок 4.2 — Гистограмма распределения числа заданий с полученной разницей между истинным и предсказанным значениями оценок $score_{cpu}$ и $score_{mem}$, для трех методов агрегации

4.1.3 Оценка точности предложенного метода на реальных данных

На момент предварительной апробации, были доступны оценки для заданий за 4 месяца работы суперкомпьютера. После еще 4-х месяцев, было принято решение апробировать метод с учетом новых данных. В итоге было рассмотрено ∼4600 заданий. Был использован ранее выбранный комбинированный подход №1 и агрегация с использованием средневзвешенного значения softmax.

При данной апробации было рассмотрено 2 варианта использования предложенного метода предсказания. Первый вариант аналогичен описанному в предыдущем пункте: поиск схожих заданий производился среди всех доступных заданий (в том числе более новых по сравнению с рассматриваемым заданием). Это позволяет получить наибольшее количество схожих заданий, что, в свою очередь, позволяет получить больше заданий с предсказанными оценками. Второй вариант использования — это симуляция онлайн работы алгоритма, при котором предсказание оценок выполняется сразу после завершения задания, из-за чего доступны данные только о предшествующих заданиях, причем в целях ускорения работы метода рассматриваются только последние N заданий (значение N подбирается в зависимости от условий, в которых метод работает).

Начнем с апробации первого варианта, так как с его помощью можно получить наиболее точную картину о том, насколько хорошо работает метод. В таблице 4 приведены результаты работы метода при доступности всех данных. Как можно заметить, в среднем работа метода показывает удовлетворяющую требованиям качество работы: средняя ошибка составляет 2.62. Гистограмма распределения разницы между предсказанным и истинным значениями оценок показана на рисунке 4.3.

Таблица 4 — Результат апробации для варианта с использованием всех доступных заданий

	% заданий с предсказанной оценкой	MAE	MSE	% заданий с AE больше 10
$score_{mem}$	86.94	2.62	34.54	4.71
$score_{cpu}$	86.93	2.19	23.21	3.27

Можно также заметить, что все показатели метрик оценки качества работы метода ухудшились по сравнению с результатами из предыдущего раздела. Главное предположение, почему такое произошло — увеличение доступного для анализа количества данных. Оно увеличилось в 3 раза с момента предварительной апробации. Это дало большой прирост в количестве заданий, для которых удалось предсказать: с 56.62% до 86.93%. Но можно сделать вывод, что качество новых предсказаний ниже.

Первый рассмотренный вариант — идеальный случай, когда время обработки задания не учитывается. Второй вариант является более приближенным

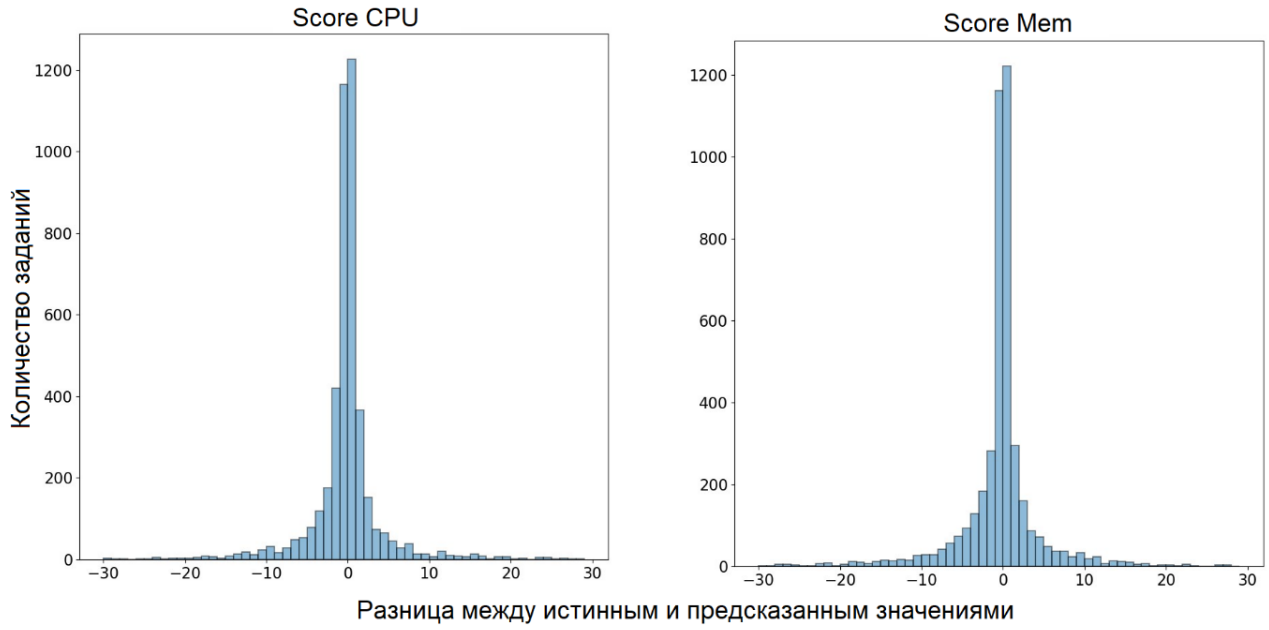

Рисунок 4.3 — Гистограмма распределения числа заданий с полученной разницей между истинным и предсказанным значениями для оценок $score_{cpu}$ и $score_{mem}$

к реальности, так как зачастую есть потребность в получении данных о задании сразу после его завершения. При обработке потока суперкомпьютерных заданий в режиме реального времени искать схожие задания среди всех предыдущих заданий не является возможным, поскольку это может потребовать слишком много времени. В связи с чем возникает вопрос: сколько предыдущих заданий необходимо учитывать? Это количество заданий для сравнения в дальнейшем будем называть “окном”.

Для решения задачи подбора размера окна был проведен эксперимент, в котором оценивалось качество работы метода в зависимости от размера окна. На рисунке 4.4 показан графики изменения MAE и процента заданий, для которых удалось предсказать значение оценки, в зависимости от размера окна. Как можно заметить, размер окна от 1250 до 1500 кажется лучшим выбором для комбинации метрик, так как дальнейшее увеличение размера практически не дает никакого прироста к качеству работы: количество заданий с предсказаниями не растет, и при этом MAE также немного, но все же увеличивается.

В таблице 5 приведены результаты работы метода при размере окна, равным 1250.

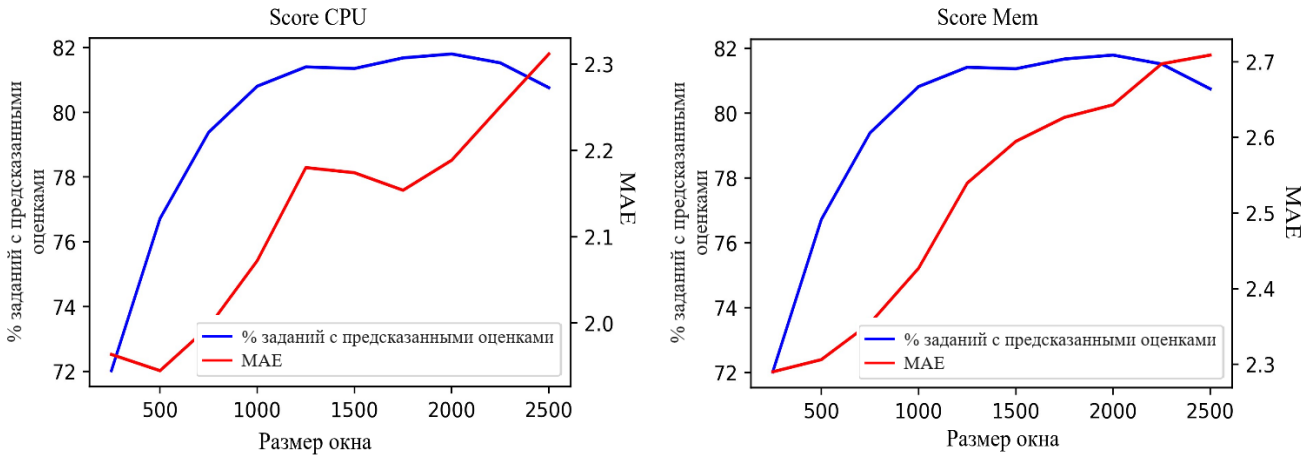

Рисунок 4.4 — Графики изменения MAE и процента заданий, для которых удалось предсказать значение оценки, в зависимости от размера окна

Таблица 5 — Результат апробации для случая с использованием только предшествующих заданий, размер окна=1250

	% заданий с предсказанной оценкой	MAE	MSE	% заданий с AE больше 10
$score_{mem}$	81.41	2.53	32.59	4.51
$score_{cpu}$	81.39	2.18	24.83	3.68

Если сравнивать два варианта апробации, то можно заметить, что учет не только предшествующих, но и будущих заданий не дает никакого прироста к качеству предсказаний, но дает возможность предсказать оценки для еще 5% заданий, из-за чего предлагается использовать следующий подход при анализе реального потока суперкомпьютерных заданий:

– сразу после завершения задания выполняется предсказание оценки с использованием второго варианта апробации;
– если предсказать оценку не удалось, через N дней проводится перепроверка с учетом новых завершившихся заданий. Число N может варьироваться.

4.2 Обнаружение программных пакетов

В данном и последующих разделах рассмотрим практически важные задачи, для решения которых применялись предложенные ранее методы обнаружения схожих приложений. Основной задачей данного раздела является разработка и анализ качества работы методов по обнаружению программных пакетов, используемых в заданиях суперкомпьютеров. Под программным пакетом будем подразумевать набор взаимосвязанных программных модулей, предназначенных для решения определенных прикладных задач некоторой предметной области. Примерами программных пакетов являются LAMMPS [36], NAMD [37], Firefly [38] и т.д.

Обнаружение пакетов позволяет получить подробную информацию о том, как используется суперкомпьютер (например, какие предпочтения у пользователей). Это поможет помочь пользователям, например, путем обнаружения и оптимизации наиболее часто используемых пакетов для конкретной вычислительной системы. С помощью методов обнаружения пакетов можно также понять, какие пакеты не используются, что позволит определить, какие лицензии можно не продлевать. Если расширить задачу обнаружения пакетов до обнаружения их версий, можно выделить часто используемые версии пакетов, которые не установлены централизованно — после установки данных версий пакетов, пользователям не будет надобности заново собирать данный пакет, и будет уверенность в том, что данная версия пакета собрана корректно и эффективно сконфигурирована.

“Ломоносов-2” имеет существующую систему обнаружения пакетов, основанную на системе XALT [39]. XALT заменяет команды компоновщика (ld) и запуска (mpirun), что позволяет получить информацию о том, где находятся все исполняемые и объектные файлы, сколько исполняемых файлов запущено, как было скомпилировано приложение и т. д. Путем поиска конкретных ключевых слов в путях к файлам можно определить, какие библиотеки и программные пакеты используются. Но у этого подхода есть недостаток. При динамической компоновке XALT может обнаруживать наличие определенных библиотек, так как имена скомпонованных файлов часто включают определенные ключевые слова, а пользователя зачастую не переименовывают скомпонованные файлы. Но при статической компоновке пользователь может переименовать исполняе-

мый файл, и XALT уже не сможет определить наличие той или иной библиотеки или пакета программного обеспечения. Также XALT может случайно найти ключевые слова одного пакета в путях к заданию с другим пакетом, что некорректно. Несмотря на то, что данные случаи возникают редко на практике, метод без данного недостатка хорошо бы дополнял существующую систему.

Перед тем, как приступать к адаптации наших методов выделения схожих для решения задачи обнаружения пакетов, было необходимо изучить, какие другие работы существуют в этой области. Так, в статье [40] содержится обзор того, как обнаружение пакетов программного обеспечения построено в Суперкомпьютерном центре Огайо. Авторы используют комбинацию pbsacct [41] и XALT для анализа запущенных заданий. XALT используется так же, как и на суперкомпьютере "Ломоносов-2". Pbsacct — это система анализа потока заданий для кластеров высокопроизводительных вычислений, использующих Portable Batch System (PBS). В нем хранится информация о сценариях заданий, что позволяет обнаруживать используемые пакеты, путем сопоставления сценариев с образцом. Этот подход очень похож на то, что делает XALT, но в нашем случае мы не можем использовать pbsacct, потому что "Ломоносов-2" использует Slurm вместо PBS. Кроме того, pbsacct имеет те же проблемы, что и XALT, потому что он зависит от ключевых слов, присутствующих в сценариях заданий. Других работ, направленных на обнаружение использования программных пакетов в HPC, найдено не было.

4.2.1 Решение задачи выделения программных пакетов с помощью статического метода выделения схожих приложений

В данном разделе описывается настройка и адаптация метода, предложенного в разделе 2.1, для решения проблемы обнаружения программных пакетов. Адаптация данного алгоритма для получения информации об использовании пакетов является достаточно простым процессом. Для этого необходимо создать базу знаний, содержащую информацию об исполняемых файлах, для которых известны программные пакеты, используемые в них. Такую информацию предоставляет система XALT, установленная на суперкомпьютере "Ломоносов-2". Конечно, есть некоторая вероятность того, что XALT может выдать непра-

вильные данные, но на практике такие случаи встречаются редко. Это было подтверждено после ручной проверки множества заданий в базе знаний на наличие использования того или иного пакета. В настоящее время база знаний, собранная на суперкомпьютере “Ломоносов-2”, содержит информацию о 20 пакетах, и для каждого пакета имеются уникальные примеры исполняемых файлов. Количество примеров для часто используемых пакетов достаточно велико (более 50), в то время как для редко используемых может быть меньше 10, однако со временем база данных будет расширяться.

После того как задание начало исполняться, проводится его сравнение с базой знаний заданий. Результатом этого сравнения является оценка степени схожести текущего задания со всеми заданиями из базы знаний. Выбор ближайших заданий из базы знаний осуществляется при помощи установленного порога, определенного в разделе апробации статического метода выделения схожих приложений. Если среди ближайших заданий обнаружено несколько различных пакетов, то предполагается, что все эти пакеты используются в новом задании.

Для оценки эффективности и точности данного метода было проведено масштабное тестирование на суперкомпьютере “Ломоносов-2”. В период с начала октября по середину ноября 2020 года было обработано более 3600 заданий, и с помощью XALT было выявлено более 2600 заданий, использующих программные пакеты. Для упрощения проверки анализировались задания, в которых система XALT выявила использование только одного программного пакета. После сравнения результатов предложенного метода с системой XALT, была получена точность 0.77. Точность рассчитывалась по формуле 4.6, где A — количество заданий, в которых как статический метод, так и XALT обнаружили использование программных пакетов, а B — количество заданий, в которых ни статический метод, ни XALT не выявили использование программных пакетов.

$$\text{Точность} = \frac{A + B}{\text{общее количество заданий}} \tag{4.6}$$

Случаи, когда наш метод обнаружил пакет, а XALT — нет, были проанализированы вручную. После выполнения ручной проверки было выявлено более двухсот заданий (8% от общего числа), в которых статический метод обнаружил наличие пакетов, которые не были выявлены другим методом, и последующая ручная проверка подтвердила присутствие этих пакетов в указанных заданиях.

Указанные двести заданий использовали около пятнадцати различных пакетов и запускались примерно сорока уникальными пользователями.

Противоположный случай, когда XALT обнаружил пакет, а наш метод — нет, рассматривался как ошибочная классификация, поскольку вероятность того, что XALT неправильно обнаружит пакет, мала. Эти ложные классификации составляли 15% общего числа случаев, что побуждает проанализировать случаи, когда наш статический метод не обнаруживает пакеты, хотя они действительно использовались в данных заданиях (таких случаев было около 350). Существует несколько причин для этого. Во-первых, иногда метод не способен извлечь информацию из исполняемых файлов из-за ограничений доступа к этим файлам или изменений в них до момента анализа (например, перекомпиляции приложения). Во-вторых, это может происходить при появлении новых имен функций в исполняемых файлах, которые модель Doc2Vec не учитывала в процессе обучения и игнорирует их. Эта проблема решается периодическим переобучением модели с добавлением в обучающую выборку новых исполняемых файлов. В-третьих, отсутствие записей о пакете в базе знаний или изменения в процессе компиляции могут привести к значительным различиям в содержимом исполняемого файла, что делает его непохожим на известные примеры в модели классификатора Doc2Vec.

Третья проблема решается регулярным обновлением базы знаний новыми заданиями. В частности, добавление новых записей в базу знаний уже значительно улучшило производительность нашего метода. В последнюю неделю ноября 2020 года было запущено около 300 заданий, для которых точность извлечения пакетов увеличилась до 0.9. За эту неделю было выявлено более 10 случаев, когда наш метод обнаруживал пакеты, а XALT не мог.

Пример работы метода можно хорошо продемонстрировать с помощью рисунка 4.5. На рисунке показано распределение уникальных исполняемых файлов, созданных с помощью метода t-SNE [42]. Алгоритм t-SNE применяется для снижения размерности векторов высокой размерности, где аналогичные объекты представлены соседними точками в пространстве низкой размерности, и зачастую применяется для визуализации. Метки точек предоставлены системой обнаружения пакетов на суперкомпьютере “Ломоносов-2” на базе XALT. Результаты показывают много зеленых точек, обозначающих задания без обнаруженных пакетов. Несмотря на то, что точки в центре довольно хаотичны, можно четко выделить несколько крупных скоплений. Группа 1 (отмечена крас-

ным кружком слева) имеет только одну помеченную работу с пакетом Cabaret, который XALT не смог идентифицировать в других заданиях, в то время как предложенный статический метод хорошо справляется с этой задачей и разметил их соответствующим образом. То же самое можно сказать и о пакетах LAMMPS (светло-зеленый) и GROMACS (темно-синий). Это показывает, что вектора фиксированной длины хорошо инкапсулируют информацию о пакетах, и их использование для выделения пакетов более чем оправданно.

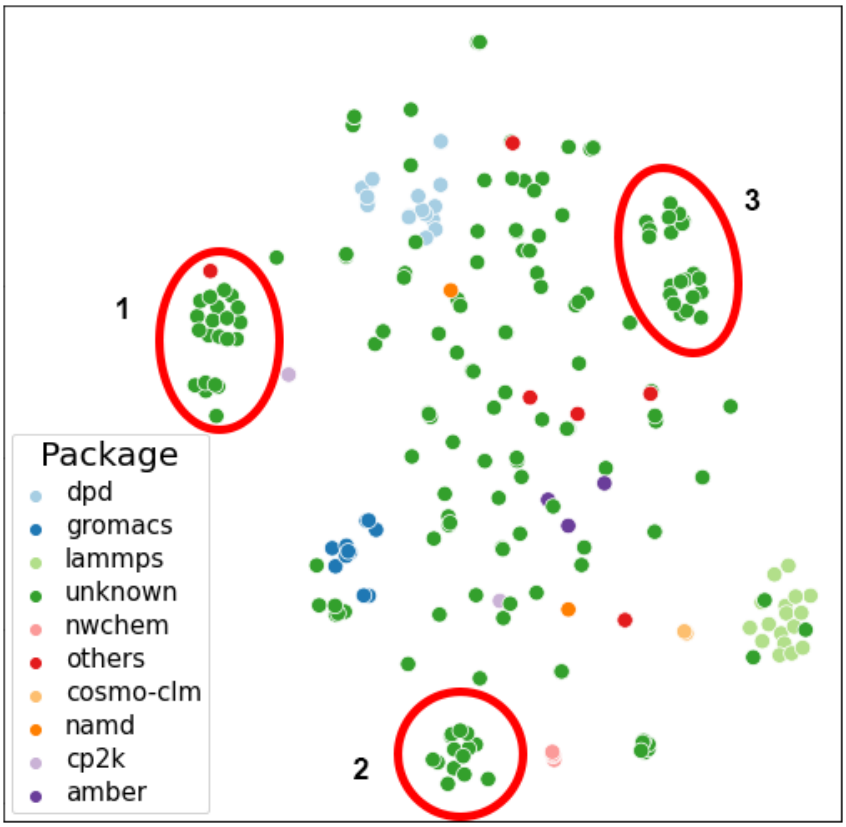

Рисунок 4.5 — Распределение уникальных исполняемых файлов, созданных с использованием t-SNE. Выделенные кластеры: 1 — Cabaret, 2 — XAMG, 3 — неизвестно.

Для информирования администраторов суперкомпьютеров о результатах статического метода ежедневно отправляется сводный отчёт по электронной почте (рисунок 4.6). Каждая строка в отчёте соответствует одному заданию, и окрашивается в зелёный цвет, если наш метод обнаружил пакеты, но XALT не обнаружил, и в красный цвет, если происходит обратное. Отчёт позволяет быстро определить случаи, когда применение того или иного метода обнаружения пакетов дало результат. Также отчёт полезен для выявления неправильной классификации пакетов статическим методом и принятия мер по их исправлению для улучшения точности работы предложенного решения.

Дайджест по пакетам (Lomonosov-2) Inbox ×

Stat2 <info@stat2.parallel.com> 8:49 PM (3 minutes ago)
to me

Дайджест по пакетам за 09-06-2022

Найдено пакетов 124/190

Job id	unique	doc2vec	xalt	Comment
1417940		cp2k	cp2k	nm: /home/hmd_2177/bin/ompi-hmd: File format not recognized
1417956		firefly	firefly	srun only | nm: /mnt/scratch/users/natasha_81677/firefly/qdpt_oxy.sh: File format not recognized
1418036		firefly		srun only | nm: /mnt/scratch/users/aag_81673/ff/qdpt_oxy.sh: File format not recognized
1418463	lammps	lammps		srun only | [Errno 13] Permission denied: '/mnt/scratch/users/glagoleva/surface-polycation/sample2/pc-512-ch-N1-48-Nch-12-12-4z-4y-f_1_2/adsorbed_chains-new/run2/start.sh'
1419392		firefly		nm: /mnt/scratch/users/mikhailnickkh_1704/FF/ff_run: File format not recognized
1419484		lammps	lammps	
1418171			gromacs	[Errno 13] Permission denied: '/mnt/scratch/users/vvvas_2123/2.64L_BLENDS_2/N8/3.64L_4LD/0.64L_4LD_0.125/run_gromacs_2.sh'
1419269			nwchem	nm: /mnt/scratch/users/grigorenko/nwchem/flavin/batch.sh: File format not recognized
1419272			rosetta	probably a python script | [Errno 2] No such file or directory: 'mnt/scratch/users/golovin_2251/progs/lom_tools/run_multiple_tasks.py' | /usr/bin/python2.7: hashes didn't match | srun only
1419278				[Errno 2] No such file or directory: 'mnt/scratch/users/alexsetyaev_2248/ompi_with_top.sh'

Рисунок 4.6 — Пример ежедневного отчета со случаями обнаружения пакетов статическим методом.

4.2.2 Решение задачи выделения программных пакетов с помощью динамического метода выделения схожих приложений

Для решения задачи обнаружения пакетов также можно применить описанный в разделе 2.2 динамический алгоритм, аналогично уже реализованному статическому методу, описанному в предыдущем разделе. Новое задание, которое мы хотим проанализировать, сравнивается с уже выполненными заданиями за некоторый предыдущий период времени для выявления наиболее близких по поведению заданий. Как и в случае со статикой, будет считать, что если новое

задание близко к некоторым ранее изученным, то в нем используются те же пакеты, что были обнаружены в этих близких известных заданиях.

Для сравнения заданий используется динамический метод, который предоставляет оценку расстояния между заданиями. Аналогично статическому методу, будет использоваться факт схожести двух заданий, то есть можно переиспользовать порог из раздела 2 про апробацию динамического метода на вручную размеченных данных, по которому определяется схожесть.

В нашем случае мы сравниваем с заданиями, выполненными в течение последних 1-2 месяцев. Увеличение рассматриваемого периода заданий затруднено из-за высокой вычислительной сложности использованного метода. Тем не менее в процессе проведения экспериментов мы установили, что для нашего случая этого периода достаточно.

Таким образом, алгоритм работы предлагаемого метода следующий:

– Строится база знаний за предыдущие 1-2 месяца работы суперкомпьютера. База знаний состоит из пары {динамические характеристики : используемые пакеты в задании}. Данные об использовании пакета извлекаются из XALT и статического метода.
– После завершения нового задания, производится поиск схожих заданий из базы знаний с использованием динамического метода. Длительность нового задания должна быть больше одного часа для получения достаточного объема динамических данных по заданию.
– Если схожие задания из базы знаний были найдены, то считается, что в новом задании используются те же программные пакеты, что и в схожих заданиях.

Для проверки работоспособности было принято решение провести масштабное тестирование. Были отобраны все задания, запущенные в октябре и ноябре 2020 года. Общее количество заданий, включенных в данное тестирование, составило более 3300. Результаты эксперимента представлены в таблице 6. Под другими методами в данной таблице подразумеваются результаты работы XALT и статического метода выделения пакетов. Считается, что пакет был обнаружен, если по крайней мере один из двух других методов обнаружил использование пакета.

После сравнения полученных результатов, была получена точность 0.81. Точность считалась аналогично тому, как она считалась для статического метода: по формуле 4.6, только в данном случае A — количество заданий, в которых

и динамический, и другие методы обнаружили использование программных пакетов; B — количество заданий, в которых ни динамический, ни другие методы не обнаружили использование программных пакетов. Интересно отметить, что динамический метод выделил больше заданий с пакетами по сравнению с другими методами, чем статический метод по сравнению с XALT. Причиной этому может быть отсутствие статических данных у некоторых заданий, в связи с чем статический метод не смог выделить использование пакетов.

Все случаи, где методы не совпали, были подвергнуты ручному анализу. Следует отметить, что в большинстве случаев сложно вручную определить, действительно ли использовались пакеты, анализируя только данные системы мониторинга и планировщика заданий, но все же иногда бывают случаи, очевидные для администраторов. Было выявлено, что по крайней мере в 112 заданиях динамический метод правильно определил использование пакета (использовались 3 разных пакета), в то время как другие методы ничего не смогли обнаружить. В остальных случаях требуется дополнительный детальный анализ и возможно обращение к пользователям для уточнения используемых ими пакетов.

Таблица 6 — Результаты сравнения динамического метода с другими методами обнаружения пакетов (XALT + статический метод)

		Динамический метод обнаружения пакетов	
		Пакеты обнаружены	Пакеты не обнаружены
Другие методы обнаружения пакетов	Пакеты обнаружены	1783	250
	Пакеты не обнаружены	358	903

По результатам эксперимента метод показывает высокое качество работы и уже может применяться на практике. Изначально планировалось использовать этот метод для анализа только завершившихся заданий, но после анализа его работы было принято решение отказаться от его использования на практике. Основным фактором, повлиявшим на данное решение, оказалась сложность

верификации результатов работы метода. Во многих случаях, когда динамический метод выявил наличие пакета, в то время как другие методы его не выделяли, возникает затруднение в точной верификации использования указанного пакета в задании. Также немаловажным фактором стала высокая вычислительная сложность метода — достаточно сложно было оправдать периодический вычислительно тяжелую обработку суперкомпьютерных заданий ради небольшого увеличения количества примеров использования программных пакетов.

4.2.3 Анализ результатов работы предложенных методов

Чтобы проверить эффективность обнаружения пакетов на практике, мы рассмотрим задания, которые выполнялись на суперкомпьютере “Ломоносов-2” с января по май 2021 года. В данном разделе будет производиться сравнение результатов работы только XALT с результатами работы XALT вместе со статическим методом выделения программных пакетов. Общая статистика показала, что после запуска статического метода обнаружения пакетов количество помеченных заданий (т.е. заданий с обнаруженным использованием пакетов) удвоилось.

В таблице 7 показаны результаты работы методов выделения программных пакетов как с помощью статического метода, так и XALT. Если посмотреть на весь поток заданий суперкомпьютера, то за этот период времени было запущено 124.4 тысяч заданий, и среди них было обнаружено 26.9 тысяч заданий, использующих пакеты, что составляет примерно 21.6%. Но если учесть затраченные ресурсы (CPU-часы), то этот показатель возрастает до 45.6%. Также следует отметить, что если рассматривать только XALT без статического метода, то это соотношение почти вдвое меньше — 23.2%. Это показывает эффективность статического метода, поскольку он позволяет выделять гораздо больше случаев использования пакетов.

Как было сказано ранее, XALT использует пути к исполняемым файлам для обнаружения пакетов, и у данного подхода есть недостаток: пользователи иногда переименовывают исполняемые файлы и пути к ним. Это хорошо

Таблица 7 — Количество выделенных заданий с использованием программных пакетов с помощью XALT и статического методов

Пакет	Всего	Только XALT	Только статический метод	Статический метод и XALT
LAMMPS	6066	1390	1859	2817
NWChem	2421	1257	952	121
cp2k	4860	82	4042	736
Amber	422	88	203	131
Gromacs	6072	1244	1738	3090
cabaret	318	36	241	41
molpro	973	973	0	0
firefly	1729	93	987	649

заметно в заданиях с использованием CP2K, где часто происходит указанное переименование. Статический метод не зависит от путей и поэтому может легко определить сходство в таких случаях. Отчасти из этого можно сделать вывод, что XALT чаще всего обнаруживает задания с крупными пакетами, так как их исполняемые файлы редко переименовываются, или в которых используется динамическая компоновка, а с менее распространенными пакетами, такими как CP2K и Cabaret, он справляется не так хорошо.

Также стоит отметить, что наборы заданий, обнаруженные XALT и статическим методом, могут сильно различаться. Например, в случае с NWChem только 212 заданий были классифицированы как методом XALT, так и статическим методом, что составляет всего 8.8% от общего числа обнаруженных запусков. Подобные ситуации вызваны несколькими проблемами. Первая проблема заключается в том, что часто нет возможности извлечь термины из бинарных файлов по разным причинам: модификация файла, отсутствие прав на чтение и т.д. Эти случаи составляют около 30% от всех запущенных заданий. Вторая проблема заключается в том, что бывают случаи, когда пользователи используют разные обертки для запуска программ. Это очень актуально для таких пакетов, как NAMD, где пользователи используют оболочку Charm для запуска заданий, и Quantum Espresso, где пользователи используют Python. Третьей проблемой является отсутствие возможности получить какие-либо исполняемые файлы для анализа. Это особенно заметно в пакете molpro, в зада-

ниях которых утилита nm не может извлечь термы из таблицы символов из-за иного формата исполняемого файла.

Использование предложенного метода в связке с существующей системой выделения пакетов XALT позволяет получить более точное представление о том, как используется суперкомпьютер и на какие пакеты следует обратить внимание. Некоторые результаты представлены на рисунке 4.7, где указано, сколько CPU-часов занимают задания с определенным пакетом за период с января по май 2021 года. Исключительное использование данных XALT может привести к неверным выводам, например, что NWChem использует вдвое меньше ресурсов, чем Gromacs, что фактически неверно, разница составляет всего 10%. То же самое относится и к пакетам Firefly и LAMMPS — по данным XALT Firefly потребляет на 25% больше ресурсов, чем LAMMPS, в то время как на самом деле Firefly почти в 2 раза больше ресурсов, чем LAMMPS. Стоит отметить, что даже при совместном использовании двух систем обнаружения пакетов могут быть упущены некоторые случаи использования пакетов, поэтому истинная картина может несколько отличаться.

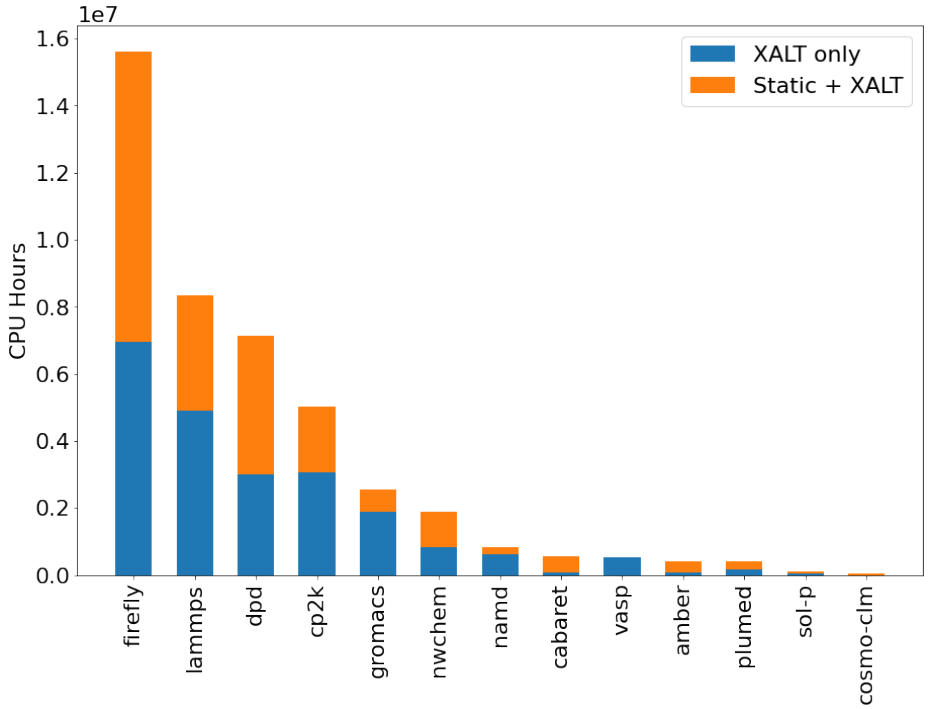

Рисунок 4.7 — Распределение затраченных CPU-часов между заданиями с использованием нескольких известных пакетов

Аналогичную статистику можно рассматривать с точки зрения определения количества пользователей, использующих пакеты. Рисунок 4.8 показывает статистику о том, сколько уникальных пользователей используют тот или иной

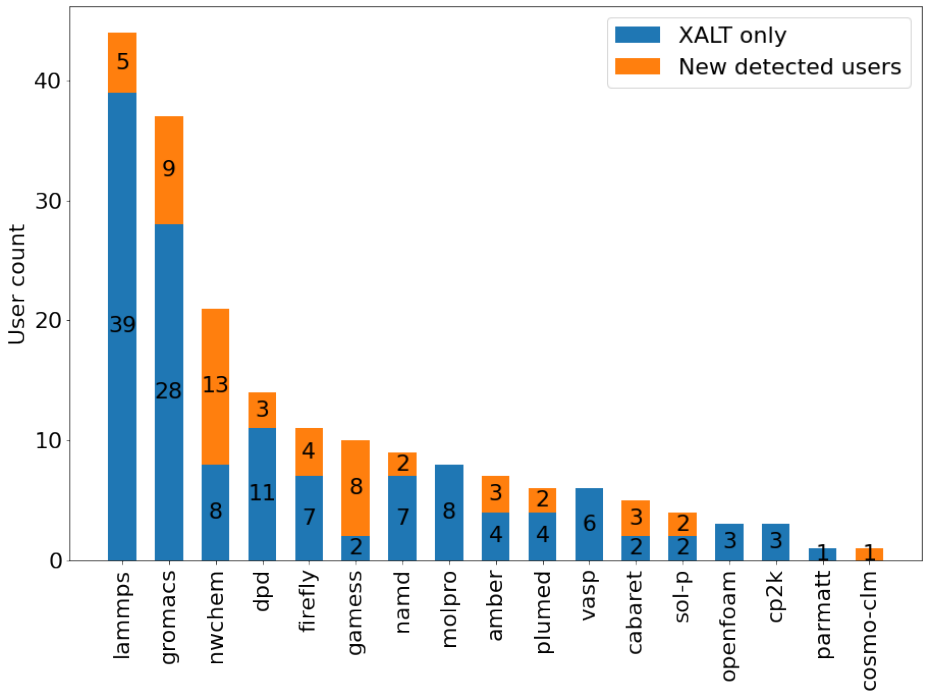

Рисунок 4.8 — Количество пользователей, использующих популярные пакеты

пакет. С момента запуска статического метода было обнаружено более 50 новых пользователей пакетов, про которых ранее не было известно, что они используют какие-либо программные пакеты. Учитывая, что в общем 285 пользователей запустили свои задания в течение периода времени с января по май 2021 года, это значительный прирост к доступной информации о потоке заданий. И, как отмечалось ранее, новая информация меняет представление о популярности конкретных пакетов (например, NWChem).

Проанализировав результаты, можно сделать вывод, что статический метод и XALT — не взаимозаменяемые системы обнаружения пакетов. Статический метод пропускает примерно столько же случаев использования пакетов, как и XALT, и наилучшее решение — это их совместное использование для обнаружения максимального количества случаев использования пакетов. Но можно быть точно уверенным в том, что при наличии входных данных, то есть информации об используемых в исполняемых файлах именах функций и переменных, статический метод будет более точным, нежели XALT. Все сводится к тому, что статический метод привязан к более достоверным данным, нежели путям и именам исполняемых файлов, как это делает XALT.

4.3 Кластеризация приложений

Кластеризация — это задача машинного обучения, которая заключается в группировке объектов по их характеристикам на подмножества, называемые кластерами. Целью кластеризации является выявление внутренней структуры данных и поиск скрытых закономерностей. Кластеризация помогает упростить задачу анализа больших объемов информации, выделить группы схожих объектов и, например, аномалии.

В нашем случае, задача кластеризации сводится к группировке суперкомпьютерных приложений. При анализе суперкомпьютерного потока заданий, можно заметить следующую тенденцию — задания зачастую имеют схожие поведение и производительность. Это чаще всего обусловлено двумя факторами: пользователи часто запускают однотипные запуски друг за другом — серии экспериментов одного приложения; несколько пользователей решают схожую прикладную задачу с использованием одних и тех же прикладных программных пакетов, таких как LAMMPS, Gromacs и т.д. И выделение таких групп схожих приложений позволит упростить их анализ, так как вместо детального анализа всех заданий будет достаточно рассмотреть только малую часть заданий из выделенного кластера, и результаты анализа будут применимы и для всех других заданий из этого кластера. Также кластеризация заданий поможет в выделении групп заданий с неэффективным исполнением в связи с тем, что поведение таких заданий будет аномально отличаться от остальных. Рассмотрим существующие работы, направленные на кластеризацию приложений.

Напомним, что для анализа суперкомпьютерных приложений доступны статические и динамические данные. Если рассматривать статический анализ, то все найденные статьи в том или ином виде описывают кластеризацию исходных кодов приложений, зачастую с целью установления факта плагиата. Для кластеризации используются статистические характеристики абстрактных синтаксических деревьев, которые получаются после анализа исходного кода приложения. Примерами таких работ являются [43] и [44].

Если рассматривать анализ динамических данных, то задача сводится к кластеризации временных рядов. Одним из основных способов кластеризации здесь является вычисление статистических свойств временных рядов, что позволяет описывать временные ряды вектором фиксированной длины,

и в дальнейшем применять доступные методы кластеризации, как, например, агломеративная кластеризация. Примером такой работы является [45], в которой авторы статьи вначале извлекали так называемые KPI (Key Performance Indicators), описывающие поведение временных рядов, а потом дополнительно высчитывали статистические характеристики временных рядов (такие как среднее/минимальное/максимальное значения, стандартное отклонение и т.д.). Полученные характеристики подавались на вход методу агломеративной кластеризации. Основная причина, по которой авторы данной работы решали проблему кластеризации приложений, очень похожа на нашу: кластеризация позволяет группировать приложения по типу решаемых задач и/или поведению, что упрощает задачу поиска неэффективных или аномальных запусков.

Любые методы кластеризации опираются на оценку близости между рассматриваемыми объектами. Наиболее используемой оценкой близости в данной области является Евклидово расстояние, которое оперирует над многомерными точками. Но при кластеризации не всегда исходные объекты являются многомерными точками, из-за чего обычно используют предварительную обработку для приведения их к сравниваемым сущностям.

Также отдельное внимание необходимо уделить выбору метрики для оценки качества результата работы метода кластеризации. В данной области существует большое количество подобных метрик, и они разделяются на две группы. Первая группа включает методы, для которых необходимо иметь “правильную” ручную разметку, то есть точное соответствие объекта и группу, к которой объект должен относиться. Примерами таких метрик являются индекс Рэнда (Rand Index, RI), индекса взаимной информации (Mutual Information, MI), MojoFM и т.д. Наиболее распространенной метрикой является индекс Рэнда из-за его простоты и интерпретируемости: значение метрики пропорционально количеству пар, которые были корректно отнесены к группам, и имеет значение от 0 до 1, где значение 0 — полное несоответствие разметок, а значение 1 — точное совпадение результата метода кластеризации с “правильной” разметкой. Но данная метрика зачастую показывает завышенные результаты для разметок, в которых количество и размеры кластеров сильно варьируются. Из-за этого на практике чаще используется модифицированная версия метрики — скорректированный индекс Рэнда (Adjusted Rand Index, ARI), в котором также учитывается количество и размер кластеров. Метрика ARI принимает

значения от -1 до 1: случайная разметка будет иметь ARI равной 0, а точное совпадение — значение 1.

Вторая группа включает в себя метрики, для которых нет необходимости иметь точную группировку объектов. Данные метрики пытаются оценить качество полученной кластеризации с помощью статистических свойств полученных кластеров. Примерами таких метрик являются коэффициент силуэта (silhouette coefficient), Calinski-Harabasz Index, Davies-Bouldin Index и т.д. Наиболее распространенным является коэффициент силуэта. Он учитывает как компактность кластеров (чем ближе объекты внутри кластера друг к другу, тем лучше), так и разделимость кластеров (чем дальше объекты одного кластера от объектов других кластеров, тем лучше). Для вычисления коэффициента силуэта для каждого объекта сначала определяется среднее расстояние до всех объектов внутри того же кластера. Затем находится среднее расстояние до объектов из ближайшего “соседнего” кластера. Коэффициент силуэта для объекта вычисляется как разность между этими двумя значениями, деленная на максимум из них. Общий коэффициент силуэта для всего набора данных вычисляется как среднее значение коэффициентов силуэта для каждого объекта.

Все вышеупомянутые методы кластеризации и оценки их качества работы будут учитываться в зависимости от доступных исходных данных и требований к методам кластеризации при их выборе в последующих разделах.

4.3.1 Описание предлагаемого метода кластеризации приложений на основе метода агломеративной кластеризации

Как говорилось ранее, для кластеризации необходимо иметь оценку расстояния между объектами. Ранее в разделах 2 и 3 были предложены 2 метода сравнения суперкомпьютерных приложений, которые и будут использоваться в процессе кластеризации.

Идея предлагаемого метода кластеризации заключается в том, чтобы использовать как информацию об исполняемых файлах, так и поведении приложения. Для этого предлагается проводить кластеризацию в два этапа:

– Этап 1 — “грубая” кластеризация на основе статической информации. Данный этап основан на допущении, что задания по поведению будут

схожи, только если их исполняемые файлы также схожи. Очевидно, что бывают исключения, которые не вписываются в данное допущение, но по наблюдениям на практике оно верно для абсолютного большинства заданий. Поэтому, на данном этапе для кластеризации заданий используется оценка расстояния между заданиями на основе статических данных. Полученные на данном этапе кластера будем называть статическими.

– Этап 2 — “тонкая” кластеризация на основе динамической информации. После получения статических кластеров, каждый из них разбивается на подкластера, которые формируются с помощью второго метода кластеризации. Этот метод кластеризации основан на оценке расстояния между приложениями на основе динамических данных.

Пример работы предложенного метода кластеризации приведен на рисунке 4.9.

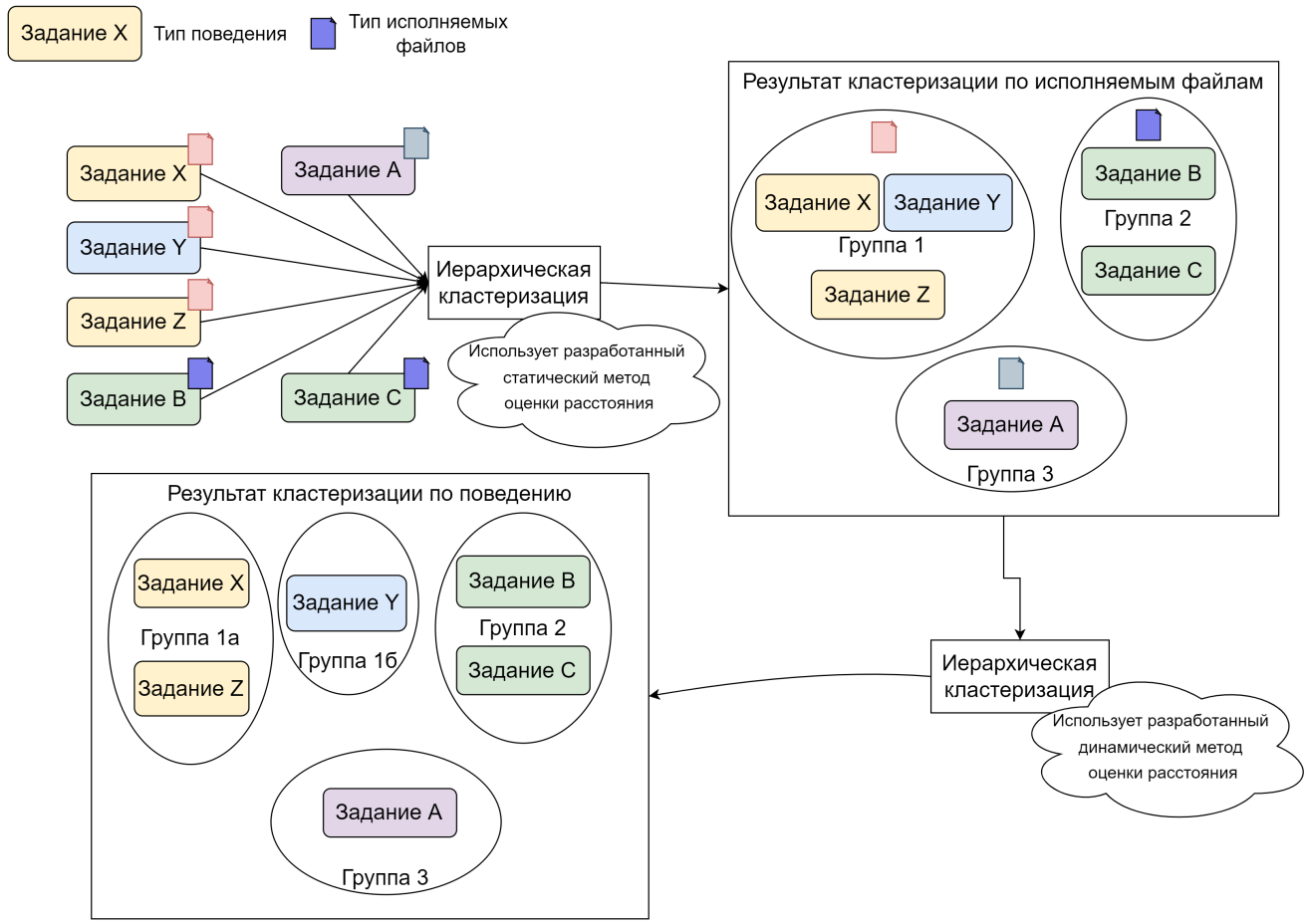

Рисунок 4.9 — Пример работы предложенного метода кластеризации на основе статических и динамических данных.

В дальнейшем данный алгоритм будем называть оффлайн методом, так как данный алгоритм подходит для анализа большого количества заданий, но не подходит для анализа потока суперкомпьютерных заданий в реальном времени, так как при появлении даже одного нового задания необходимо заново проводить кластеризацию всех заданий.

Одним из основных вопросов, которые необходимо решить перед использованием предложенного оффлайн метода, является выбор методов кластеризации на этапах 1 и 2. Для выбора необходимо учесть следующие ограничения:

– Метод кластеризации не должен быть привязан к фиксированному количеству кластеров. Данное ограничение обусловлено тем, что заранее неизвестно, какое количество групп приложений существует. Данное ограничение исключает такие популярные методы кластеризации, как K-means и спектральная кластеризация.
– Метод кластеризации должен принимать на вход любую функцию расстояния. Данное ограничение исключает такой метод кластеризации, как BIRCH, который привязан к Евклидовому расстоянию.
– Метод кластеризации для работы не должен строить вспомогательные объекты. Например, метод K-means в ходе работы использует центры кластеров, но для временных рядов и исполняемых файлов нельзя получить “среднее” значение.

Наиболее популярными группами методов кластеризации, подходящих под вышеуказанные ограничения, являются иерархическая кластеризация и кластеризация на основе плотности. Наиболее распространенным методом иерархической кластеризации является агломеративная кластеризация. Суть метода состоит в том, что все объекты изначально относятся к разным кластерам, после чего кластера итеративно объединяются в более крупные по определенному правилу. Например, после объединения среднее расстояние между объектами одного кластера не должно превышать установленный порог, или для каждого объекта кластера должен существовать другой объект в кластере, расстояние до которого меньше установленного порога.

Примером метода кластеризации на основе плотности является DBSCAN. Для работы метода необходимо указать 2 параметра — минимальное количество объектов, формирующих кластер (min_c), и пороговое значение для расстояния между объектами (d). Далее вводится понятия основного объекта p — объекта, у которого есть больше, чем min_c объектов, расстояние до которых меньше d.

Далее вводится понятие достижимости объекта: объект q достижим из основного объекта p, если существует такая последовательность объектов p, p_1, ..., p_j, q, где все объекты p_i являются основными, и расстояние между соседними элементами последовательности меньше min_c. Это позволяет определить понятие кластера: все достижимые объекты для основного объекта p образуют один кластер. Если после кластеризации есть объекты, которые не попали ни в один из кластеров, они считаются выбросами.

Было принято решение рассмотреть оба варианта кластеризации и выбрать наиболее подходящий на основе результатов тестирования.

4.3.2 Тестирование предложенного метода кластеризации

Для начала тестирования необходимо определить, на каких заданиях проводится данное тестирование. Для этого вручную были размечены ∼300 заданий для первичного тестирования и подбора параметров методов кластеризации, а также дополнительно ∼300 заданий для проверки переносимости подобранных параметров на новые задания. Второй размеченный набор заданий будем называть тестовой выборкой. Выборка для подбора параметров включает в себя задания четырех уникальных пользователей, а вторая выборка — других семи уникальных пользователей. В связи с тем, что имеется ручная разметка заданий, было принято решение использовать скорректированный индекс Рэнда (ARI) для оценки качества работы метода кластеризации.

Из-за того, что параметры статического и динамического методов выделения схожих заданий уже были подобраны при их апробации, в предложенном методе кластеризации необходимо только подобрать параметры методов кластеризации. Для агломеративной кластеризации параметрами являются алгоритм объединения кластеров (например, с помощью среднего/максимального расстояния между объектами) и установленный порог расстояния. Для DBSCAN это минимальное количество элементов в кластере и установленный порог расстояния.

Во время подбора параметров возник вопрос: возможно ли выбрать метод кластеризации и их параметры таким образом, чтобы качество кластеризации было высоким для всех пользователей? Ведь поведение приложений пользова-

телей может сильно отличаться друг от друга, из-за чего и параметры методов кластеризации могут по-разному себя проявлять во время процесса кластеризации. Для этого было предложено высчитывать скорректированную среднюю оценку ARI для всех пользователей по формуле 4.7. Данная оценка будет высчитываться для всевозможных параметров используемых методов кластеризации, по которому и будет выбираться набор параметров, наиболее подходящий для всех пользователей. Наряду со скорректированной средней оценкой также будет высчитываться максимальная оценка ARI, полученная при подборе параметров метода кластеризации, для конкретного пользователя. Это нужно для сравнения того, как сильно оценка ARI для наиболее подходящего набора параметров для всех пользователей отличается от наилучшего набора параметров для конкретного пользователя.

$$\text{Скорр-я средняя оценка} = \frac{\sum_{\text{user}} \text{ARI пользователя} * \text{кол-во заданий пользователя}}{\text{Общее количество заданий}} \tag{4.7}$$

Таблица 8 — Полученные оценки ARI для пользователей

	User 1	User 2	User 3	User 4
Наилучшая возможная оценка ARI	0.659	0.859	0.687	0.68
Оценка ARI для наиболее подходящего набора параметров	0.635	0.824	0.687	0.68

Таблица 9 — Сравнение наилучших оценок ARI для пользователей с использованием различных методов кластеризации

Статический этап	Динамический этап	User 1	User 2	User 3	User 4
Агломеративная	Агломеративная	0.659	0.859	0.687	0.68
Агломеративная	DBSCAN	0.634	0.599	0.660	0.296
DBSCAN	Агломеративная	0.659	0.859	0.687	0.68
DBSCAN	DBSCAN	0.634	0.603	0.565	0.296

Результат сравнения параметров и различных методов кластеризации показан в таблицах 8 и 9. Стоит отметить тот факт, что при применении DBSAN и агломеративной кластеризации на первом этапе кластеризации оценки ARI идентичны, что означает возможность использования любого из этих двух методов. На этапе кластеризации на основе динамических данных ситуация другая: использование DBSCAN показывает результаты значительно хуже, чем при

использовании агломеративной кластеризации. Данная картина повторяется и при выборе наилучшего набора параметров. Из-за этого было принято решение использовать агломеративную кластеризацию на обоих этапах кластеризации, тем более что агломеративная кластеризация может представлять результаты в виде дендрограммы, что позволяет визуально лучше понять структуру кластеров.

Если рассматривать непосредственно сами значения оценки ARI, то они для всех пользователей больше 0.65. Это показывает, что кластеризация далека от случайной, и соответствует хорошему качеству работы метода. Для дополнительной проверки результат кластеризации был вручную проанализирован. Анализ показал, что во всех случаях предложенный метод кластеризации разбивал кластера в исходной разметке на более маленькие, что и повлияло на итоговую оценку. Стоит отметить, что преложенный метод ни разу не относил к одному кластеру задания, которые в исходной разметке были в разных кластерах. Также можно заметить, что для агломеративной кластеризации наилучшая возможная оценка ARI для пользователя незначительно отличается от оценки ARI для наиболее подходящего набора параметров для всех пользователей, что в свою очередь означает: нет надобности в подборе параметров кластеризации для отдельного пользователя, и можно использовать один набор параметров для всех пользователей.

Для проверки данного утверждения необходимо проверить метод кластеризации на новых, ранее не рассматриваемых пользователях. Для этого подойдут задания из тестовой выборки, в которую вошли задания семи уникальных пользователей. Результат вычисления оценки ARI с помощью подобранных параметров метода кластеризации показан в таблице 10. Как можно заметить, что для 5 из 7 пользователей предложенный метод кластеризации выдал результат, очень близкое к ручной разметке; для 2 пользователей значение оценки было сильно ниже, чем у остальных. Детальный анализ заданий данных пользователей показал, что динамический метод не смог выделить различия в поведении для нескольких кластеров, из-за чего они были объединены в один, что сильно повлияло на оценку ARI.

Таблица 10 — Оценки ARI для заданий пользователей из тестовой выборки.

	User 5	User 6	User 7	User 8	User 9	User 10	User 11
Оценка ARI	0.9	0.86	0.69	0.86	0.87	0.98	0.68

При подсчете оценке ARI для всех заданий, а не только одного пользователя, получилось значение 0.75, что показывает хорошее качество работы предложенного метода кластеризации. И это также подтверждает утверждение, что задания пользователей можно кластеризовать с помощью общего набора параметров методов кластеризации.

4.3.3 Адаптация предложенного метода кластеризации для проведения регулярного анализа заданий

Как говорилось ранее, предложенный метод кластеризации заданий подходит только для редкой по времени кластеризации, так как метод не позволяет определить кластер нового задания без кластеризации всех предыдущих заданий. С учетом того, что количество запускаемых заданий на суперкомпьютерном комплексе велико, полная повторная кластеризация каждые 15-30 минут вычислительно накладна.

Для решения данной задачи необходим метод кластеризации новых заданий на основе существующей разметки (она определяется оффлайн алгоритмом из раздела 5.2). Предлагается использовать подход, идентичный использованному ранее при решении задачи выделения пакетов. В данном случае базой знаний будет выступать не соответствие задания к пакету, а соответствие задания к кластеру, то есть {задание : ID кластера, к которому принадлежит задание}. Предлагается использовать следующий алгоритм:

- При появлении нового задания, оно сравнивается со всеми заданиями из базы знаний сначала статическим методом выделения схожих заданий, а потом среди найденных — динамическим методом.
- Если в базе знаний нет схожих заданий — считается, что новое задание не относится ни к одному из существующих кластеров. В таком случае заводится новый кластер, в котором только одно задание — новое.
- Если в базе знаний есть схожие задания — выбирается наиболее близкое задание, которое определяется динамическим методом выделения схожих заданий. Новое задание считается относящимся к тому же кластеру, что и самое ближайшее к нему задание, и база знаний обновляется.

Предлагаемый метод будем в дальнейшем называть онлайн методом. Предполагается, что данный метод будет проводить кластеризацию с частой периодичностью (каждые 15-30 минут), а обновление базы знаний с помощью оффлайн метода будет проводиться редко — раз в месяц.

Для проверки работоспособности предложенного метода было проведено сравнение результатов работы онлайн и оффлайн методов, что позволило понять, как сильно онлайн метод расходится от исходного метода. Для этого были рассмотрены все задания с суперкомпьютера “Ломоносов-2” за период с января по апрель 2022 года, количество которых составило около 6 тысяч. В базу знаний вошли все задания с января по март, а задания за апрель были кластеризованы с помощью онлайн метода. После сравнения результатов работы онлайн и оффлайн методов была получена оценка Adjusted Rand Index равная 0.9948, что означает практически идентичную разметку. Это показывает, что онлайн метод можно использовать для частой периодической кластеризации заданий суперкомпьютера, в то время как оффлайн метод будет использоваться для редкого обновления базы знаний.

4.3.4 Примеры применения результатов работы метода кластеризации приложений

В данном разделе будет показано несколько вариантов использования предложенного метода кластеризации суперкомпьютерных заданий для получения новых знаний о них.

Первый вид анализа — изучение статистики по полученным кластерам. Для этого предложенных методом были кластеризованы задания, которые использовались для проверки работоспособности онлайн метода в предыдущем разделе: 6 тысяч заданий в период с января по апрель 2022 года. При их анализе сразу были замечены крупные кластеры: $\sim$25% всех заданий принадлежали 10 кластерам. Эта еще более заметно при рассмотрении кластеров в рамках используемых пакетов: 50% из $\sim$1500 заданий, использующих программный пакет LAMMPS, принадлежат 7 кластерам, 25% из $\sim$1500 заданий с Gromacs принадлежат 3 кластерам, и 60% из $\sim$350 заданий с CP2K принадлежат всего 1 кластеру! Это показывает, что кластеризация позволяет не просто выделить

крупные группы схожих заданий, а также упрощает их дальнейший анализ. Крупные кластеры также выделяются среди заданий, которые завершились некорректно (у данных заданий статус завершения “FAILED”): 4 кластера занимают 25% таких заданий.

Другой вид полезного на практике анализа — выделение аномалий в сериях запусков. Зачастую пользователи запускают большое количество однотипных экспериментов с небольшими изменениями во входных данных или параметрах запуска, и данные запуски обычно имеют схожее поведение во время их исполнения. Метод кластеризации может объединить данные задания в одну группу, и можно использовать данное свойство для выделения аномалий. В данном контексте аномалией является задание, которое существенно отличается от предыдущих и последующих заданий по своему поведению во время исполнения (контекст задания). Причинами такого существенного отличия могут быть запуск другого приложения, изменение параметров запуска, влияющее на поведение программы (например, изменение алгоритма обработки данных/расчета), специфические особенности реализации программы, при которых для определенного набора входных данных/параметров наблюдается нетипичное поведение, ошибка в реализации приложения, что приводит к некорректному поведению, или аппаратные/системные сбои в вычислительных узлах суперкомпьютера.

Существуют ситуации, когда обнаружить аномалии в работе задания невозможно без учета контекста и анализа серии запусков. Например, чрезмерная общая нагрузка на сетевую файловую систему могут снизить производительность задания, что выделяет его среди типичных запусков. Сбои на коммутаторах могут привести к задержкам в сети, что влияет на использование CPU и интенсивность сетевой нагрузки, и без учета серии запусков такие аномалии могут быть расценены как особенности программной реализации.

Таким образом, аномалиями в сериях запусков будем называть такие задания, которые в последовательности запусков заданий пользователя длиной N сильно отличаются по поведению от всех других заданий в данной последовательности, при этим все другие задания схожи между собой. Число N может варьироваться в зависимости от требуемой строгости выделения таких аномалий. Для демонстрации приведем следующий пример запусков заданий, где задания раскрашены по тому, к какому кластеру они принадлежат.

$Job1, Job2, Job3, Job4, Job5, Job6, Job7, Job8, Job9, Job10, Job11, Job12$

Видно, что большая часть последовательности запусков заданий принадлежат зеленому кластеру, но среди них появляется задание из красного кластера – значит считаем задание Job7 аномальным. Если же задания из других кластеров (синий и желтый) находятся на границе последовательности запусков кластера, то их не стоит считать аномальными (поскольку нет достаточной информации о соседних с ними запусках).

Для проверки применимости метода были проанализированы все задания суперкомпьютера “Ломоносов-2” за январь-май 2023 года. Как ранее говорилось, работа метода сильно зависит от параметра N — количества рассматриваемых соседей. Во время апробации были рассмотрены все значения N от 3 до 10, и все выделенные задания вручную проанализированы. Наилучшие результаты были получены при N равным 9, где под наилучшими понимается выделение аномалий без ошибок второго рода, то есть отсутствие выделенных аномалий, которые на самом деле таковыми не являются. Основная идея данного подхода в том, что проверить ошибки второго рода вручную можно, в то время как сложно оценить количество ошибок первого рода, так как количество запускаемых заданий очень велико.

При вышеупомянутом значении N были выделены 248 аномальных случаев. Это составляет 1.7% от общего числа заданий. В результате ручной проверки было выявлено, что

- выделенные задания действительно выбиваются по поведению в сериях запусков заданий пользователя;
- команды запуска аномальных заданий и заданий в его окрестности отличаются незначительно – значит запускались однотипные приложения с немного измененными, или даже идентичными входными параметрами;
- поведение аномальных заданий в большинстве случаев не выглядит подозрительным, то есть без учета контекста запусков данное задание нельзя выделить как аномальное.

Это показывает, что предложенный метод действительно можно использовать для выявления аномалий в сериях запусков. Стоит отметить, что далеко не всегда такие выявленные аномальные задания работают неэффективно, ведь

даже небольшое изменение параметров запуска может сильно влиять на поведение задания, и это может быть абсолютно нормальным свойством данного типа приложений. Поэтому было принято решение, что, когда данный вид анализа будет запущен для обработки потока суперкомпьютерных заданий в реальном времени, мы не будем оповещать всех пользователей о найденных аномальных случаях. Если пользователю данный вид анализа будет интересен, будет предоставлена возможность включить обнаружение таких аномалий среди их запусков и настроить оповещение.

4.4 Программная реализация предложенных методов анализа суперкомпьютерных заданий

В данном разделе будет описано то, как на данный момент устроено программное решение для реализации методов, описанных в разделах 4.1-4.3. Это решение предназначено для анализа с определенной периодичностью всего потока заданий, выполняющихся на суперкомпьютере.

4.4.1 Описание используемых источников данных

Все программные реализации, описанные в данной главе, используют разработанные методы обнаружения схожих приложений, описанные в главах 2 и 3, то есть работают со статическими и/или динамическими данными о задании. Рассмотрим каждый из этих типов данных.

Используемые статические данные состоят из данных планировщика заданий и данных XALT. Для хранения общей информации о заданиях на суперкомпьютере Ломоносов-2 администраторы используют базу MongoDB, которая включает в себя данные планировщика заданий, а также результаты дополнительной аналитики. Данные планировщика заданий (например, SLURM) предоставляют общую информацию о задании, как, например, длительность исполнения, какие узлы были выделены, логин пользователя, а также путь к исполняемому файлу или скрипту, который был запущен. Поскольку пользователи часто используют скрипты для запуска нескольких приложений, данные от планировщика заданий могут быть недостаточными для статического анализа. Поэтому также используется информация от системы XALT, которая содержит данные обо всех запущенных заданиях: полные пути ко всем исполняемым файлам, а также пути к объектным файлам, с которыми компонуется исполняемый файл. Система XALT хранит данные в базе данных MySQL.

Под дополнительной аналитикой подразумевается любой анализ запускаемых заданий. Например, в задаче выделения программных пакетов используется информация о пакетах, полученная существующей системой обнаружения пакетов: во всех путях у запущенных исполняемых и объектных файлов произ-

водится поиск ключевых слов, относящихся к определенному программному пакету. При обнаружении таких ключевых слов заданию присваивается использование соответствующего программного пакета. В задаче оценки качества использования суперкомпьютерных ресурсов, для трети заданий вычисляются данные оценки на основе реальных данных, которые потом используются в данной работе для заполнения базы знаний. Почти все вспомогательные данные хранятся в базе данных MongoDB с общей информацией о заданиях.

Динамические данные поступают только из системы мониторинга DiMMon, которая собирает информацию с вычислительных узлов суперкомпьютера и генерирует средние значения по различным счетчикам за минуту для каждого узла. Сопоставление данных мониторинга узла и запущенных заданий производится с помощью данных планировщика заданий, который предоставляет информацию о том, на каких узлах были запущены задания, а также времени начала и завершения их выполнения. Динамические данные хранятся в базе данных PostgreSQL. Список используемых динамических данных приведены в разделе 3.1.

4.4.2 Архитектура программного решения дла анализа суперкомпьютерного потока заданий

Сценарий работы программного решения для анализа потока заданий устроен следующим образом. Анализ проводится периодически, по умолчанию каждые 30 минут. Процесс начинается с отбора новых заданий, которые были запущены или завершили свое исполнение за предыдущий период, а далее начинается их анализ. При рассмотрении различных методов анализа суперкомпьютерного потока заданий, описанных ранее в данной главе, становится понятно, что первые шаги алгоритма зачастую схожи: выбор заданий для анализа, извлечение данных и их предобработка и т.д., зачастую не зависят от того, какой метод анализа используется. Из-за этого вводится понятие очереди заданий. В данном случае очередью заданий будем называть упорядоченный набор заданий, для которых необходимо провести анализ с помощью нескольких методов, имеющих общие шаги и входные данные. На практике, это выливается в образование двух очередей: для обработки статическими и динамическими

методами. В частности, к статической очереди привязана реализация статического метода выделения пакетов, а к динамической очереди — реализации методов оценки качества использования суперкомпьютерных ресурсов, динамического метода выделения пакетов и кластеризации заданий. Стоит отметить, что динамические методы могут использовать результаты статического анализа, и из-за того, что динамические методы анализируют только завершившиеся задания с длительностью больше 30 минут, результаты работы статического анализа к этому моменту уже доступны.

Для выполнения анализа применяется слоевая архитектура, в которой каждый слой выполняет определенную стадию обработки:

– инициализация;
– извлечение данных и предобработка;
– анализ;
– завершение анализа.

Каждый слой привязан к определенной очереди заданий.

Первый слой — инициализация необходимых для анализа моделей и структур. Например, для статического анализа необходимо загрузить веса модели Doc2Vec, для динамического анализа — параметры нормализации и преобразования динамических характеристик. Для обнаружения пакетов или предсказания качества использования суперкомпьютерных ресурсов необходимо загрузить базу знаний. Поскольку все слои связаны с определенной очередью заданий, слой инициализации включает инициализацию всех необходимых моделей и структур для всех типов анализа, которые будут применяться к заданиям. Программно, все вспомогательные модели и структуры хранятся в глобальном словаре, который заполняется текущим слоем и используется в последующих слоях.

Все последующие слои работают непосредственно с заданиями из очереди. Слой извлечения данных и предобработки отвечает за извлечение данных о задании с применением необходимых для анализа преобразований. Для статического метода извлекаются имена функций и переменных из исполняемого файла, и происходит их преобразование в вектор фиксированной длины с помощью модели Doc2Vec. Для динамического метода извлекаются данные системы мониторинга для задания, происходит их нормализация и преобразование. Слой предобработки извлекает и обрабатывает данные сразу для всех последующих шагов. Программно, данные о задании представляют собой объект с полями,

заполняемыми слоем извлечения данных и предобработки, и этот объект используется последующими слоями вместе с глобальным словарем для анализа.

Следующий слой отвечает непосредственно за проведение анализа поступающих заданий. После получения информации о задании из предыдущего слоя, проводится анализ с применением методов обнаружения схожих приложений. Для статического метода выделения пакетов, вектор фиксированной длины, представляющий анализируемое задание, сравнивается с векторами всех заданий из базы знаний. На основе результатов сравнения определяется использование того или иного пакета для нового задания. Для оценки метрики качества использования суперкомпьютерных ресурсов анализируемое задание сравнивается со всеми заданиями из базы знаний с помощью статического и динамического методов. На основе найденных схожих заданий производится оценка качества.

Результаты анализа передаются в слой завершения, где они сохраняются и могут использоваться для корректировки базы знаний. Например, при оценке качества использования суперкомпьютерных ресурсов новое задание может быть добавлено в базу знаний, если для него доступны истинные значения метрик, при этом заменяя самое старое задание.

В листинге 4.1 изображен развернутый псевдокод, описывающий работу программного решения на примере задачи выделения пакетов с использованием динамических методов обнаружения схожих приложений и предсказания оценок качества использования вычислительных ресурсов. Данный псевдокод иллюстрирует все шаги алгоритмов и их соответствие различным слоям, описанным ранее, для динамической очереди заданий.

Листинг 4.1 Псевдокод программного решения на примере задач выделения пакетов и предсказания качества использования суперкомпьютерных вычислительных ресурсов

```
# Объявление глобального объекта класса Job, который хранит все данные об ана
    лизируемом задании
Job job
# Объявление глобального объекта класса Env, который хранит все вспомогательн
    ые данные для анализа
Env env

Функция main(Q): // Q - очередь заданий на обработку
```

```
→# Слой инициализации
→# Вызов функции слоя инициализации с общими шагами
→common_init()
→# Вызов функции слоя инициализации для задачи оценки качества использования ресурсов суперкомпьютера
→metric_init()
→# Вызов функции слоя инициализации для задачи выделения пакетов
→package_init()

→Для каждого id задания job_id в очереди Q:
→→Очищение полей объекта job
→→Запись job_id в объект job
→→# Слой извлечения данных и предобработки
→→# Вызов функции слоя извлечения данных и предобработки с общими шагами
→→common_extract()
→→# Вызов функции слоя извлечения данных и предобработки для задачи оценки качества использования ресурсов суперкомпьютер
→→metric_extract()
→→# Слои анализа и завершения
→→# Вызов функции слоя анализа для задачи оценки качества использования ресурсов суперкомпьютера
→→metric_analysis()
→→# Вызов Функции слоя завершения для задачи оценки качества использования ресурсов суперкомпьютера
→→metric_finish()
→→# Вызов функции слоя анализа для задачи выделения пакетов
→→package_analysis()
→→# Вызов Функции слоя завершения для задачи выделения пакетов
→→package_finish()

def common_init():
→Подключение к локальной БД
→Подключение к БД Mongo, где хранятся все базовые данные о задании (данные планировщика, данные об истинных значениях метрик качества использования ресурсов и т.д.)
→Загрузка параметров методов предобработки динамических данных из локального хранилища

def package_init():
→Загрузка базы знаний из локального хранилища (knowledge_base_package)
→Загрузка динамических данных о заданиях из базы знаний из локальной БД (dynamic_data_package)
→Запись knowledge_base_package и dynamic_data_package в объект env
```

```
def metric_init():
→Загрузка базы знаний из локального хранилища (knowledge_base_metric)
→Загрузка динамических данных о заданиях из базы знаний из локальной БД (
   dynamic_data_metric)
→Загрузка статических данных о заданиях из базы знаний из локальной БД (
   static_data_metric)
→Запись knowledge_base_package, static_data_metric и dynamic_data_package в
   объект env

def common_extract():
→Загрузка базовых данных о задании job из БД Mongo
→Загрузка динамических данных о задании job из локальной БД (dynamic_data)
→Предобработка динамических данных
→Запись dynamic_data в объект job

def metric_extract():
→Загрузка статических данных о задании job из локальной БД (static_data)
→Запись static_data в объект job

def package_analysis():
→Инициализируем массив, в котором будут храниться названия обнаруженных прог
   раммных пакетов (packages)
→Для задания job_i из базы знаний:
→→Сравнение job_i с job с помощью динамического метода на основе DTW
    Если оценка расстояния между заданиями ниже порогового значения, добавляе
   м все пакеты задания job_i в массив packages
→Запись массива packages в объект job (job.packages)

def metric_analysis():
→Инициализируем массив найденных близких заданий по статическому признаку (
   close_static)
→Инициализируем массив найденных близких заданий по динамическому признаку (
   close_dynamic)
→Для задания job_i из базы знаний:
→→Сравнение job_i с job с помощью статического метода на основе модели
   Doc2Vec
→→Если статическая оценка расстояния между заданиями ниже порогового значен
   ия, добавляем job_i в close_static

→Если в close_static есть задания:
→→Для задания job_j из close_static:
→→→Сравнение job_j с job с помощью динамического метода на основе DTW
```

```
→→→Если динамическая оценка расстояния между заданиями ниже грубого порого
   вого значения, добавляем job_i в close_dynamic

→# Если нет близких статических заданий или нет близких динамических заданий
    среди близких статических заданий
→Если в close_dynamic нет заданий:
→→Для задания job_i из базы знаний:
→→→Сравнение job_i с job с помощью динамического метода на основе DTW
→→→Если статическая оценка расстояния между заданиями ниже строгого порого
   вого значения, добавляем job_i в close_dynamic

→Если в close_dynamic есть задания:
→→Агрегация всех оценок качества использования суперкомпьютерных ресурсов и
   з заданий в close_dynamic для предсказания оценки для job
→→Запись агрегированной оценки в объект job (job.metrics)

def package_finish():
→Запись job.packages в локальную БД для сохранения результатов

def metric_finish():
→Запись job.metrics в локальную БД для сохранения результатов
→Если существуют истинные значения оценок для задания job:
→→Удаление самого старого задания из базы знаний
→→Добавление задания job в базу знаний
→→Сохранение базы знаний в локальной БД
```

Заключение

В данной работе представлены два разных метода поиска схожих приложений, работающих на суперкомпьютере. Первый метод основан на анализе статической информации о приложениях: именах используемых функций и переменных, извлеченных из исполняемых и объектных файлов. Метод показывает хорошие результаты в выявлении схожих приложений (метрика ARI на тестовой выборке равна 0.79). Второй метод основан на анализе динамических характеристик, предоставляемых системой мониторинга и описывающих производительность приложений во время выполнения. Метод основывается на алгоритме Dynamic Time Warping. Результаты экспериментов показывают хорошие результаты в обнаружении похожих по поведению приложений: точность 0.85 на тестовой выборке, где точность — отношение количества пар заданий с корректно определенной схожестью к общему количеству пар заданий в выборке.

На основе предложенных методов оценки схожести заданий были предложены варианты решения важных практических задач, которые встают перед администраторами суперкомпьютерных центров. Одной из таких задач является решение проблемы обнаружения пакетов программ (таких как GROMACS, NAMD и др.), используемых в заданиях, которые запускаются на суперкомпьютере. Эта задача очень важна, так как информацию об используемых пакетах можно применять не только для получения новой статистики о функционировании суперкомпьютера или предоставления администраторам новой информации о потребностях пользователей, но и в качестве составляющей рекомендательной системы. Например, если известны свойства пакета (масштабируемость, интенсивность использования процессоров, графических ускорителей или памяти), можно подсказать пользователям, как использовать пакет более эффективно.

Первый предложенный подход, основанный на статическом анализе, показал точность 0.9 на тестовой выборке. Во время тестирования было обнаружено более 40% новых заданий, в которых использовались различные пакеты и что не было обнаружено ранее другими методами. Также было показано, что данный подход может не только обнаруживать известные пакеты, но и помогать выявлять новые, неизвестные пакеты (например, в нашем случае был обнару-

жен пакет XAMG для решения систем линейных алгебраических уравнений). Статический метод уже используется на суперкомпьютере “Ломоносов-2” на постоянной основе.

Второй подход, основанный на динамическом анализе, также показал высокую точность на тестовой выборке — 0.85. В ходе его тестирования мы обнаружили, что при рассмотрении заданий с уникальным поведением этот метод выявлял до 20% новых заданий с пакетами, но из-за сложности верификации результатов было принято решение отказаться от применения динамического метода для выделения программных пакетов.

Предложенные в работе методы выделения схожих приложений также были применены для решения задачи предсказания оценок качества использования суперкомпьютерных ресурсов. Данные оценки качества были разработаны ранее администраторами суперкомпьютера “Ломоносов-2”, и их расчет уже запущен в реальном времени. Но данные оценки рассчитываются только для трети запускаемых заданий, из-за чего есть потребность в предсказании этих оценок для оставшихся заданий. Предложенный подход на основе выделения схожих приложений показал высокую эффективность: с помощью него можно предсказать оценки заданий для 87% заданий со средним отклонением в 2.5%, что считается допустимым качеством работы для данной прикладной задачи.

Также была показана эффективность применения методов выделения схожих приложений для решения задачи кластеризации заданий. Кластеризацию предлагается проводить в два этапа. На первом этапе выделяются кластера на основе статических данных, что позволяет сгруппировать задания со схожими исполняемыми файлами. На втором этапе каждый такой кластер разбивается на более маленькие кластера на основе динамических данных. Это позволяет выделять кластера заданий, схожих как по поведению во время их исполнения, так и по исполняемым файлам. На обоих этапах применяется метод агломеративной кластеризации. На тестовой выборке, предложенный метод получил оценку ARI равную 0.75, что говорит о хорошем качестве работы метода. Данный метод кластеризации может быть использован для решения множества практических задач. В качестве примера были приведены задача выделения крупных кластеров заданий для их дальнейшего анализа, а также задача выделения аномальных заданий в сериях запусков отдельных пользователей.

Публикации автора по теме диссертации

Научные статьи, опубликованные в рецензируемых научных изданиях, рекомендованных для защиты в диссертационном совете МГУ по специальности и отрасли наук:

1. Shaikhislamov, D. Solving the problem of detecting similar supercomputer applications using machine learning methods / D. Shaikhislamov, V. Voevodin // Communications in Computer and Information Science. — 2020. — vol. 1263. — pp. 46-57. — EDN OPGJNI. — (Scopus Q4, Импакт-фактор 0.182 (SJR))[1.125/1 п.л.]
2. Shaikhislamov, D. Development and Practical Application of Methods for Detecting Similar Supercomputer Jobs / D. Shaikhislamov, V. Voevodin // Communications in Computer and Information Science. — 2021. — vol. 1437. — pp. 18-30. — EDN HZXGMO. — (Scopus Q4, Импакт-фактор 0.182 (SJR))[1.1875/1.0675 п.л.]
3. Shaikhislamov, D. Analysis of Software Package Usage Based on Methods for Identifying Similar HPC Applications / D. Shaikhislamov, V. Voevodin // Communications in Computer and Information Science. — 2021. — vol. 1510. — pp. 310-321. — EDN DGFJFF. — (Scopus Q4, Импакт-фактор 0.182 (SJR))[1.125/1 п.л.]
4. Voevodin, V. V. How to Assess the Quality of Supercomputer Resource Usage / V. V. Voevodin, D. I. Shaikhislamov, D. A. Nikitenko // Supercomputing Frontiers and Innovations. — 2022. — vol. 9, No. 3. — pp. 4-18. — EDN VKETHG. — (Scopus Q4, Импакт-фактор 0.138 (SJR))[2.0625/0.55 п.л.]

Список литературы

1. A systematic literature review on source code similarity measurement and clone detection: Techniques, applications, and challenges [Текст] / M. Zakeri-Nasrabadi [и др.] // Journal of Systems and Software. — 2023. — Т. 204. — С. 111796. — URL: https://www.sciencedirect.com/science/article/pii/S0164121223001917.
2. *Burrows, S.* Efficient plagiarism detection for large code repositories [Текст] / S. Burrows, S. M. M. Tahaghoghi, J. Zobel // Softw. Pract. Exper. — USA, 2007. — Февр. — Т. 37, № 2. — С. 151—175.
3. *Roy, C. K.* NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization [Текст] / C. K. Roy, J. R. Cordy // 2008 16th IEEE International Conference on Program Comprehension. — 2008. — С. 172—181.
4. DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones [Текст] / L. Jiang [и др.] // 29th International Conference on Software Engineering (ICSE'07). — 2007. — С. 96—105.
5. *Tekchandani, R.* Semantic code clone detection using parse trees and grammar recovery [Текст] / R. Tekchandani, R. K. Bhatia, M. Singh // Confluence 2013: The Next Generation Information Technology Summit (4th International Conference). — 2013. — С. 41—46.
6. Using a Nearest-Neighbour, BERT-Based Approach for Scalable Clone Detection [Текст] / M. Chochlov [и др.] // 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME). — 2022. — С. 582—591.
7. Deep learning similarities from different representations of source code [Текст] / M. Tufano [и др.] // Proceedings of the 15th International Conference on Mining Software Repositories. — Gothenburg, Sweden : Association for Computing Machinery, 2018. — С. 542—553. — (MSR '18). — URL: https://doi.org/10.1145/3196398.3196431.
8. [Research Paper] On the Use of Machine Learning Techniques Towards the Design of Cloud Based Automatic Code Clone Validation Tools [Текст] /

G. Mostaeen [и др.] // 2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM). — 2018. — С. 155—164.

9. Cross-Architecture Bug Search in Binary Executables [Текст] / J. Pewny [и др.] // 2015 IEEE Symposium on Security and Privacy. — 2015. — С. 709—724.

10. *David, Y.* Statistical similarity of binaries [Текст] / Y. David, N. Partush, E. Yahav // Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation. — Santa Barbara, CA, USA : Association for Computing Machinery, 2016. — С. 266—280. — (PLDI '16). — URL: https://doi.org/10.1145/2908080.2908126.

11. Scalable Graph-based Bug Search for Firmware Images [Текст] / Q. Feng [и др.] // Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. — Vienna, Austria : Association for Computing Machinery, 2016. — С. 480—491. — (CCS '16). — URL: https://doi.org/10.1145/2976749.2978370.

12. *David, Y.* Similarity of binaries through re-optimization [Текст] / Y. David, N. Partush, E. Yahav // SIGPLAN Not. — New York, NY, USA, 2017. — Июнь. — Т. 52, № 6. — С. 79—94. — URL: https://doi.org/10.1145/3140587.3062387.

13. Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection [Текст] / X. Xu [и др.] // Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. — Dallas, Texas, USA : Association for Computing Machinery, 2017. — С. 363—376. — (CCS '17). — URL: https://doi.org/10.1145/3133956.3134018.

14. *Yamamoto, K.* Classifying Jobs and Predicting Applications in HPC Systems [Текст] / K. Yamamoto, Y. Tsujita, A. Uno // High Performance Computing / под ред. R. Yokota [и др.]. — Cham : Springer International Publishing, 2018. — С. 81—99.

15. *Gonzalez, J.* Automatic detection of parallel applications computation phases [Текст] / J. Gonzalez, J. Gimenez, J. Labarta // 2009 IEEE International Symposium on Parallel & Distributed Processing. — 2009. — С. 1—11.

16. On the usefulness of object tracking techniques in performance analysis [Текст] / G. Llort [и др.] // SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. — 11.2013. — С. 1—11.

17. *Berndt, D. J.* Using Dynamic Time Warping to Find Patterns in Time Series [Текст] / D. J. Berndt, J. Clifford // Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining. — Seattle, WA : AAAI Press, 1994. — С. 359—370. — (AAAIWS'94).

18. *Salvador, S.* Toward Accurate Dynamic Time Warping in Linear Time and Space [Текст] / S. Salvador, P. Chan // Intell. Data Anal. — Amsterdam, The Netherlands, The Netherlands, 2007. — Окт. — Т. 11, № 5. — С. 561—580.

19. Speeding up similarity search under dynamic time warping by pruning unpromising alignments [Текст] / D. Silva [и др.] // Data Mining and Knowledge Discovery. — 2018. — Март. — Т. 32.

20. *Al-Naymat, G.* SparseDTW: A Novel Approach to Speed up Dynamic Time Warping [Текст] / G. Al-Naymat, S. Chawla, J. Taheri // Conferences in Research and Practice in Information Technology Series. — 2012. — Янв. — Т. 101.

21. *Li, Z.-x.* Similarity Measure for Multivariate Time Series Based on Dynamic Time Warping [Текст] / Z.-x. Li, K.-w. Li, H.-s. Wu // Proceedings of the 2016 International Conference on Intelligent Information Processing. — Wuhan, China : ACM, 2016. — 15:1—15:5. — (ICIIP '16). — URL: http://doi.acm.org/10.1145/3028842.3028857.

22. *Keogh, E.* Derivative Dynamic Time Warping [Текст] / E. Keogh, M. Pazzani // First SIAM International Conference on Data Mining. — 2002. — Янв. — Т. 1.

23. A shape-based similarity measure for time series data with ensemble learning [Текст] / T. Nakamura [и др.] // Pattern Analysis and Applications. — 2013. — Нояб. — Т. 16, № 4. — С. 535—548.

24. *Olszewski, R. T.* Generalized feature extraction for structural pattern recognition in time-series data [Текст] : дис. ... канд. / Olszewski Robert T. — Pittsburgh, PA : Carnegie Mellon University, 2001.

25. *Pei, W.* Modeling Time Series Similarity with Siamese Recurrent Networks [Текст] / W. Pei, D. M. J. Tax, L. van der Maaten // CoRR. — 2016. — Т. abs/1603.04713. — arXiv: 1603.04713.

26. *Grabocka, J.* NeuralWarp: Time-Series Similarity with Warping Networks [Текст] / J. Grabocka, L. Schmidt-Thieme // CoRR. — 2018. — Т. abs/1812.08306. — arXiv: 1812.08306.

27. Multivariate Time-series Similarity Assessment via Unsupervised Representation Learning and Stratified Locality Sensitive Hashing: Application to Early Acute Hypotensive Episode Detection [Текст] / J. Dhamala [и др.] // CoRR. — 2018. — Т. abs/1811.06106. — arXiv: 1811.06106.

28. *Le, Q. V.* Distributed Representations of Sentences and Documents [Текст] / Q. V. Le, T. Mikolov // CoRR. — 2014. — Т. abs/1405.4053. — arXiv: 1405.4053.

29. Efficient Estimation of Word Representations in Vector Space [Текст] / T. Mikolov [и др.]. — 2013. — arXiv: 1301.3781 [cs.CL]. — URL: https://arxiv.org/abs/1301.3781.

30. *Řehůřek, R.* Software Framework for Topic Modelling with Large Corpora [Текст] / R. Řehůřek, P. Sojka // Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks. — Valletta, Malta : ELRA, 05.2010. — С. 45—50.

31. Scikit-learn: Machine Learning in Python [Текст] / F. Pedregosa [и др.] // Journal of Machine Learning Research. — 2011. — Т. 12. — С. 2825—2830.

32. *Zhihua Wen.* An effectiveness measure for software clustering algorithms [Текст] / Zhihua Wen, V. Tzerpos // Proceedings. 12th IEEE International Workshop on Program Comprehension, 2004. — 06.2004. — С. 194—203.

33. *Jeong, Y.-S.* Weighted dynamic time warping for time series classification [Текст] / Y.-S. Jeong, M. K. Jeong, O. A. Omitaomu // Pattern Recognition. — 2011. — Т. 44, № 9. — С. 2231—2240. — URL: https://www.sciencedirect.com/science/article/pii/S003132031000484X ; Computer Analysis of Images and Patterns.

34. *Voevodin, V.* Universal Assessment System for Analyzing the Quality of Supercomputer Resources Usage [Текст] / V. Voevodin, S. Zhumatiy // Supercomputing / под ред. V. Voevodin, S. Sobolev. — Cham : Springer International Publishing, 2021. — С. 427—442.

35. *Voevodin, V.* Overhead Analysis for Performance Monitoring Counters Multiplexing [Текст] / V. Voevodin, K. Stefanov, S. Zhumatiy // Supercomputing / под ред. V. Voevodin [и др.]. — Cham : Springer International Publishing, 2022. — С. 461—474.

36. LAMMPS - a flexible simulation tool for particle-based materials modeling at the atomic, meso, and continuum scales [Текст] / A. P. Thompson [и др.] // Comp. Phys. Comm. — 2022. — Т. 271. — С. 108171.

37. Scalable molecular dynamics on CPU and GPU architectures with NAMD [Текст] / J. C. Phillips [и др.] // The Journal of Chemical Physics. — 2020. — Июль. — Т. 153, № 4. — С. 044130. — eprint: https://pubs.aip.org/aip/jcp/article-pdf/doi/10.1063/5.0014475/16709546/044130_1_online.pdf. — URL: https://doi.org/10.1063/5.0014475.

38. *Granovsky, A. A.* Firefly version 8 [Текст] / A. A. Granovsky. — http://classic.chem.msu.su/gran/firefly/index.html.

39. User Environment Tracking and Problem Detection with XALT [Текст] / K. Agrawal [и др.] //. — 11.2014. — С. 32—40.

40. HPC Software Tracking Strategies for a Diverse Workload [Текст] / H. Na [и др.] // 2020 IEEE/ACM International Workshop on HPC User Support Tools (HUST) and Workshop on Programming and Performance Visualization Tools (ProTools). — 2020. — С. 1—9.

41. *Baer, T.* Pbsacct: A Workload Analysis System for PBS-Based HPC Systems [Текст] / T. Baer, D. Johnson // Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment. — Atlanta, GA, USA : Association for Computing Machinery, 2014. — (XSEDE '14). — URL: https://doi.org/10.1145/2616498.2616539.

42. *Maaten, L. van der.* Viualizing data using t-SNE [Текст] / L. van der Maaten, G. Hinton // Journal of Machine Learning Research. — 2008. — Нояб. — Т. 9. — С. 2579—2605.

43. *Ďuračík, M.* Scalable Source Code Plagiarism Detection Using Source Code Vectors Clustering [Текст] / M. Ďuračík, E. Kršák, P. Hrkút // 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS). — 2018. — С. 499—502.

44. *Kuhn, A.* Semantic clustering: Identifying topics in source code [Текст] / A. Kuhn, S. Ducasse, T. Gîrba // Information and Software Technology. — 2007. — Т. 49, № 3. — С. 230—243. — URL: https://www.sciencedirect.com/science/article/pii/S0950584906001820 ; 12th Working Conference on Reverse Engineering.

45. *Halawa, M. S.* Unsupervised KPIs-Based Clustering of Jobs in HPC Data Centers [Текст] / M. S. Halawa, R. P. Díaz Redondo, A. Fernández Vilas // Sensors. — 2020. — Июль. — Т. 20, № 15. — С. 4111. — URL: http://dx.doi.org/10.3390/s20154111.