Московский государственный университет имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра системного программирования

Михальцов Данила Алексеевич

Определение и локализация регрессий производительности во время исследовательского тестирования мобильных приложений

Выпускная квалификационная работа

Научный руководитель:
к.ф.-м.н.
Турдаков Денис Юрьевич
Научный консультант:
Сорокин Константин Сергеевич

Москва, 2023

Аннотация

Определение и локализация регрессий производительности во время исследовательского тестирования мобильных приложений

Михальцов Данила Алексеевич

В настоящее время очень развит рынок мобильных приложений. Современные мобильные приложения представляют собой сложные программные продукты, поэтому в них есть ошибки, в том числе нефункциональные. Проблемы с производительностью в приложениях негативно влияют на пользовательский опыт, поэтому разработчики заинтересованы в автоматизированном поиске таких ошибок. В данной работе описан процесс построения автоматизированной системы регрессионного тестирования для нахождения проблем с производительностью в мобильных приложениях на ОС Android. Поиск таких проблем производится с помощью обнаружения и локализации аномалий в данных трассировки, собранных во время автоматизированного исследовательского тестирования, с использованием исторических данных. Предложены подходы для классификации и локализации аномалий в данных трассировки. Проведена оценка качества обоих подходов, для классификации получено значение метрики F1-score $= 0,701$. Разработанная система может быть успешна применена для поиска потенциальных проблем с производительностью в мобильных приложениях.

Detection and localization of performance regressions during exploratory testing of mobile applications

Содержание

1 Введение

В последнее десятилетие наблюдается значительный рост рынка мобильных приложений. Сектор мобильных приложений является ключевой отраслью для многих компаний. Операционная система Android от Google является одной из самых популярных мобильных операционных систем, она занимает более 70% этого рынка. Взаимодействие пользователя со смартфоном происходит через приложения. В настоящий момент в магазине приложений Google Play доступно более 2,9 миллиардов приложений. Современные мобильные приложения решают самые разные задачи. Они, как и мобильные операционные системы, являются сложными программными продуктами, требующими значительных ресурсов для разработки, поддержки и контроля качества. Однако в силу многих причин них всё равно присутствуют ошибки.

Ошибки в приложениях сильно влияют на пользовательский опыт, поэтому компании заинтересованы в том, чтобы находить и исправлять как можно больше ошибок. Для поиска ошибок применяются различные подходы, в том числе автоматизированное тестирование. Тестирование часто разделяют на функциональное и нефункциональное. Первый вид направлен на проверку того, что приложение работает корректно: выполняются определённые функциональные требования. Второй вид позволяет проверить, насколько приложение удволетворяет нефункциональным требованиям. Такие требования определяют качественные характеристики приложения, например, связанные с его производительностью.

Проблемы с производительностью в приложениях проявляются, например, следующим образом: долгие загрузки, медленная скорость отклика или увеличенное энергопотребление. Приложения, при прочих равных обладающие большей стабильностью, более конкурентоспособны, поэтому для разработчиков приложений крайне актуальным является исправление ошибок, которые приводят к таким проблемам. Согласно выводам работы [1], у ошибок, вляющих на производительность, есть особенности. Во-первых, разработчикам требуется больше усилий на исправление таких ошибок. По сравнению с исправлением функциональных ошибок, требуется большее количество времени на исправление и большее количество вовлечённых разработчиков. Во-вторых, в отличие от поиска функциональных проблем, которые в большинстве случаев находятся с помощью непосредственного обнаружения разработчиками или пользователями приложения, большое количество проблем с производительностью в приложении обна-

руживается во время размышлений разработчиков о том, как работает их код. В силу обозначенных особенностей для выявления проблем с производительностью в приложении особенно актуально автоматизированное тестирование.

В процессе разработки мобильного приложения некоторые изменения могут приводить к появлению *регрессий производительности* — ухудшению производительности приложения по сравнению с прошлыми версиями. Для обнаружения регрессий применяется метод *регрессионного тестирования*: приложение регулярно проходит набор определённых тестов, в том числе это могут быть сеансы исследовательского тестирования.

Один из видов автоматизированного тестирования приложений, популярный в последнее время — *исследовательское тестирование* (exploratory testing, GUI testing) [2, 3, 4, 5]. Это тестирование производится с помощью имитации тестирующей системой взаимодействия пользователя с графическим интерфейсом приложения. Исследовательское тестирование обычно используется для поиска функциональных ошибок, например, для поиска сценариев исследования приложения, при которых оно некорректно завершается. Поскольку тестирование заканчивается по истечении заранее заданного времени, для сеанса тестирования является важной характеристикой количество исследованных состояний приложения.

Во время исследовательского тестирования мобильного приложения можно собирать различные *данные трассировки* — показатели загруженности устройства, например, активность использования ресурсов процессора, количество занимаемой приложением оперативной памяти и т.д. Аномалии в этих данных могут свидетельствовать о наличии проблем с производительностью в приложении.

Поскольку существующие инструменты исследовательского тестирования не поддерживают поиска проблем с производительностью, а он востребован среди разработчиков мобильных приложений, в данной работе предлагается разработать систему регрессионного тестирования мобильных приложений для автоматизированного поиска проблем с производительностью при помощи обнаружения и локализации аномалий в данных трассировки, собранных во время исследовательского тестирования.

Работа построена следующим образом. В главе 2 формулируются цель и постановка задачи. В главе 3 описываются избранные работы, посвящённые исследовательскому и динамическому тестированию приложений с целью выявляния нефункциональных

ошибок, а также представлены работы с описанием некоторых подходов к решению задачи поиска аномалий в многомерных временных рядах, в том числе в данных трассировки. В главе 4 описан процесс исследования и построения решения задачи. В главе 5 представлены детали реализации системы регрессионного тестирования и результаты оценки качества работы метода локализации аномалий. В главе 6 представлено заключение.

2 Постановка задачи

Целью данной работы является создание системы для определения и локализации регрессий производительности во время исследовательского тестирования мобильных приложений. Для её достижения требуется решить ряд задач:

- разработать модуль для инструмента исследовательского ИСП РАН для сбора данных трассировки (таких как использование ресурсов процессора, потребление памяти и т.д.);
- разработать и реализовать метод обнаружения аномалий в данных трассировки на базе построения модели ожидаемого поведения приложения;
- разработать и реализовать метод локализации аномальных отрезков в данных трассировки на основе анализа временных рядов с помощью матричных профилей;
- подготовить наборы данных для оценки качества предложенных методов и оценить качество работы методов обнаружения и локализации аномалий (F1-score и значения TP/FP/FN соответственно);
- реализовать систему регрессионного тестирования, объединив реализованные методы с инструментом исследовательского тестирования ИСП РАН.

3 Обзор существующих решений

В данной части приведен обзор наиболее близких к тематике работы существующих решений. Описываются избранные работы, посвящённые динамическому тестированию приложений с целью выявляния нефункциональных ошибок, а также представлены некоторые современные подходы к решению задачи поиска аномалий во временных рядах, в том числе в данных трассировки.

Поиск аномалий в многомерных временных рядах — популярное направление для исследований. Для решения этой задачи было предложено множество методов. Здесь рассмотрены современные методы, некоторые из них применялись к показателем нагрузки компьютерных систем.

В работе [6] ставится задача раннего обнаружения аномалий в многомерных временных рядах с целью предотвращения катастроф на производстве с использованием исторических данных. В работе предлагается подход на основе свёрточных нейронных сетей. Отличием от постановки задачи данной работы является упор на работу системы обнаружения аномалий в реальном времени.

В работе [7] предложены статистические подходы для обнаружения аномалий в данных о состоянии датацентров в реальном времени. Особенностью метода является непрерывная работа в реальном времени и легковесность подхода: исторические данные используются только для нахождения распределений конкреный характеристик до внедрения системы поиска аномалий.

В работе [8] производится поиск аномалий в многомерных временных рядах. Для разных промежутков временных рядов строятся сигнатурные матрицы, которые в дальнейшем используются свёрточной нейронной сетью для нахождения аномальных участков. Однако, подход, предложенный в этой статье, используется для нахождения аномалий в пределах одного многомерного ряда, а в постановке данной работы используются исторические данные, поэтому этот метод неприменим для решения задач этой работы.

В работе [9] предложен классификатор, основанный на машинном обучении, который на основе данных трассировки определяет, является ли приложение вредоносным. Отличия от подхода данной работы: классификатор обучается на размеченном наборе данных из обычных и вредоносных приложений, тестирование производится вручную, нет никакой привязки к историческим данным.

В статье [10] описываются способы получения набора релевантных источников дан-

ных трассировки и соответствующий метод обнаружения аномального поведения с помощью полученных данных. Рассмотрены три вида классификаторов на основе машинного обучения: логистическая регрессия, неглубокие нейронные сети (*shallow neural nets*) и методы опорных векторов. Оценивается качество обнаружения аномалий выбранными методами. В работе тестирование приложений проводилось на реальных устройствах, а не эмуляторах, потому что иначе при тестировании не представляется возможным собрать некоторые характеристики, например, использование сети и загруженность процессора.

В статье [11] была проведена классификация ошибок, влияющих на производительность и исследована их "живучесть". Для нахождения примеров ошибок авторы по ключевым словам искали версии приложений с открытым исходным кодом, в которых были исправлены ошибки, связанные с производительностью, а затем вручную классифицировали каждую исправленную ошибку. Авторы работы выложили получившийся набор данных из 500 версий приложений с классифицированными видами ошибок в открытый доступ.

Таким образом, существуют различные инструменты исследовательского тестирования [12, 2]. Однако, нет инструмента исследовательского тестирования, который во время тестирования собирает и каким-то образом использует данные трассировки. Существующие инструменты нацелены на нахождение функциональных ошибок: некорректных завершений, зависаний и т.д. Разрабатываемая в данной работе система же предназначена для поиска проблем с производительностью в приложениях, которые вызваны нефункциональными ошибками в программном обеспечении.

4 Исследование и построение решения задачи

Рассмотрим устройство системы регрессионного тестирования, позволяющей определять и локализовывать проблемы с производительностью во время исследовательского тестирования мобильных приложений (рис. 1).

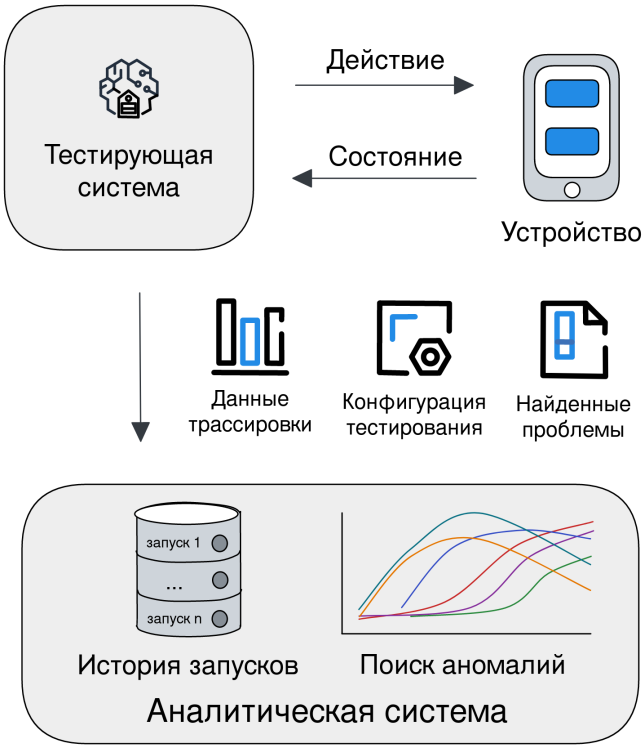

Рис. 1: Схема системы регрессионного тестирования

В систему регулярно загружаются новые версии тестируемого приложения $(v_1, \ldots, v_n)$. На очередной версии приложения производится автоматизированное исследовательское тестирование с фиксированной конфигурацией $\mathcal{C} = (D, S, T, I)$, где D – спецификация устройства, S — стратегия тестирования, T — заданное ограничение по времени тестирования (например, не более 30 минут) и I — интервал между совершением действий (например, 2 секунды). В процессе исследовательского тестирования специальный компонент системы собирает данные трассировки — многомерный временной ряд $U = (\mathbf{u_1}, \mathbf{u_2}, \ldots, \mathbf{u_k})$, где $\mathbf{u_i}$ — временной ряд (вектор значений), показывающий степень загрузки устройства по одной из количественных характеристик, например, по интенсивности использования ресурсов процессора. Результатом проведения исследо-

вательского тестирования является *запуск* $R = (C, U, F)$, состоящий из конфигурации тестирования C, данных трассировки U и набором найденных функциональных проблем F.

В нижней части схемы изображена аналитическая система — часть системы регрессионного тестирования, агрегирующая данные предыдущих запусков и отвечающая за поиск аномалий в новых данных трассировки на основе подобранных соответствующих исторических данных. Для каждой новой версии v' и соответствующего запуска R' аналитическая система определённым образом строит множество истории схожих запусков $H = (R_{n_1}, R_{n_2}, \ldots, R_{n_k})$, на основе которого строится модель ожидаемого поведения для последующего сравнения с ней данных трассировки U'. Если в данных трассировки были найдены аномалии, это может означать присутствие проблем с производительностью в новой версии приложения. В таком случае, пользователю системы регрессионного тестирования сообщается об аномалии с указанием локализованных аномальных участков временного ряда.

Создание описанной системы можно разбить на следующие части:

1. реализация сбора данных трассировки;
2. разработка метода обнаружения аномалий;
3. разработка метода локализации аномальных участков;
4. оценка качества предложенных методов;
5. организация аналитической системы.

Каждая из этих частей описана далее.

4.1 Реализация сбора данных трассировки

Данные трассировки собираются во время исследовательского тестирования и являются частью данных о запуске R, хранимых в исследовательской системе. В данной части работы требовалось провести анализ существующих инструментов для сбора данных трассировки с целью выявления их применимости в контексте исследовательского тестирования. Были рассмотрены следующие варианты сбора данных трассировки: реализация с использованием стандартного ПО Linux; Android Systrace; профилировщики

экосистемы Android; Perfetto. В результате анализа этих способов сбора данных были определены особенности каждого из них и возможность их применимости в контексте работы.

Реализация с использованием стандартного ПО Linux

Поскольку ОС Android базируется на модифицированной версии ядра Linux, представляется возможным использовать некоторые инструменты Linux для сбора данных трассировки. Был реализован экспериментальный модуль для регулярного сохранения показателей использования ресурсов процессора и использования оперативной памяти для заданного процесса. С помощью программного пакета adb (Android Debug Bridge) из экосистемы Android можно подключиться к Android-устройству (к реальному устройству или к эмулятору) и получить доступ к инструментам для отладки устройства, в том числе к командной оболочке. Благодаря этому можно исполнять произвольные команды, используя инструменты Linux для сбора данных трассировки. Программа обрабатывала вывод команды

```
top -p{pid} -b -n 1 -q -o %CPU,%MEM
```

для построения двумерного временного ряда, состоящего из информации об активности использования ресурсов процессора и количества используемой оперативной памяти.

Однако такой подход к реализации сбора данных трассировки сильно ограничен в возможностях. Для каждого вида данных трассировки требуется реализовывать отдельную программу для сбора соответствующих данных. Информацию приходится получать из разных источников, и каждый из них может иметь свои особенности, что затрудняет расширение списка типов собираемых данных. Также эта реализация при очередном запросе замера характеристики использует промежуточные подпроцессы для связи с устройством, что говорит о её неэффективности.

Android Systrace

Systrace — это утилита экосистемы Android, позволяющая собирать данные трассировки во время тестирования приложения. Данная утилита способна собирать различные данные трассировки, используя информацию из ядра ОС, в том числе данные

планировщика, активность работы с файловым хранилищем и информацию о процессах. Данные трассировки представляются в сжатом текстовом формате. Данный инструмент всё реже применяется на практике, и разработчики Android рекомендуют использовать его переработанную и улучшенную версию — инструмент Perfetto (описан далее).

Профилировщики экосистемы Android

Профилировщики позволяют разработчикам приложений записывать данные трассировки прямо во время запуска приложения в среде разработки. Экосистема Android предоставляет профилировщики для отслеживания использования ресурсов процессора, сети и энергопотребления [13]. Профилировщики сильно интрегрированы непосредственно в среды разработки ПО. С одной стороны, это помогает разработчикам приложений видеть результаты профилирования в более удобном для них виде. С другой — это усложняет программное использование профилировщиков в качестве промежуточного инструмента. Они не предоставляют простого автоматизируемого способа получения данных трассировки, поэтому данный метод не был использован в этой работе, однако при необходимости сбора информации об активности использовании сети он может быть задействован. Также стоит отметить, что некоторые виды профилировщиков используют Android Systrace и Perfetto для получения данных.

Perfetto

Perfetto [14] — это кроссплатформенный инструмент трассировки, появившийся в относительно новых версиях Android (начная с Android 10), став заменой Systrace. Perfetto предлагает большее количество источников данных по сравнению с Systrace и позволяет записывать сколь угодно долгие файлы с данными трассировки, используя новый бинарный формат. Perfetto может собирать следующие данные:

- данные планировщика о времени работы процессов;
- информация о системных вызовах;
- частота процессора по ядрам;
- использование оперативной памяти;

- вся информация об использовании динамической памяти;
- данные об энергопотреблении;
- отладочные сообщения и сообщения об ошибках;
- информация о пользовательских событиях Android (ATrace).

В результате обзора существующих средств для получения данных о нагрузках на устройство в качестве такого инструмента был выбран Perfetto. На текущий момент он является де-факто стандартом для эффективного сбора данных о нагрузках на устройство. Он легковесен, поскольку работает на устройстве, а не через промежуточные интерфейсы, предоставляет большое количество источников данных и имеет удобный программный интерфейс. Далее описана специфика работы этого инструмента.

Описание используемых данных трассировки

Perfetto записывает данные в бинарном формате, оптимизированном для легковесной (по затратам вычислительной мощности) записи, чтобы как можно меньше влиять на производительность тестируемой системы. Чтобы получить данные из бинарного файла, нужно воспользоваться обработчиком данных трассировки (Trace Processor). Обработчик данных трассировки разбирает полученный бинарный файл, и предоставляет интерфейс для получения записанной информации с помощью SQL-запросов к набору таблиц, содержащих всю собранную информацию.

Данные трассировки, собираемые Perfetto, представлены в основном в виде *отрезков* (slices) и *счётчиков* (counters). Первые представляют собой набор из интервала времени и описания события, которое в нем было зафиксировано. Вторые — это набор из временно́й метки и числового значения. Данные, представленные в виде счётчиков, больше подходят для формирования временных рядов, однако и по данным-отрезкам можно составить временной ряд. Далее в работе будет приведён пример данных, полученных от Perfetto и описан способ преобразования отрезков и счётчиков во временные ряды.

При исследовании возможностей Perfetto было выявлено, что сбор данных сразу из всех доступных источников может привести к существенному увеличению размера собираемых данных. Поэтому необходимо заранее определить подмножество требуемых

источников данных. Экспериментально было выявлено, что бóльшую часть объёма занимает информация о системных вызовах и информация о пользовательских событиях Android.

В данной работе исследуются следующие данные трассировки для построения на их основе многомерных временных рядов:

- данные планировщика процессов;
- данные об использовании оперативной памяти;
- данные о частоте ядер процессора.

Данные планировщика процессов. Данные планировщика процессов представлены в виде отрезков (рис. 2). Предоставляется полная информация о работе процессов на разных ядрах процессора. События планировщика (см. таблицу 1) представляют собой следующий набор информации: процесс с некоторым именем и приоритетом в момент времени `ts`, проработав время `dur`, был прерван по причине `end_state` (например, вытеснение, блокировка взаимно исключающего доступа или любая другая причина для перемещения исполняемого процесса в очередь).

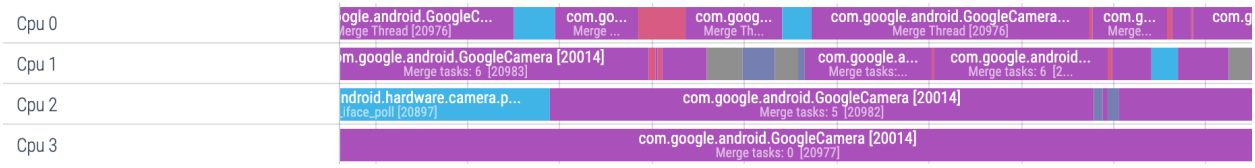

Рис. 2: Визуальное представление данных планировщика, собранных Perfetto

По этой таблице для некоторого интервала времени можно понять, сколько за этот интервал был запущен конкретный процесс, и насколько процессор в целом был загружен в этот интервал. Чтобы преобразовать полученные данные к временному ряду, можно разбить отрезок $[\min\limits_{\texttt{ts} \in \texttt{table}} \texttt{ts}, \max\limits_{\texttt{ts} \in \texttt{table}} \texttt{ts}]$ на некоторое количество интервалов $I_i = [I_i^{\texttt{start}}, I_i^{\texttt{end}}]$ длины `interval_duration` и посчитать сумму длительностей исполнения процессов p_j (любых процессов или какого-то конкретного), попадащих в этот интервал. Причём стоит учесть, что если время завершения процесса $\texttt{ts}(p_j) + \texttt{dur}(p_j) > I_i^{\texttt{start}} + \texttt{interval\_duration}$ (для минимального i, т.ч. $\texttt{ts}(p_j) \in I_i$), то следует записать работу процесса сразу в нескольких интервалах с соответствующими длительностями.

ts	dur	cpu	end_state	thread.name
ts_1	247188	2	S	logd.klogd
261187012418183	12812	2	D	traced_probes0
261187012421099	220000	4	D	kworker/u16:2
261187012430995	72396	2	D	traced_probes1
261187012454537	13958	0	D	traced_probes0
261187012460318	46354	3	S	traced_probes2
261187012468495	10625	0	R	org.schabi.newpipe.HEAD
261187012479120	6459	0	D	traced_probes0
261187012485579	7760	0	R	org.schabi.newpipe.HEAD
261187012493339	34896	0	D	traced_probes0

Таблица 1: Пример строк таблицы с информацией о событиях планировщика

Также стоит отметить, что длина интервала выбирается в соответствии с временем ожидания между действиями инструмента исследовательского тестирования.

Учитывая столбец `proc_name`, из данных планировщика можно извлечь два разных временных ряда: активность использования ресурсов процессора конкретным приложением и активность использования ресурсов процессора всеми процессами на устройстве. Необходимость собирать второй из них обоснована тем, что если проблема, связанная с производительностью, возникнет на уровне операционной системы, то временной ряд, показывающий использование ресурсов процессора конкретным приложением не даст нужной информации для нахождения потенциальных аномалий. Поскольку в данных планировщика содержится информация о ядре процессора (столбец `cpu`), на котором работали процессы, можно строить указанные выше временные ряды не только по всем ядрам вместе, не используя информацию из этого столбца, но и отдельно по каждому ядру процессора.

После указанной предварительной обработки данных планировщика получается два одномерных ряда, которые можно далее комбинировать в том числе с другими данными для получения многомерного, более информативного временного ряда.

Данные об использовании оперативной памяти Perfetto позволяет собирать информацию о количестве используемой памяти (RSS — resident set size, размер занимаего пространства), используя внутренние структуры Android и Linux. Информацию можно получить из обработчика данных трассировки в виде таблицы (см. 2).

ts	value	counter_name	proc_name	pid
6025377029287	9986048.0	mem.rss	/system/bin/init	1
6025377029287	6750208.0	mem.rss	/system/bin/init	149
6025377029287	10059776.0	mem.rss	/system/bin/logd	175
6025377029287	7417856.0	mem.rss	/system/bin/ueventd	151
6025377029287	5492736.0	mem.rss	/system/bin/servicemanager	177
6025377029287	6631424.0	mem.rss	/system/bin/hwservicemanager	178
6025377029287	2875392.0	mem.rss	/system/bin/sh	179
6025377029287	2539520.0	mem.rss	/vendor/bin/qemu-props	180
6025377029287	6475776.0	mem.rss	/vendor/bin/hw/android.hardware.keymaster	196
6025377029287	4968448.0	mem.rss	/system/bin/hw/android.system.suspend	221

Таблица 2: Пример строк таблицы с информацией о размере используемой оперативной памяти

Данные о размере используемой оперативной памяти конкретным процессом сразу представлены в виде временного ряда, поэтому дальнейшая обработка не требуется.

4.2 Метод обнаружения аномалий

Для определения того, является ли запуск $R = (C, U, F)$ аномальным с точки зрения данных трассировки, подсчитывается расстояние между его данными трассировки U и ожидаемой моделью, построенной аналитической системой по подобранным ей историческим данным H. Для построения модели ожидаемого поведения используется понятие барицентра (центроида). Барицентр — это специальное агрегированное представление набора временных рядов, полученное с помощью усреднения, учитывающее специфику временных рядов ([15]).

Для построения барицентра необходимо определить способ вычисления расстояния между двумя временными рядами $t_i = \{t_i^1, \ldots, t_i^m\}$ и $t_j = \{t_j^1, \ldots, t_j^m\}$. Тривиальный подход заключается в том, чтобы взять евклидово расстояние между ними:

$$\left\| t_i - t_j \right\|_2 = \sqrt{\sum\nolimits_{k=1}^{m} \left(t_i^k - t_j^k\right)^2}$$

Этот метод расчёта расстояний между временными рядами показывает плохие результаты, поскольку он не учитывает специфики временных рядов. В частности, евклидова метрика неустойчива к сдвигу одного временного ряда относительного другого, а

также к их растяжению и сужению. Например, если взять временной ряд и его же, но немного смещённый по времени, евклидова метрика покажет большую разницу. Наряду со случайными незначительными колебаниями, эта закономерность характерна для временных рядов использования ресурсов, полученных в ходе автоматизированного исследовательского тестирования.

Ввиду упомянутых причин, для сравнения временных рядов используется метрика динамической трансформации временной шкалы (DTW, dynamic time warping, [16]). В имеющихся обозначениях определим расстояние между t_i и t_j:

$$\left\|t_i - t_j\right\|_{\text{DTW}} = \min_{\pi} \sqrt{\sum_{(x,y)\in\pi} (t_i^x - t_j^y)},$$

где $\pi = [\pi_0, \ldots, \pi_K]$ — это путь, который удволетворяет следующим свойствам:

- π — это множество пар индексов $\pi_k = (x, y)$ временных рядов t_i и t_j соответственно.
- $\pi_0 = (0, 0)$, $\pi_K = (n-1, n-1)$.
- $\forall k > 0$ $\pi_k = (x_k, y_k)$ и $\pi_{k-1} = (x_{k-1}, y_{k-1})$ связаны соотношениями: $x_{k-1} \leq x_k \leq x_{k-1} + 1$ и $y_{k-1} \leq y_k \leq y_{k-1} + 1$.

Для вычисления барицентра используется метод усреднения с помощью динамической трансформации временной шкалы (*DTW Barycenter Averaging*, *DBA*, [15]).

В заданных выше обозначениях, имея новый запуск R', можно получить его данные трассировки U', набор истории запусков $H = (R_{n_1}, \ldots, R_{n_k})$ и соответствующих исторических данных — временных рядов $U(H) = \{U_{n_1}, \ldots, U_{n_k}\}$. Строится барицентр $B(H)$ временных рядов из множества $U(H)$. После этого можно посчитать расстояния между построенным барицентром и историческими данными.

$$D(H) = \{\left\|u - B(H)\right\|_{\text{DTW}},\ u \in U(H)\}$$

Данные трассировки запуска R' признаются аналитической системой аномальными на основе метода интерквартильного размаха (interquartile range method). Вычисляется расстояние между данными трассировки U' и барицентром: $d = \left\|U' - B(H)\right\|_{\text{DTW}}$. Метод интерквартильного размаха основывается на квартилях уровня 25 и 75 множества $D(H)$: Q_1 и Q_3 соответственно. *Интерквартильное расстояние* находится по формуле:

$$\text{IQR} = Q_3 - Q_1$$

Тогда временной ряд U' считается аномальным, если его расстояние d до барицентра $B(H)$ больше, чем $Q_3 + \omega \cdot \text{IQR}$. Здесь ω — это гиперпараметр, влияющий на чувствительность метода обнаружения аномалий. Его значение влияет на величину допустимого отклонения от ожидаемого поведения. Было протестировано несколько значений для ω (см. главу 5).

На рис. 3 приведен пример работы метода интерквартильного размаха в виде графика специального вида — диаграммы размаха (*box plot*). На графике границы прямоугольника, т.н. ящика, соответствуют Q_1 и Q_3. Верхняя и нижние линии составляют упомянутый выше отрезок. Во всех случаях прямоугольники одинаковы, поскольку строятся по набору исторических данных, однако на графике они выглядят разными из-за разных масштабов вертикальных осей. Графики построены на 6 примерах, построенных на основе данных трассировки, собранных на приложениях с внедрёнными проблемами, влияющими на производительность, и приложениях, не имеющих соответствующих проблем. В последнем случае, данные трассировки, используемые для иллюстрации, не пересекаются с набором исторических данных. Подробнее процесс оценки качества обнаружения аномалий описан в главе 5.

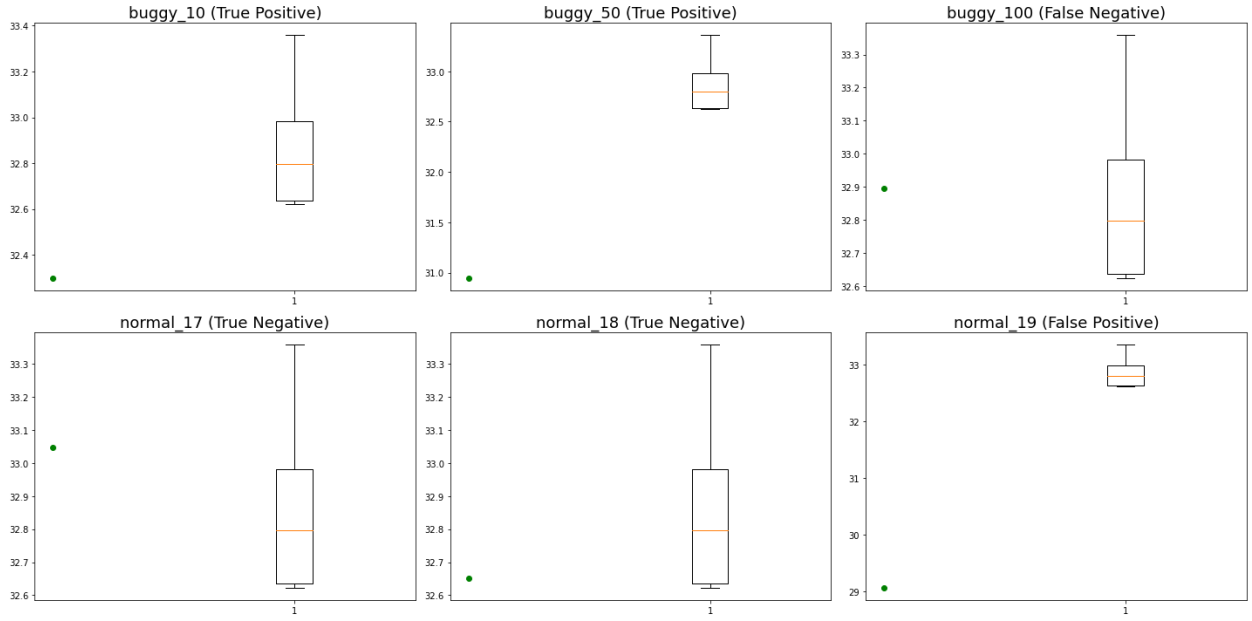

Рис. 3: Результат работы метода интерквартильного размаха для приложения OmniNotes

Выявление в многомерном временном ряду данных трассировки происходит раздельно, затем результаты агреггируются следующим образом: при обнаружении аномальности временного ряда с данными об использовании оперативной памяти производится повторная проверка их с помощью специальной эвристики. Для текущего временного ряда и `k` временных рядов из исторического набора вычисляется набор перцентилей $\{p_0, p_{25}, p_{50}, p_{75}, p_{100}\}$. Далее используется основанная на плотности пространственная кластеризация для приложений с шумами (Density-based spatial clustering of applications with noise, DBSCAN [17]) для полученных наборов характеристик использования памяти за каждый рассматриваемый запуск (один проверяемый и k из исторических данных). Если в результате кластеризации целевой временной ряд не принадлежит никакому кластеру, то он считается аномальным, т.к. его характеристики достаточно сильно отличаются от аналогичных характеристкик, полученных для исторических данных.

4.3 Метод локализации аномальных участков

Если данные трассировки U признаны аномальными, то производится локализация аномальных участков на тех одномерных временных рядах $T = U_i$, которые являются аномальными по сравнению с барицентром $B = B(H)$.

Для локализации аномальных участков временного ряда используются матричные профили [18, 19, 20, 21]. Введём обозначения. Временные ряды — упорядоченные последовательности вещественных чисел длины n будем обозначать как $T \in \mathbb{R}^n$ и $H \in \mathbb{R}^n$. Подпоследовательностью $T_{i,m} \in \mathbb{R}^m$ временного ряда T называется последовательное подмножество значений T длины m, которое начинается с позиции i: $T_{i,m} = [t_i, t_{i+1}, \ldots, t_{i+m-1}]$, аналогично для $B_{i,m}$. *Матричным профилем* (matrix profile) временного ряда T, использующим временной ряд B как вспомогательный, назовём вектор $M(T, B) \in \mathbb{R}^{n-m+1}$, в котором на i-той позиции хранится минимальное по j расстояние от фиксированной подпоследовательности $T_{i,m}$ до $B_{j,m}$. Таким образом, чем больше значение матричного профиля на i-той позиции, тем больше подпоследовательность $T_{i,m}$ аномальна с точки зрения временного ряда B.

Для выявления аномальных подпоследовательностей необходимо найти *разногласия* (discords) — участки временного ряда, в которых значения матричного профиля больше определённого порога. На рисунке 4 разногласия — это участки временного ряда, где величина матричного профиля выше заданного порга, обозначенного пунктиром. В качестве порогового значения используются величины $M_{0,90}$ и $M_{0,95}$ — квантили множества значений матричного профиля двух уровней. Две величины нужны для указания степени уверенности метода в том, что на данный участок является аномальным.

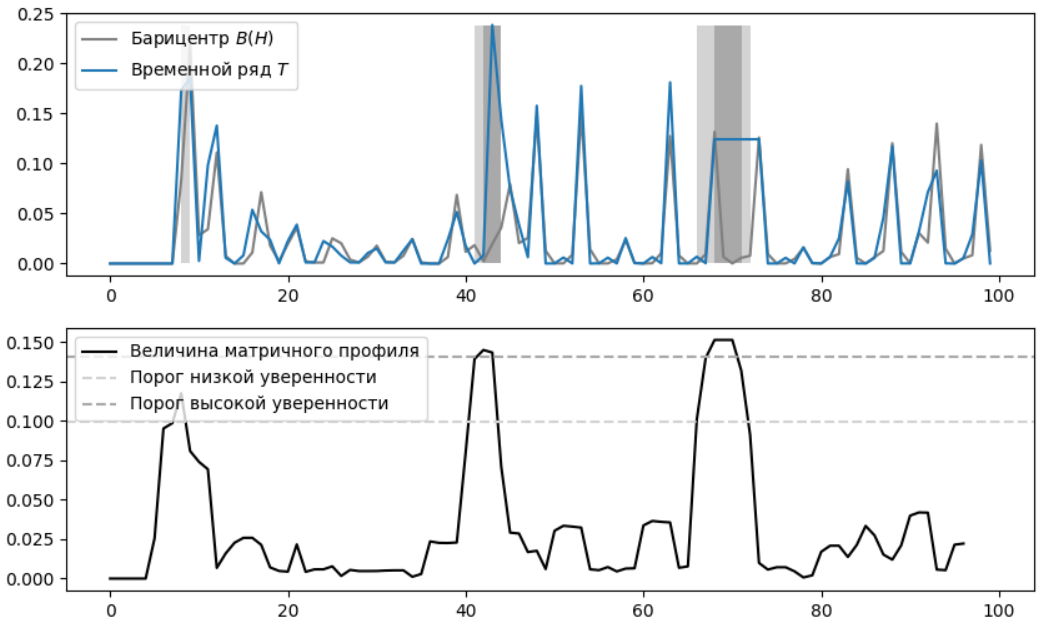

Рис. 4: Пример вычисленного матричного профиля на данных трассировки

4.4 Оценка качества предложенных методов

Автоматизированное исследовательское тестирование производилось с помощью инструмента, разработанного в Институте системного программирования им В.П. Иванникова РАН.

4.4.1 Оценка метода обнаружения аномалий

Метод обнаружения аномалий был оценен на следующих типах данных:

- данные, собранные на приложениях с искусственно внедрёнными проблемами с производительностью;
- данные, собранные на обычных приложениях, но впоследствии модифицированные особым образом.

Также были проведены попытки оценить работу метода классификации на данных, собранных на приложениях с открытым исходным кодом, в старых версиях которых

есть известные проблемы с производительностью. Список таких проблем был собран авторами [11]. С помощью поиска по ключым словами среди них удалось найти 141 версию 31 приложения не старше 2021 года. Однако оценить работу метода на данных, собранных во время исследовательского тестирования на этих версиях приложений не удалось, т.к. реальноая проблема с производительностью требует тщательного анализа и для гарантированного воспроизведения в процессе исследовательского тестирования требуется закладывать большое количество времени, поэтому данные для оценки метода в этой работе ограничиваются упомянутыми вариантами.

Для внедрения искусственных проблем с производительностью в приложения с открытым исходным кодом были выбраны приложения Booking.com, RedReader, Forecastie [22]. Подробная таксономия проблем с производительностью, созданной авторами работы [11], позволяет заключить, что сложные избыточные вычисления являются одной из самых частых проблем с производительностью в реальных приложениях. Избыточные вычисления вызывают нагрузки на устройство, в результате чего пользовательский опыт может ухудшиться из-за возможных проблем, возникающих из-за излишнего потребления ресурсов устройства. Для симуляции описанных ситуаций были созданы несколько версий выбранных приложений, в каждую из которых были внедрены избыточные вычисления разной степени интенсивности.

Для создания второго типа данных трассировки для оценки классификации, в которых обычные данные трассировки модифицируются специальным образом для имитиации аномалий, было проделано следующее. В статье [23] авторы проанализировали природу утечек памяти. Для имитации утечек памяти были использованы два паттерна поведения использования памяти из этой статьи: линейно растущий паттерн и пилообразный паттерн. На рисунке 5 можно увидеть пример модифицированного временного ряда, где был воспроизведен пилообразный паттерн утечки памяти.

Для формирования набора данных оба паттерна утечки были применены на данных трассировки, полученных с приложений Booking.com, Ebay и GnuCash.

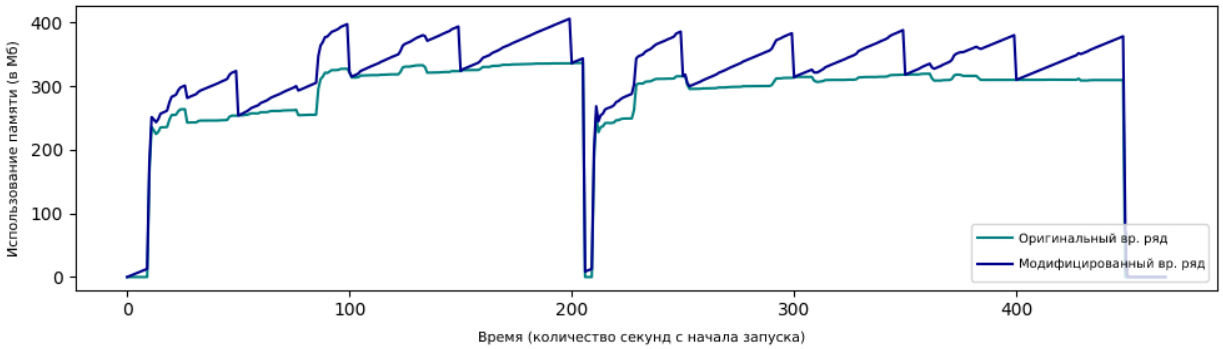

Рис. 5: Пример модифицированного временного ряда

Результаты оценки классификации данных трассировки представлен в главе 5.

4.4.2 Оценка качества локализации аномалий

Для проверки работоспособности метода локализации аномалий он был протестирован тремя пользователями на данных трассировки, полученных во время исследовательского тестирования приложений, в которые были внедрены проблемы с производительностью. Поскольку метод классификации аномалий уже оценивается в этом работе, а целью данной части работы является проверка качества работы локализации аномалий, рассматривались данные трассировки, которые действительно являются аномальными, этим достигается оценка именно метода локализации аномалий.

Участникам оценки было предложено изучить по 12 запусков R_i, которые были загружены в систему регрессионного тестирования вместе с историческими данными, представляющими собой данные трассировки, собранные на таком же приложении, но без наличия проблем с производительностью. Каждый запуск R_i представляет собой отчёт о сеансе исследовательского тестирования некоторой версии приложения $A \in$ (OmniNotes, Forecastie, RedReader). Для каждого приложения таких версий несколько, они отличаются тяжестью внедрённых избыточных вычислений. Вместе с запуском R_i участник получал информацию о том, в какую часть приложения были внедрены проблемы с производительностью для каждого приложения. Используемый инструмент исследовательского тестирования предоставляет возможность проследить путь исследования для каждого запуска. Используя эту информацию и графики данных трассировки с локализованными аномальными участками, участник сопоставлял участки графика

данных трассировки с происходящим на экране, это позволило оценить, содержит ли исследуемый в данный момент экран приложения проблемы с производительностью.

Участникам предлагалось оценить для каждого рассмотренного запуска следующие величины:

- TP: количество участков данных трассировки на экранах приложения со внедрёнными проблемами с производительностью, которые были локализованы методом как аномальные;
- FP: количество участков данных трассировки на экранах приложения без внедрённых проблем с производительностью, которые были локализованы методом как аномальные;
- FN: количество участков данных трассировки на экранах приложения со внедрёнными проблемами с производительностью, которые не были локализованы методом как аномальные.

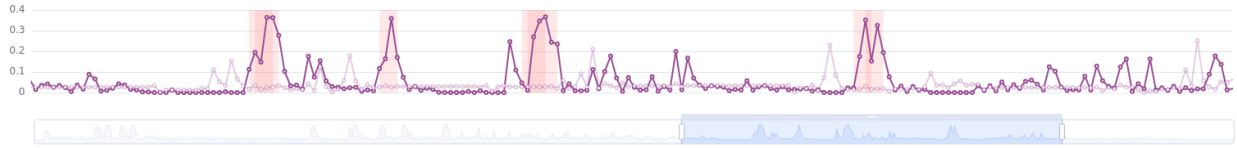

Рис. 6: Пример работы метода локализации на фрагменте данных трассировки приложения RedReader

На рисунке 6 приведён пример работы метода локализации на фрагменте данных трассировки для приложения RedReader. Согласно информации о пути исследования, представленную инструментом исследовательского тестирования, и информации о расположении внедрённых проблемах с производительностью в приложении, можно заключить, что первые 3 локализованные участка представляют верно распознанные участки (TP), а четвёртый — ложноположительный результат (FP).

	TP (сумма)	FP (сумма)	FN (сумма)
RedReader	95	31	5
Forecastie	110	18	0
OmniNotes	84	33	0

Таблица 3: Результаты оценки качества локализации аномалий тремя пользователями

В таблице 3 приведены результаты оценки качества метода локализации аномальных отрезков. По результатам оценки были сделаны следующие выводы. Количество участков данных трассировки на экранах приложения со внедрёнными проблемами с производительностью, которые не были локализованы методом как аномальные (велчина FN), практически равно нулю — если в данных трассировки есть аномалия, то метод практически всегда локализует её. Однако величина FP, которая показывает количество ложных срабатываний, отлична от нуля. Было отмечено, что примерно половина из таких случаев происходит в начале или в конце сеанса тестирования. Это можно обосновать тем, что в эти периоды тестирования инструмент исследовательского тестирования производит специальные действия, связанные с началом и завершением тестирования. Эти периоды разумно исключить из рассмотрения инструментом локализации. Также стоит отметить, что в качестве положительного вердикта о локализации аномалии при оценке качества учитывались оба порога уверенности ($M_{0,90}$ и $M_{0,95}$).

4.5 Организация аналитической системы

В данном разделе описываются подробности создания аналитической системы, представленной в нижней части рисунка 1. Аналитическая система хранит в себе историю запусков R_i, чтобы впоследствии для нового запуска $R' = (C', U', F')$ подобрать множество схожих запусков $H = (R_{n_1}, R_{n_2}, \ldots, R_{n_k})$, на основе которого строится модель ожидаемого поведения для последующего сравнения с ней данных трассировки U'.

Подбор данных трассировки

В процессе создания и тестирования системы регрессионного тестирования было выявлено, что существуют приложения, для которых процесс исследовательского тестирования может быть сильно недетерминированным, то есть при одной и той же

конфигурации тестирования C тестирование может пойти по разным путям и данные запусков, в том числе данные трассировки U, будут сильно отличаться. Это характерно не только для приложений с динамическим контентом, в которых содержимое экрана регулярно меняется (например, лента новостей), но и для некоторых других приложений из-за особенностей принятия решения о выборе следующего действия. Примером последнего может служить приложение Face App, для которого далее на рисунке 7 с помощью введённой в этом разделе метрики схожести запуском показано, что множество запусков разбивается на кластеры, внутри которых запуски схожи.

Поскольку данные трассировки U_{n_i} должны являться образцом для новых данных трассировки U' , то для правильной классификации данных трассировки во втором из описанных случаев будет полезно подбирать в качестве исторических данных трассировки U_{n_i} данные, соответствующие тем запускам R_{n_i}, путь исследования приложения которых максимально схож. В используемом инструменте исследовательского тестирования можно получить следующую характеристику. *Путём исследования* запуска R назовём последовательность $P(R) = \{\text{hash}(S_1), \ldots, \text{hash}(S_m)\}$, в которой $\text{hash}(S_i)$ — значение хеш-функции от строкового представления S_i состояния приложения после очередного действия тестирующей системы. Состояние приложения характеризуется строкой, составленной из описания особенностей состояния приложения. Путь $P(R)$ характеризует то, как проходил запуск R. Если у двух запусков R_1 и R_2 схожи пути исследования, то у них должны быть похожие данные трассировки, так как исследование происходило по более схожему сценарию, чем если бы пути были сильно различны. В качестве метрики схожести путей исследования использовался алгоритм Рэтклиффа-Обершелпа. Значение данной метрики $D(R_1, R_2)$ находится в пределах от 0 до 1.

Примеры попарных значений метрики схожести путей для некоторых запусков приведены на рисунках 7 и 8. Таким образом, в случае, когда запуски не отличаются, введённое ограничение по схожести запусков не влияет на подбор исторических данных, а в случае, когда запуски отличаются друг от друга по схожести, метрика позволяет составить набор исторических данных оптимальным образом, чтобы построенный барицентр $B(H)$ отражал информацию о данных трассировки в максимально похожих сценариях тестирования.

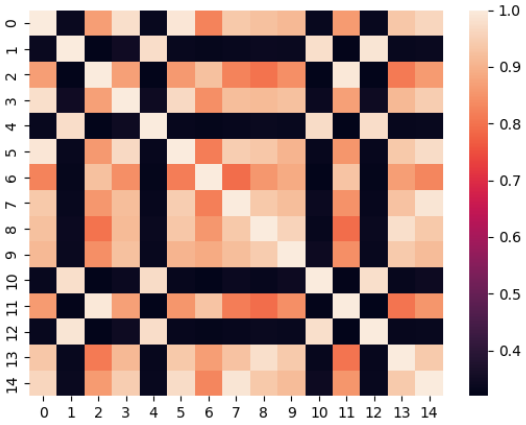

Рис. 7: Пример матрицы схожести 15 запусков для приложения FaceApp: есть несколько кластеров, внутри которых запуски схожи

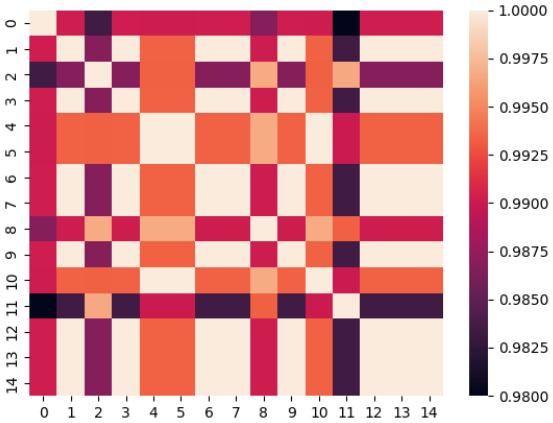

Рис. 8: Пример матрицы схожести 15 запусков для приложения WSJ Reader: все запуски имеют коэффициент схожести не меньше 0,98

Итак, для вхождения запуска R_i в формируемую историю запусков $H = (R_{n_1}, \ldots, R_{n_k})$ для нового запуска $R' = (C', U', F')$ требовалось выполнение следующих условий:

1. Конфигурации параметров исследовательского тестирования C_i и C' совпадают (например, ограничение по времени тестирования и интервал между совершением действий).

2. Множество выявленных функциональных проблем F_i пусто, то есть в качестве исторических данных не рассматриваются запуски, в процессе которых были обнаружены функциональные проблемы, например, некорректное завершение программы во время тестирования.

3. Метрика схожести путей исследования запусков $D(R_1, R_2) \geq 0,8$.

Чем больше запусков будет содержаться в историческом наборе H, тем более точно агрегированное представление данных трассировки $B(H)$ будет отражать ожидаемые данные трассировки. Однако, для адаптивности системы к постепенным изменениям работы приложения по мере его разработки требуется ограничить набор входящих в H запусков до k последних. Такое ограничение обеспечивает построение барицентра $B(H)$ лишь по недавным, более актуальным данным. В качестве k было выбрано значение 12.

Интерфейс аналитической системы

Аналитическая система предоставляет пользователю интерфейс для просмотра загруженных запусков в систему. Если данные трассировки U_i запуска R_i считаются аномальными, то пользователю показывается предупреждение о том, что в соответствующей версии приложения v_i могут быть проблемы с производительностью. Система помогает пользователю обнаружить найти отличия в данных трассировки U' от агрегированных исторических данных $B(H)$. После выбора интересующего запуска R' пользователь может посмотреть различия на графиках с выделенными участками, где наблюдается аномалия. Поскольку U' и $B(H)$ являются многомерными временными рядами, а в процессе классификации данных трассировки каждый тип данных трассировки рассматривается на предмет аномальности отдельно (и лишь потом агрегируется), аномальные участки, вычисленные с помощью матричных профилей, показываются только на графиках, вид данных трассировки которых оказался аномальным в процессе классификации данных трассировки.

На рисунке 9 показана часть интерфейса аналитической системы. Показан просмотр данных трассировки запуска R_0. Для тех графиков, которые были признаны аномальными с точки зрения аналитической системы, показан индикатор обнаружения аномалии и для них производится локализация аномальный участков. Пользователю предоставляется удобный интерактивный многомерный график с отмеченными "подозритель-

ными” с точки зрения системы участками. Также в интерактивном режиме показывается информация о точном значении показателей загруженности устройства в каждый момент времени и о времени с начала тестирования. Эта информация, в совокупности с другой информацией, собираемой инструментом исследовательского тестирования, вроде видеозаписи тестирования, снимков экрана и журналом системных сообщений (system messages logs), может помочь специалисту, разрабатывающему приложение, обнаружить причину регрессии производительности и исправить её.

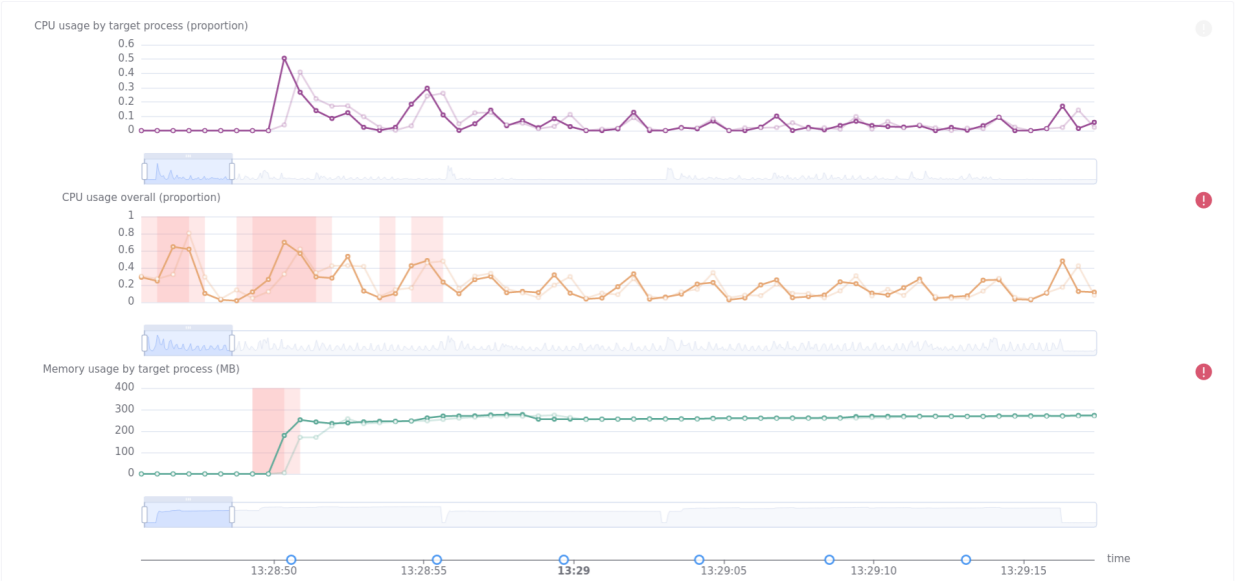

Рис. 9: Пример локализации аномальных участков на многомерном временном ряде данных трассировки приложения Booking.com

5 Описание практической части

5.1 Детали реализации

Приложения с открытым исходным кодом, которые были взяты за основу для модификации, чтобы получить набор приложений, имеющих проблемы с производительностью, реализованы на языках программирования Java 8, Java 11 и Kotlin 1.6.

В работе использовался инструмент автоматизированного исследовательского тестирования, разработанный в Институте системного программирования им. В.П. Иванникова РАН. Данный инструмент имеет возможность расширения его возможностей путём добавления т.н. адаптеров — специальных модулей, которые запускаются во время работы основной программы. Для сбора необходимых в контексте работы данных трассировки был реализован модуль `PerfettoAdapter`, который с помощью ранее упомянутого в работе программного интерфейса adb подключается к тестируемому устройству, обрабатывает конфигурационный файл, содержащий список источников данных трассировки и запускает Perfetto с нужными параметрами. После завершения исследовательского тестирования по установленному лимиту времени или по исчерпанию возможных состояний целевого приложения, `PerfettoAdapter` с помощью adb загружает файл с устройства на компьютер и удаляет файл с устройства.

Экспериментально было выявлено, что при сборе всех возможных данных получаются файлы огромного размера: за 10 минут тестирования собирается больше 4 Гб информации. В итоговом варианте конфигурации (подробнее — в прошлой главе) удалось сократить размер данных за то же время до файла порядка 200 Мб.

Perfetto работает для любой версии Android, однако на старых версиях Android (Android 9 и ниже) запуск этого инструмента подразумевает более сложный набор действий, включая скачивание самого инструмента. Поскольку в цели работы не входит написание универсального инструмента для сбора данных трассировки, было принято решение ограничиться новыми версиями Android и реализовать модуль только для них.

Адаптер, как и соответствующая часть инструмента ИСП РАН, реализован на языке Python 3.9 с использованием широких возможностей стандартной библиотеки языка. Для обработки и визуализации данных использовались библиотеки `pandas`, `numpy` и `matplotlib` ([24], [25], [26]).

Для автоматизированного многократного запуска инструмента ИСП РАН и сбора

данных трассировки в процессе тестирования был написан модуль, который последовательно запускает инструмент на разных версиях приложений и выбирает из полученных выходных данных только нужные для работы бинарные файлы с собранными Perfetto данными.

Для преобразования данных из бинарного формата Perfetto в формат DataFrame библиотеки `pandas` был реализован модуль с использованием официальной программной библиотеки `perfetto` для Python 3. Как уже было упомянуто в работе, класс `TraceProcessor` позволяет получать данные, собранные Perfetto во время исследовательского тестирование с помощью SQL запросов (рис. 10).

```
tp = TraceProcessor(file_path='./stats.perfetto-trace')
df = tp.query('''
        select
            c.ts, c.value, t.name as counter_name,
            p.name as proc_name, p.pid
        from
                    counter as c
            left join
                process_counter_track as t on c.track_id = t.id
            left join
                process as p using (upid)
        where
                t.name = 'mem.rss'
            and
                proc_name = 'it.feio.android.omninotes.alpha'
    ''').as_pandas_dataframe()
```

Рис. 10: Пример получения DataFrame из perfetto-trace файла

Далее программа выполняла постобработку полученных данных, описанную в прошлой секции работы. Например, из данных планировщика путём преобразования полученного DataFrame средствами библиотек `pandas` и `numpy` были получены временные ряды для общей активности использования ресурсов процессора и для активности использования ресурсов процессора конкретным приложением по интервалам в одну секунду. Также временные метки в данных Perfetto представлены в виде количества

наносекунд, прошедших с некоторого произвольного (ненулевого) момента времени, поэтому нужно было привести время к одной шкале сдвигом.

Поскольку оптимальное время ожидания между действиями в используемом инструменте исследовательского тестирования равно двум секундам, в данной работе `interval_duration`, описанный в прошлой главе и используемый для получения временных рядов из данных планировщика, был принят равным одной секунде для большей гранулярности. Это, с одной стороны, достаточно небольшой промежуток времени для отслеживания изменения интенсивности использования ресурсов во времени, с другой стороны — достаточно длинный для того, чтобы избежать резких перепадов во временных рядах из-за слишком сильного разбиения.

Для реализации системы регрессионного тестирования необходимо было реализовать две части: backend-часть для обработки загруженных запусков и frontend-часть, представляющую собой пользовательский интерфейс для взаимодействия с системой. В backend-часть входит подбор исторических данных, построение ожидаемой модели поведения приложения и локализация аномальных участков. Она реализована на Python 3.9 с использованием библиотек FastAPI, tslearn [27] и stumpy [28]. Для frontend-части был использован Vue.js 3 вместе с библиотекой Apache ECharts для построения интерактивных графиков.

По окончании сеанса исследовательского тестирования данные трассировки, собранные модулем `PerfettoAdapter`, передаются в аналитическую систему. Данные трассировки, вместе с другими данными о запуске, сохраняются в NoSQL СУБД. Далее они используются для построения модели ожидаемого поведения для проверки новых данных трассировки на аномальность.

Суммарно реализация `PerfettoAdapter` и всей аналитической системы представляет собой около 2000 строк программного кода.

5.2 Оценка качества обнаружения аномалий

В главе 4 был подробно описан сбор данных для оценки качества обнаружения аномалий. В данной секции приводятся результаты данной оценки. Для оценки использовались данные трассировки, полученные в результате автоматизированного исследовательского тестирование. Суммарное время тестирования превышает 60 часов.

Для каждого приложения $A \in$ (OmniNotes, Forecastie, RedReader, Booking.com, Ebay)

Приложение	**Количество нормальных временных рядов**	**Количество аномальных временных рядов**	**Метод получения аномального временного ряда**
Booking.com	31	19	Модификация временных рядов
Ebay	31	19	
GnuCash	31	19	
OmniNotes	20	8	Внедрённые проблемы с производительностью
Forecastie	20	8	
RedReader	20	8	

Таблица 4: Описание полученного набора данных для оценки классификации

имеется набор временных рядов $T_A = \{t_{\text{normal}_1}, \ldots, t_{\text{normal}_{k_A}}, t_{\text{anomalous}_1}, \ldots, t_{\text{anomalous}_{m_A}}\}$, показанный в таблице 4. Набор исторических данных размером $k = 12$ выбирался из нормальных данных трассировки случайным образом. Остальные временные ряды из T_A были использованы как целевые для проверки качества обнаружения аномалий с выбранным набором исторических данных. Экспериментально было отмечено, что результаты классификации могут незначительно зависеть от выбора исторических данных, поэтому для стабильности метрики были посчитаны как среднее арифметическое между метриками, полученными в процессе 30 разных запусков поиска аномалий с разными историческими данными. Также было оценено влияние параметра ω на результат. Итоговое количество наборов входных данных для фиксированного ω равно:

$$\sum_A \left[(k_A + m_A - \texttt{hist\_size}) \cdot \texttt{iterations} \cdot p_a\right] = 6300,$$

где $\texttt{hist\_size} = 12$, $\texttt{iterations} = 30$ и p_A означает число одномерных временных рядов, которые могут быть аномальными по построению данных для оценки: $p_A = 1$ для первых трёх приложений из таблицы 4 и $p_A = 3$ для остальных трёх приложений.

Для получения количественной оценки качества обнаружения аномалий были посчитаны метрики Precision, Recall, F_1-score для двух реализованных методов.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

	TP	TN	FP	FN	Precision	Recall	F_1-score
$\omega = 0{,}0$	2500	1669	1481	650	0,628	0,794	**0,701**
$\omega = 0{,}5$	2169	2094	1056	981	0,673	0,689	0,680
$\omega = 1{,}0$	1979	2396	754	1171	0,724	0,628	0,672
$\omega = 1{,}5$	1688	2550	600	1462	0,738	0,536	0,621

Таблица 5: Результаты оценки метода обнаружения аномалий

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$\text{F}_1\text{-score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

За TP, FP, FN, TN принимается соответственно количество правильно классифицированных случаев, где присутствует аномалия, количество ошибочно найденных аномалий, количество ненайденных аномалий, количество правильно классифицированных случаев, где аномалии нет. Значения округлены до трёх знаков после запятой.

Результаты оценки качества обнаружения аномалий представлены в таблице 5. Значения первых четырёх колонок показывают количество одномерных временных рядов, на которых производился поиск аномалий. Для данных приложений $0,0$ — оптимальное значение параметра ω с полученным $\text{F}_1\text{-score} = 0,701$. С ростом ω уменьшается значение метрики Recall, но растет значение метрики Precision. В будущем эти результаты могут быть улучшены засчёт разработки дополнительных предикатов, которые будут работать с конкретными новыми типами данных трассировки, используя их специфику.

6 Заключение

В данной работе были предложены подходы для обнаружения и локализации аномальных нагрузок на устройство во время исследовательского тестирования мобильных приложений. Эти подходы могут быть применены для поиска регрессий производительности в приложениях. На основе предложенных методов построена система регрессионного тестирования, работающая с инструментом исследовательского тестирования, разработанным в Институте системного программирования им. В.П. Иванникова РАН. Построенная система позволяет использовать данные о предыдущих схожих запусках для построения модели ожидаемого поведения приложения для нового запуска. Таким образом, при обнаружении аномалии система сможет сфокусировать внимание экспертов на "подозрительных" участках данных трассировки нового запуска, потенциально помогая находить и исправлять проблемы с производительностью в приложениях. Была проведена оценка обоих предложенных методов. Результаты оценки показали, что предложенный подход может быть успешно применён для поиска проблем с производительностью в приложениях.

Таким образом, были решены все поставленные задачи. В будущем планируется расширить набор собираемых видов данных трассировки (например, рассмотреть интенсивность использования сети и данные об энергопотреблении устройства), а также, для улучшения метода обнаружения аномалий, разработать дополнительные предикаты, которые будут работать с определёнными типами данных трассировки и использовать их специфику.

На основе данной работы была опубликована статья [29]: *D. Mikhaltsov and K. Sorokin, "Detecting anomalous device loads during exploratory testing of mobile applications", 2022 Ivannikov Ispras Open Conference (ISPRAS).*

Список литературы

[1] *Nistor, Adrian.* Discovering, reporting, and fixing performance bugs / Adrian Nistor, Tian Jiang, Lin Tan. — 2013. — 05. — Pp. 237–246.

[2] DroidBot: a lightweight UI-Guided test input generator for android / Yuanchun Li, Ziyue Yang, Yao Guo, Xiangqun Chen // 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C). — 2017. — Pp. 23–26.

[3] Humanoid: A Deep Learning-Based Approach to Automated Black-box Android App Testing / Yuanchun Li, Ziyue Yang, Yao Guo, Xiangqun Chen // 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). — 2019. — Pp. 1070–1073.

[4] *Eskonen, Juha.* Automating GUI Testing with Image-Based Deep Reinforcement Learning / Juha Eskonen, Julen Kahles, Joel Reijonen // 2020 IEEE International Conference on Autonomic Computing and Self-Organizing Systems (ACSOS). — 2020. — Pp. 160–167.

[5] Improving Automated GUI Exploration of Android Apps via Static Dependency Analysis / Wunan Guo, Liwei Shen, Ting Su et al. // 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME). — 2020. — Pp. 557–568.

[6] *Filonov, Pavel.* RNN-based Early Cyber-Attack Detection for the Tennessee Eastman Process. — 2017.

[7] Statistical techniques for online anomaly detection in data centers / Chengwei Wang, Krishnamurthy Viswanathan, Lakshminarayan Choudur et al. // 12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops. — 2011. — Pp. 385–392.

[8] A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivariate Time Series Data / Chuxu Zhang, Dongjin Song, Yuncong Chen et al. // *Proceedings of the AAAI Conference on Artificial Intelligence.* — 2019. — 07. — Vol. 33. — Pp. 1409–1416.

[9] Smart malware detection on Android / Laura Gheorghe, Bogdan Marin, Gary Gibson et al. // *Security and Communication Networks*. — 2015. — 09. — Vol. 8. — Pp. n/a–n/a.

[10] Detection of Anomalous Behavior in Modern Smartphones Using Software Sensor-Based Data / Victor Vladareanu, Valentin Voiculescu, Vlad-Alexandru Grosu et al. // *Sensors*. — 2020. — 05. — Vol. 20.

[11] Investigating types and survivability of performance bugs in mobile apps / Alejandro Mazuera-Rozo, Catia Trubiani, Mario Linares-Vásquez, Gabriele Bavota // *Empirical Software Engineering*. — 2020. — 05. — Vol. 25. — Pp. 1–43.

[12] *Choudhary, Shauvik Roy*. Automated Test Input Generation for Android: Are We There Yet? — 2015.

[13] Profilers — Android Ecosystem Documentation. — `https://developer.android.com/studio/profile/android-profiler`.

[14] Perfetto - System profiling, app tracing and trace analysis. — `https://perfetto.dev/`.

[15] *Petitjean, François*. A global averaging method for dynamic time warping, with applications to clustering / François Petitjean, Alain Ketterlin, Pierre Gançarski // *Pattern Recognition*. — 2011. — Vol. 44, no. 3. — Pp. 678–693. `https://www.sciencedirect.com/science/article/pii/S003132031000453X`.

[16] Dynamic Time Warping (DTW) // Encyclopedia of Biometrics / Ed. by Stan Z. Li, Anil Jain. — Boston, MA: Springer US, 2009. — Pp. 231–231. `https://doi.org/10.1007/978-0-387-73003-5_768`.

[17] A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise / Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu // Proceedings of the Second International Conference on Knowledge Discovery and Data Mining. — KDD'96. — AAAI Press, 1996. — P. 226–231.

[18] Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets / Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova et al. // 2016 IEEE 16th International Conference on Data Mining (ICDM). — 2016. — Pp. 1317–1322.

[19] *Yeh, Chin-Chia Michael.* Matrix Profile III: The Matrix Profile Allows Visualization of Salient Subsequences in Massive Time Series / Chin-Chia Michael Yeh, Helga Van Herle, Eamonn Keogh // 2016 IEEE 16th International Conference on Data Mining (ICDM). — 2016. — Pp. 579–588.

[20] *Yeh, Chin-Chia Michael.* Matrix Profile VI: Meaningful Multidimensional Motif Discovery / Chin-Chia Michael Yeh, Nickolas Kavantzas, Eamonn Keogh // 2017 IEEE International Conference on Data Mining (ICDM). — 2017. — Pp. 565–574.

[21] Time Series Joins, Motifs, Discords and Shapelets: A Unifying View That Exploits the Matrix Profile / Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova et al. // *Data Min. Knowl. Discov.* — 2018. — jan. — Vol. 32, no. 1. — P. 83–123. https://doi.org/10.1007/s10618-017-0519-9.

[22] pcqpcq/open-source-android-apps: Open-Source Android Apps. — https://github.com/pcqpcq/open-source-android-apps.

[23] Memory Leak Detection Algorithms in the Cloud-based Infrastructure / Anshul Jindal, Paul Staab, Pooja Kulkarni et al. — 2021. — 06.

[24] *Reback, Jeff.* pandas-dev/pandas: Pandas 1.4.2. — 2022. — . https://doi.org/10.5281/zenodo.6408044.

[25] Array programming with NumPy / Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt et al. // *Nature.* — 2020. — . — Vol. 585, no. 7825. — Pp. 357–362. https://doi.org/10.1038/s41586-020-2649-2.

[26] *Caswell, Thomas A.* matplotlib/matplotlib: REL: v3.5.2. — 2022. — . https://doi.org/10.5281/zenodo.6513224.

[27] Tslearn, A Machine Learning Toolkit for Time Series Data / Romain Tavenard, Johann Faouzi, Gilles Vandewiele et al. // *Journal of Machine Learning Research.* — 2020. — Vol. 21, no. 118. — Pp. 1–6. http://jmlr.org/papers/v21/20-091.html.

[28] *Law, Sean M.* STUMPY: A Powerful and Scalable Python Library for Time Series Data Mining / Sean M. Law // *The Journal of Open Source Software.* — 2019. — Vol. 4, no. 39. — P. 1504.

[29] *Mikhaltsov, Danila.* Detecting anomalous device loads during exploratory testing of mobile applications / Danila Mikhaltsov, Konstantin Sorokin // 2022 Ivannikov Ispras Open Conference (ISPRAS). — 2022. — Pp. 43–49.