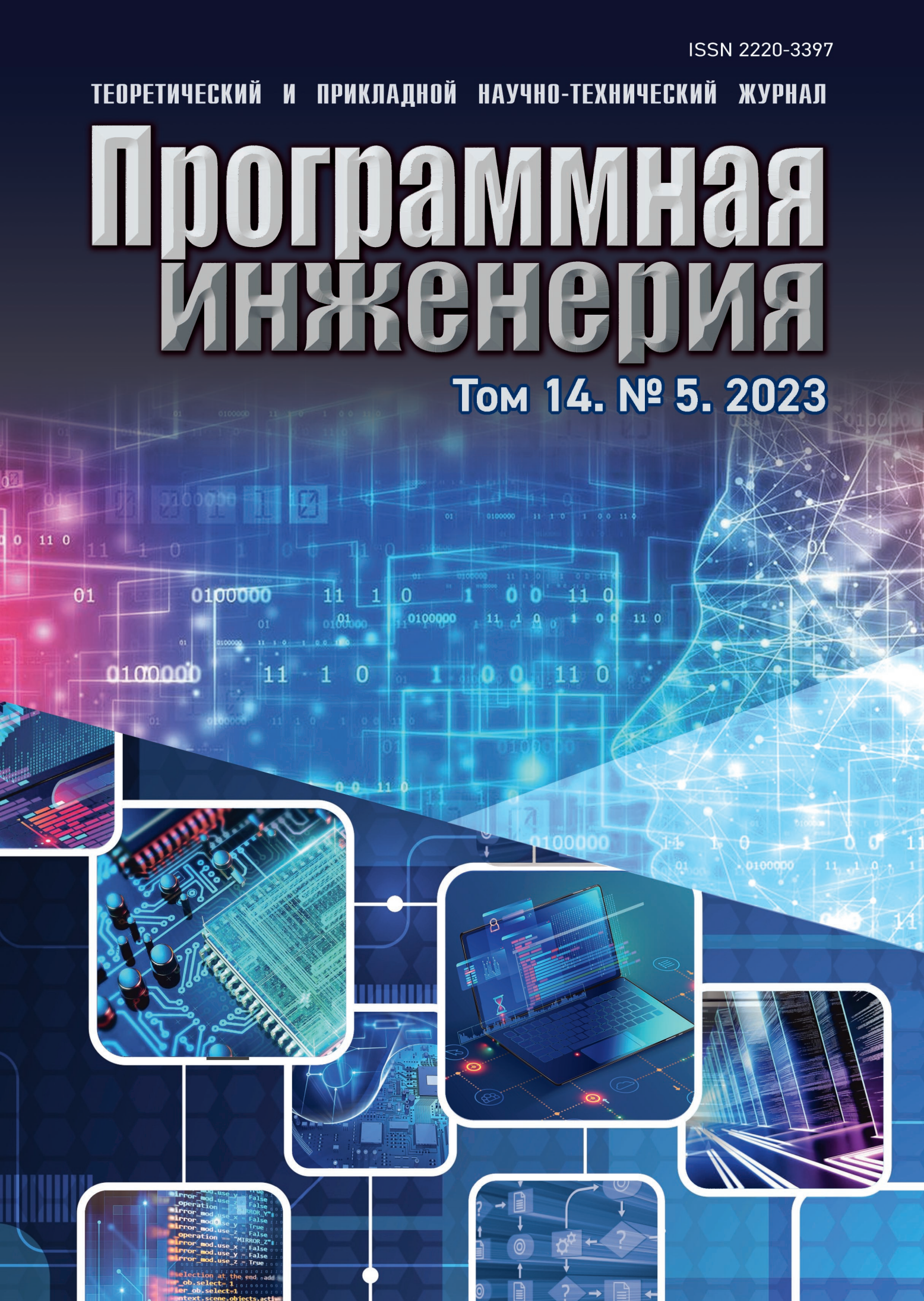
ISSN 2220-3397

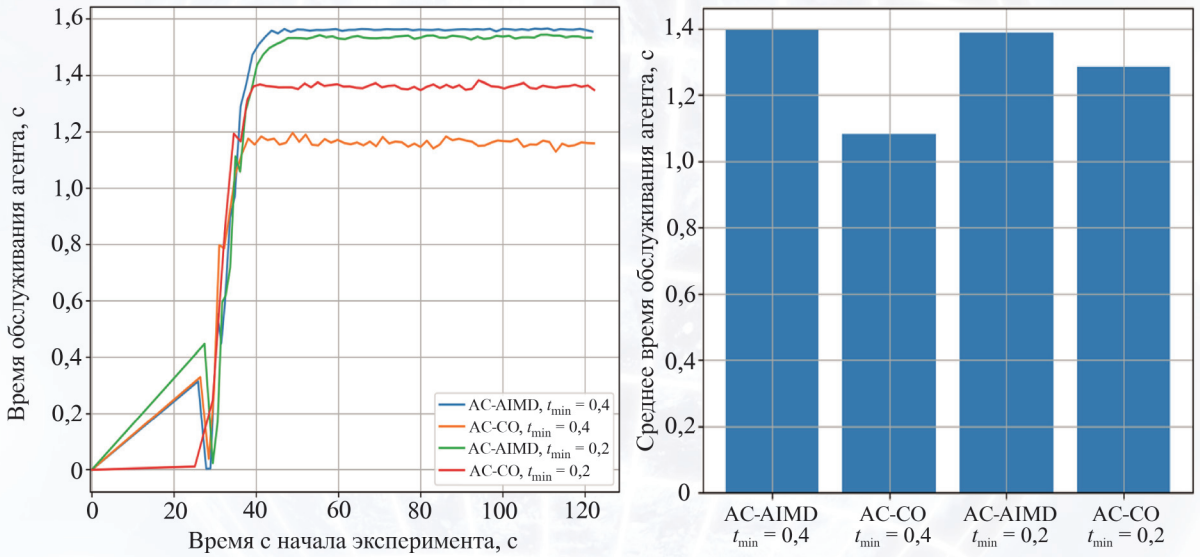
ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Программная инженерия

Том 14. № 5. 2023

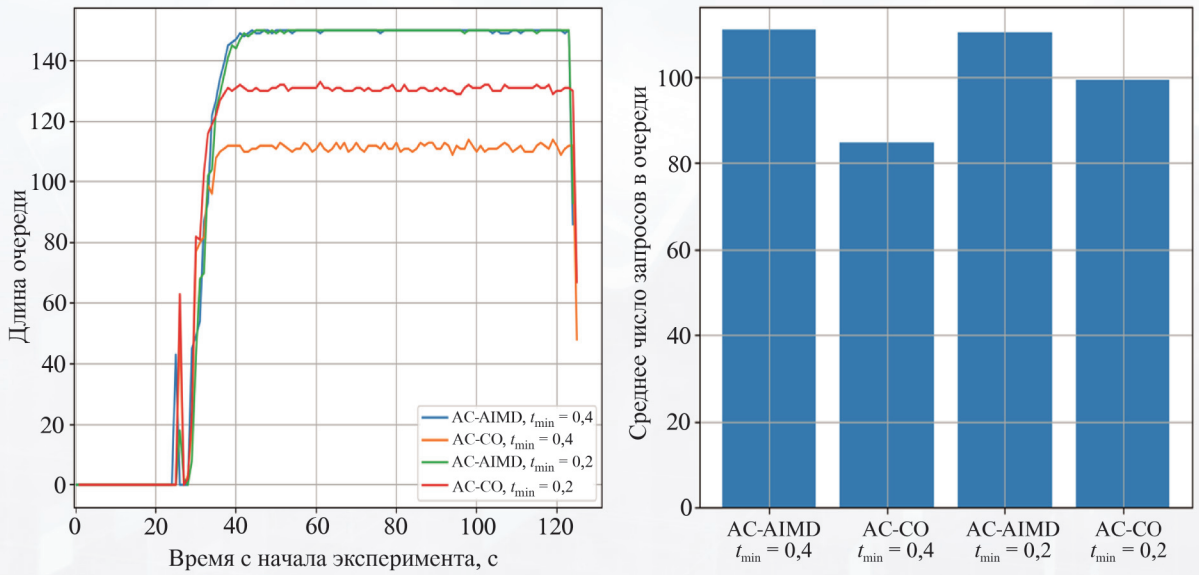


Рисунки к статье **Д. Ж. Корзуна, О. Ю. Богоявленской, К. А. Кулакова**
«ПРИМЕНЕНИЕ АЛГОРИТМА СЛУЧАЙНОЙ ОТСРОЧКИ ПРИ АКТИВНОМ УПРАВЛЕНИИ ОБМЕНОМ ИНФОРМАЦИИ В ИНТЕРНЕТ-СРЕДЕ»



а) *б)*

Рис. 2. Обработка запросов на уведомление при активном управлении:
а – поведение времени обслуживания; *б* – среднее время обслуживания за весь период



а) *б)*

Рис. 3. Очередь СИБ для обслуживания запросов на уведомление при активном управлении:
а – поведение длины очереди; *б* – средняя длина очереди за весь период

ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г. DOI 10.17587/issn.2220-3397 ISSN 2220-3397

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э. Баумана

СОДЕРЖАНИЕ

Корзун Д. Ж., Богоявленская О. Ю., Кулаков К. А. Применение алгоритма случайной отсрочки при активном управлении обменом информации в интернет-среде 207

Пащенко Д. С. Полностью удаленная разработка программного обеспечения как новый стандарт IT-отрасли: европейское исследование 2022—2023 гг. 217

Советов П. Н. Алгоритмы улучшения автоматически синтезированного набора команд расширяемого процессора 225

Марков В. Н. Аргументы-перечислители как языковый инструмент определения циклов 232

Назаров Д. И., Сигалов Д. А., Гамаюнов Д. Ю. Поиск информации о принимаемых сервером запросах в закомментированном клиентском коде веб-приложений 245

Бабич Н. А. Программная платформа для чтения, обработки и анализа данных ЭЭГ 254

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в подписных агентствах (индекс по Объединенному каталогу "Пресса России" — **22765**) или непосредственно в редакции (для юридических лиц).
Тел.: (499) 270-16-52.
Http://novtex.ru/prin/rus E-mail: prin@novtex.ru
Журнал включен в Российский индекс научного цитирования (РИНЦ) и Russian Science Citation Index (RSCI).
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

Vol. 14
N 5
2023

Published since September 2010 **DOI 10.17587/issn.2220-3397** **ISSN 2220-3397**

CONTENTS

Korzun D. G., Bogoiavlenskaia O. Yu., Kulakov K. A. Applying the Random Backoff Algorithm for Active Information Exchange Control in Smart Internet Environment 297

Pashchenko D. S. Fully Remote Software Development as a New Standard in the IT Industry: European Study 2022—2023 217

Sovietov P. N. Algorithms for Improving the Automatically Synthesized Instruction Set of an Extensible Processor 225

Markov V. N. Enumerator-Arguments as a Language tool for Defining Loops 232

Nazarov D. I., Sigalov D. A., Gamayunov D. Yu. Mining Server HTTP Endpoints from Commented-Out Client-Side Code of Web Applications 245

Babbysh N. A. Software Platform for Reading, Processing and Analyzing EEG Data 254

Information about the jornal is available online at:
http://novtex.ru/prin/eng e-mail:prin@novtex.ru

УДК 004.6, 004.7, 004.8
DOI: 10.17587/prin.14.207-216

Д. Ж. Корзун, канд. физ.-мат. наук, доц., вед. науч. сотр., dkorzun@cs.karelia.ru,
О. Ю. Богоявленская, канд. техн. наук, доц., olbgvl@cs.petrsu.ru,
К. А. Кулаков, канд. физ.-мат. наук, доц., kulakov@cs.petrsu.ru,
Петрозаводский государственный университет

Применение алгоритма случайной отсрочки при активном управлении обменом информации в интернет-среде

Поступила в редакцию 10.03.2023
Принята к публикации 04.04.2023

Обмен информацией как способ взаимодействия участников интернет-среды является основополагающей функцией распределенной системы в рамках вычислительных парадигм интернета вещей, киберфизических систем, систем окружающего интеллекта и др. Участники интернет-среды представлены программными агентами, работающими в том числе и на слабопроизводительных периферийных вычислительных устройствах. Взаимодействие выполняется через обмен информацией, накапливаемой участниками в общем информационном хранилище. Такой вариант взаимодействия реализует основу для «интеллектуальности» данной среды. Организация доступа агентов к информации и взаимодействие агентов через обмен информацией используют семантического информационного брокера, управляющего информационными потоками между участниками. При увеличении числа, разнообразия и активности участников возрастает нагрузка на брокера и сеть передачи данных. В данной работе предложен метод снижения нагрузки за счет делегирования части управления обменом информации на самих участников. Модифицирован механизм реализации операции подписки на информационные изменения в общем информационном хранилище. Вместо пассивного варианта, при котором брокер должен уведомить всех агентов, подписавшихся на изменяемую информацию, используется активный вариант, при котором агенты сами могут проверять наличие информационных изменений. Предложен новый метод двухкомпонентного управления на основе алгоритма случайной отсрочки в комбинации с адаптивной стратегией выбора времени отправки запроса на наличие информационных изменений от агента к брокеру. Возможности двухкомпонентного управления исследованы экспериментально на основе имитационной модели интернет-среды.

Ключевые слова: *интернет-среда, обмен информацией, программный агент, семантический брокер, подписка, адаптивная стратегия, случайная отсрочка*

Для цитирования:

Корзун Д. Ж., Богоявленская О. Ю., Кулаков К. А. Применение алгоритма случайной отсрочки при активном управлении обменом информации в интернет-среде // Программная инженерия. 2023. Том 14, № 5. С. 207—216. DOI: 10.17587/prin.14.207-216.

Введение

Рассмотрим интернет-среду, состоящую из множества вычислительных устройств (ВУ), как правило, слабопроизводительных, ориентированных на периферийные вычисления (*edge computing*) [1]. Каждое ВУ выполняет множество функций по сбору, обработке и передаче данных, которые реализует программный агент (ПА), запускаемый на ВУ. За счет развития ПА становится возможным разработка интеллектуальных интернет-сред (ИИС), т. е. так называемого окружающего интеллекта (*Ambient Intelligence* — AmI) [2].

Одной из перспективных областей применения таких ИИС выступают прикладные системы, ре-

ализующие постоянный мониторинг на основе мобильных ВУ и методов сенсорики [3, 4]. В частности, системы мониторинга производственного оборудования и транспорта требуют организации обработки данных из множества источников и интенсивного обмена информацией между ПА. Требуется эффективная организация доступа к общему информационному ресурсу, основанная на таком виде сенсорики, как распознавание на уровне каждого ПА текущей ситуации с загруженностью ИИС при выполнении сбора, обработки и передачи данных.

Эффективность организации доступа означает адаптацию к текущим условиям работы. Возможное решение — применение адаптивной стратегии активного управления [5], позволяющей каждому ПА индивидуально подбирать параметры обмена информацией, настраивая их под текущую ситуацию. Активное управление обменом информацией предполагает, что часть вычислительной нагрузки делегируется с централизованных серверных элементов на ПА. Такой подход представляется целесообразным для периферийных ИИС с множеством слабопроизволительных ВУ, находящихся рядом с местом сбора данных [6].

В то же время параллельный доступ к общей информации с множества ПА может приводить к повышенной нагрузке на ВУ всей ИИС. Возможное решение — алгоритм случайной отсрочки [7, 8], позволяющий каждому ПА распознавать перегрузку ИИС и снижать свою активность для уменьшения будущих потерь при обмене информацией. Отметим, что адаптация и отсрочка являются свойствами «интеллектуальности» ИИС в части управления работы с данными, а не в части технологий искусственного интеллекта для анализа таких данных [2, 3].

В данной работе исследован метод двухкомпонентного управления обменом информацией в периферийной ИИС, предложенный ранее в работе [7]. Первая компонента определяется адаптивной стратегией выбора времени отправки запросов, что позволяет каждому ПА индивидуально выходить на сбалансированный интервал задержки между запросами [5]. Вторая компонента определяется алгоритмом случайной отсрочки [8], что позволяет каждому ПА распознавать перегрузку и снижать свою активность [7]. Проведенное экспериментальное исследование показывает, что предлагаемый метод дает возможность контролировать вычислительные ресурсы ИИС: время отклика на запрос, длину очереди запросов, число передаваемых уведомлений, уровень потерь уведомлений. Полученные результаты показывают, как на уровне ПА реализовать (в виде программы для работы на слабопроизводительном ВУ) такие свойства интеллектуальности управления обменом данных, как адаптация к текущей нагрузке на ВУ в ИИС и отсрочка активности при распознавании перегрузки.

Методы организации доступа к общему ресурсу

Выделим два известных класса методов доступа к общему ресурсу, которые широко применяются для управления обменом информацией через общее информационное хранилище, в частности, алгоритмы AIMD и алгоритмы отсрочки.

1. Алгоритмы AIMD (*Additive Increment, Multiplicative Decrement*). При организации доступа необходимо выбирать время доступа (время отправки запроса). Алгоритм позволяет адаптировать время следующим образом. Если доступ происходит без потерь, то можно реже обращаться, но это увеличивает время аддитивно (консервативная реакция). Если возникают потери, то время доступа нужно сокращать мультипликативно (радикальная реакция).

Ярким примером является алгоритм AIMD протокола TCP [9]. Задержка перед отправкой запроса увеличивается аддитивно при условии отсутствия потерь обновлений информации и уменьшается мультипликативно при наличии таких потерь, причем параметр убывания задержки является функцией числа потерь обновлений, произошедших с момента последнего успешного обновления. Обобщенный алгоритм AIMD для использования в ИИС в целях управления обменом информацией на основе учета потерь и истории представлен в работе [5].

2. Алгоритмы отсрочки (*backoff*). Алгоритмы (случайной) отсрочки широко используются в распределенных системах для координации доступа к разделяемым ресурсам (например, для десинхронизации конкурентного доступа). Различные версии алгоритма реализованы во многих приложениях и системах [8—11], например, в сети Ethernet, в беспроводных сетях, при управлении транзакциями оперативной памяти, в серверах электронной почты, в многопроцессорных системах, для таймаута протокола TCP и др. Такие алгоритмы можно условно разделить на две группы: алгоритмы с обнаружением коллизий и алгоритмы предотвращением коллизий.

Алгоритмы первой группы предполагают, что управляемая система имеет механизм обнаружения коллизий доступа. Если коллизия произошла, то соревнующиеся за доступ элементы системы

генерируют случайную отсрочку по времени, ожидают и затем повторяют попытку захвата ресурса. Если коллизия повторяется, то первоначальное значение отсрочки увеличивается по некоторому мультипликативному правилу.

Алгоритмы предотвращения коллизий предполагают, что элементы системы используют случайную отсрочку перед первой попыткой захвата общего ресурса. Если коллизия тем не менее произошла, то новое значение задержки выбирается из более широкого интервала. Такой подход используется в системах, которые не обладают надежным методом идентификации коллизий, например, в беспроводных сетях семейства стандартов IEEE 802.11x [10].

В данной работе исследован алгоритм случайной отсрочки, предложенный в работе [7]. Требуется оценить эффективность дополнительных способов управления доступом в случае, когда алгоритм AIMD приводит к перегрузке ИИС.

Интеллектуальные интернет-среды

Участником интернет-среды выступает ПА, работающий на некотором ВУ в составе ИИС. Множеством ПА выполняется совместное построение цифровых сервисов на основе информации, накапливаемом в общем информационном пространстве [2]. Каждый ПА выполняет одну или более из следующих функций по получению, обработке и передаче данных [3, 4]:

- получение данных (ВУ-датчик);
- обработка данных (ВУ-сервер);
- передача данных (ВУ-маршрутизатор);
- доставка данных пользователю или реализация реакции на содержащуюся в данных информацию (ВУ-сервис, ВУ-актуатор).

Программная часть ВУ-датчика реализует выдачу данных (измерений) в цифровом формате. Программная часть ВУ-сервера реализует извлечение полезной информации, содержащейся в получаемых данных (например, распознавание информационных событий). Программная часть ВУ-маршрутизатора реализует сетевую передачу данных требуемым участникам интернет-среды. Программная часть ВУ-сервиса реализует действия по отношению к пользователю (например, доставку и визуализацию информации).

Если ПА интегрирует функции получения, доставки и обработки данных, то участника называют умным IoT-объектом. Такой ПА способен понимать статус работы собственных подсистем и в соответствии с этим выполнять действия, в том числе взаимодействуя с другими участниками. Интеллектуальность ПА проявляется в построении ответов на следующие вопросы [2, 12]:

- откуда получать данные (собственные датчики, данные от других участников) и когда запрашивать (регулярно, по уведомлению и пр.);
- как обрабатывать данные (алгоритмы — анализ сигнала, временные ряды, модели машинного обучения и др.);
- куда и какие результаты обработки передавать (взаимодействие с другими участниками).

Такие интернет-среды будем назвать интеллектуальными (ИИС), следуя концепции окружающего интеллекта — AmI. Свойства и интеллектуальности проявляются как в индивидуальном интеллекте каждого участника (умный IoT-объект), так и в коллективном интеллекте системы (взаимодействие участников при построении сервисов). Реализация такого интеллекта выполняется на основе алгоритмов анализа данных на уровне ПА. Каждый ПА является «представителем» тех данных, которые он получает от собственных датчиков или от ассоциированных с ПА источников данных.

Подход периферийных (краевых) вычислений позволяет выполнять анализ данных на уровне ПА (на локальном ВУ), реализуя совместную обработку данных непосредственно рядом с местом сбора этих данных [6, 13]. В частности, это позволяет реализовать на базе ИИС периферийную аналитику данных без затрат на сетевую передачу данных в глобальные центры обработки. Функция получения данных (ВУ-датчик) дополняется функциями обработки (ВУ-сервер), передачи (ВУ-маршрутизатор) и доставки (ВУ-сервис) данных.

Основополагающую роль в организации взаимодействия по совместной обработке данных и построения сервисов играет обмен информацией между множеством ПА. Известная модель публикации/подписки реализует непрямое (косвенное) взаимодействие множества ПА на основе семантического информационного брокера (СИБ) [13]. Если ПА является поставщиком данных, то он через СИБ публикует информацию в общем информационном хранилище. Если ПА является потребителем данных, то он подписывается в СИБ на требуемую информацию. При изменении информации в информационном хранилище к потребителям данных по подписке от СИБ поступают уведомления об информационных изменениях.

Модель публикации/подписки обеспечивает построение в ИИС персонализированных, контекстно-зависимых, проактивных сервисов информационной поддержки и управления IoT-объектами (локальными, дистанционными или виртуальными) [2, 14]. Построение сервиса является информа-

ционно-управляемым. Каждый ПА, участвующий в построении сервиса, реагирует на изменения в общем информационном хранилище, выполняет локальную обработку данных и обновляет информацию в общем хранилище. Возникает цепочка (граф) взаимодействия участников. Шаги взаимодействия инициируются происходящими информационными изменениями и соответствуют извлечению информации из данных от множества источников и итеративному обновлению в информационном хранилище.

Для такого построения сервиса необходимо решить задачу по организации взаимодействия на основе управления обменом информацией между множеством ПА. Требуется отслеживать информационные изменения, вызываемые работой множества различных ПА. На эффективность взаимодействия влияет интенсивность изменений, вносимых участвующими ПА. Возможны потери уведомлений об информационных изменениях, что обуславливается ограниченными вычислительными и сетевыми ресурсами. Этот недостаток и определяет основную проблему исследования данной статьи.

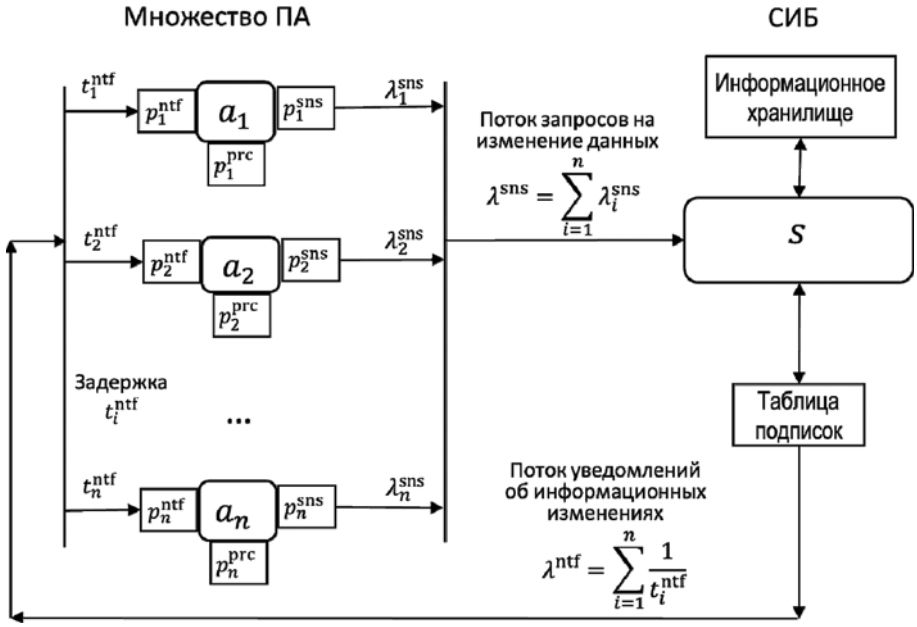


Рис. 1. Концептуальная модель взаимодействия множества ПА в ИИС

Взаимодействие в интеллектуальной интернет-среде

Определим ПА формально как тройку вычислительных процессов

$$a = \left(p^{\text{sns}},\ p^{\text{prc}},\ p^{\text{ntf}}\right),$$

где p^{sns} соответствует потоку данных, передаваемых от агента в общее информационное хранилище; p^{prc} — обработке данных на ВУ агента; p^{ntf} — потоку уведомлений от СИБ к ПА о происходящих информационных изменениях в общем хранилище. Пусть n — это число участвующих агентов. Тогда ИИС определим как $\left(s, \{a_i\}_{i=1}^{n}\right)$, где СИБ s организует взаимодействие всех ПА a_i для $i = 1, 2, ..., n$.

Концептуальная модель взаимодействия в ИИС представлена на рис. 1. Процессы p_i^{sns} приводят к потоку запросов от ПА на изменение данных в информационном хранилище через СИБ с суммарной интенсивностью

$$\lambda^{\text{sns}} = \sum_{i=1}^{n} \lambda_i^{\text{sns}},$$

где λ_i^{sns} — интенсивность запросов от a_i. Запросы на изменение данных приводят к потоку данных в обратном направлении по подписке — уведомления от СИБ к множеству ПА с суммарной интенсивностью

$$\lambda^{\text{ntf}} = \sum_{i=1}^{n} \frac{1}{t_i^{\text{ntf}}},$$

где интенсивность уведомлений $\lambda^{\text{ntf}} = 1/t_i^{\text{ntf}}$ для a_i определим через среднюю длину интервала t_i^{ntf} между последовательными уведомлениями.

На стороне СИБ формируются две очереди: запросы на изменение и запросы на уведомления. Основной способ для доставки уведомлений — это пассивная подписка, когда каждый ПА ждет уведомлений от СИБ. В этом случае характеристики очереди уведомлений зависят от структуры информации (таблица подписок), на которую подписаны ПА; от активности изменения этой информации множеством ПА, а также от вычислительных возможностей СИБ.

Активное управление обменом информации в интеллектуальной интернет-среде

В случае пассивной подписки СИБ является центральным сервером, который берет на себя всю нагрузку, связанную со своевременной доставкой уведомлений. Такой вариант оправдан, когда СИБ размещен на высокопроизводительном ВУ (локальном для интернет-среды или на удаленной

серверной системе). В то же время многие задачи по обработке данных требуется решать в условиях слабопроизводительных периферийных ВУ. В частности, такие задачи возникают в системах мониторинга производственного оборудования и в мобильных робототехнических системах, где требуется обрабатывать данные из множества источников в режиме реального времени, реализуя сервисы оперативного информационного сопровождения объекта мониторинга.

Необходим переход к децентрализованному варианту, когда подписка становится активной (на стороне ПА). По аналогии с одноранговыми распределенными системами (*peer-to-peer* — P2P) часть серверной нагрузки СИБ может делегировать на другие ВУ в ИИС [15]. При управлении обменом информации нужно обеспечить баланс между вычислениями на СИБ и на множестве ПА. Для решения этой задачи управления ранее была предложена индивидуальная адаптивная стратегия для активной подписки [5].

Адаптивная стратегия ориентирована на индивидуальное управление каждым ПА своего интервала $t = t^{\text{ntf}}$ между последовательными уведомлениями. Цель управления — адаптация к потерям, т. е. когда на интервале произошли информационные изменения, уведомления о которых не дошли до ПА. Стратегия реализует активную подписку, так как ПА сам отправляет запросы к СИБ на проверку информационных изменений. В общем случае возможна комбинация пассивной и активной подписок.

Рассмотрим вариант активной подписки. Пусть заданный ПА проверяет информационные изменения на основе последовательности запросов с задержками t_j между запросами j и $j + 1$ для дискретного времени $j = 0, 1, \ldots$ и заданной начальной задержки t_0. За время t_j обнаружено $k_j \geqslant 0$ потерь, т. е. число информационных изменений, уведомления по которым не доставлены в ПА. Тогда адаптивная стратегия использует следующее правило вычисления задержки ПА до следующего запроса:

$$t_{j+1} = t_{j+1}^{\text{adp}} = \begin{cases} t_j + \delta, \text{ если } k_j = 0 \\ \dfrac{1 + \alpha k_j}{k_j + 1} t_j, \text{ если } k_j \geqslant 1, \end{cases} \quad (1)$$

где $0 \leqslant \alpha \leqslant 1$ — коэффициент скользящего среднего в истории потерь; δ — аддитивный коэффициент роста задержки.

Эта стратегия может быть сведена к алгоритму AIMD за счет $\alpha = 0$, и система рассматривает не абсолютное число потерь уведомлений, а бинарное событие потери как таковое. В этом случае стратегия принимает вид:

$$t_{j+1} = t_{j+1}^{\text{AIMD}} = \begin{cases} t_j + \delta, \text{ если } k_j = 0 \\ \dfrac{1}{\mu} t_j, \text{ если } k_j \geqslant 1 \end{cases}, \quad (2)$$

где $\mu \geqslant 1$ — коэффициент мультипликативного убывания. Часто используемый вариант — бинарный, когда $\mu = 2$.

Таким образом, ПА «осторожно» (аддитивное правило) увеличивает время задержки в случае отсутствия потерь. При обнаружении потерь ПА «радикально» (мультипликативное правило) уменьшает задержку следующего запроса.

В то же время необходимо отслеживать экстремально низкие значения задержки и прекращать мультипликативное убывание. Малая задержка приводит к частым запросам к СИБ, что увеличивает загрузку сети и СИБ, а значит, и число потерь. В простейшем варианте можно считать $t_{j+1} = t_j$.

Ранее в работе [7] было предложено использовать случайную отсрочку для компенсации малых задержек. В настоящей статье предложен общий метод активного управления обменом информации на основе объединения адаптивной стратегии с алгоритмом случайной отсрочки.

Активное двухкомпонентное управление обменом информации

Чрезмерное уменьшение задержки в адаптивной стратегии повышает нагрузку на СИБ. В данной работе предлагается метод активного управления обменом информацией в ИИС, в котором сбалансированы две компоненты управления отправкой запросов на уведомления: адаптивная задержка и случайная отсрочка.

Повышение нагрузки на СИБ вызывает потери. В адаптивной стратегии (1), (2) обнаружение потерь приводит к уменьшению задержки, а значит, нагрузка на СИБ снова повышается. Будем считать коллизией неприемлемое снижение качества услуг от СИБ, обнаружив которое каждый ПА индивидуально применит случайную отсрочку [7]. Такой способ нацелен на снижение числа запросов на уведомления, когда ПА ожидает последующее повышение качества услуг со стороны СИБ. Общее правило двухкомпонентного управления имеет следующий вид:

$$t_{j+1} = t_{j+1}^{\text{ntf}} = \begin{cases} t_j^{\text{adp}} + t_{j+1}^{\text{bck}}, \text{ если коллизия} \\ t_j^{\text{adp}}, \text{ иначе} \end{cases}. \quad (3)$$

Здесь вместо задержки t_j^{adp} из аддитивной стратегии (1) можно использовать t_j^{AIMD} из выражения (2). Таким образом, при слишком малых значения t_j^{adp}

общая задержка увеличивается за счет случайной отсрочки.

Вычисление t_{j+1}^{bck} является итеративным с начальным значением $t_{j+1,0}^{\text{bck}}$, которое определяется минимальным ожидаемым временем отклика $t_{\min}^{\text{rsp}}$ (время обработки запроса на уведомления на стороне СИБ, возможно, включая время сетевой передачи). Оценку времени отклика может выполнять СИБ и передавать ПА среднее время обработки запроса $t_{\text{avg}}^{\text{rsp}}$. Оценку $t_{\text{avg}}^{\text{rsp}}$ каждый ПА может выполнять индивидуально, измеряя время от отправки запроса до получения результата. По среднему времени обработки запроса можно выбрать значение $t_{j+1,0}^{\text{bck}} \approx t_{\min}^{\text{rsp}} < t_{\text{avg}}^{\text{rsp}}$ (например, 75 % от среднего или правило трех сигм).

Если на следующих шагах вычисления (3) коллизия продолжается, то ПА увеличивает отсрочку экспоненциально:

$$t_{j+1,l}^{\text{bck}} = \min\left(\vartheta t_{j+1,l-1}^{\text{bck}},\ t_{\max}^{\text{bck}}\right) \text{ для } l = 1, 2, \quad (4)$$

где $\vartheta > 1$ — коэффициент мультипликативного увеличения отсрочки (например, бинарный вариант $\vartheta = 2$, по аналогии с (2)). Верхняя граница возможной отсрочки $t_{\max}^{\text{bck}}$ определяется на основе среднего времени обработки запроса $t_{\text{avg}}^{\text{rsp}}$, исходя из условия $t_{\max}^{\text{bck}} \leqslant n t_{\text{avg}}^{\text{rsp}}$, где n — общее число ПА в ИИС. Рост времени отсрочки прекращается не позднее, чем через n итераций.

Внесение элемента случайности в отсрочку достигается за счет выбора начального значения $t_{j+1,0}^{\text{bck}} = t_{\min}^{\text{rsp}} < t_{\text{avg}}^{\text{rsp}}$. Среднее время обработки запроса ПА:

$$t_{\text{avg},\,j+1}^{\text{rsp}} = \beta t_{\text{avg},\,j}^{\text{rsp}} + (1-\beta) t_j^{\text{rsp}}, \quad (5)$$

где $t_{\text{avg},\,j}^{\text{rsp}}$ — среднее значение на предыдущем шаге; t_j^{rsp} — текущее измерение времени отклика, а $0 \leqslant \beta \leqslant 1$ — коэффициент скользящего среднего (по аналогии с (1)).

Коллизию определим как ситуацию, когда задержка t_{j+1}, вычисляемая по формуле (1) или (2), становится меньше минимального ожидаемого времени отклика

$$\theta = t_{\min}^{\text{rsp}} - t_{j+1}^{\text{adp}}.$$

Если $\theta > 0$, то обнаружена коллизия. Используем нормальное распределение с математическим ожиданием θ и среднеквадратическим отклонением σ для генерации начального значения случайной отсрочки $t_{j+1,0}^{\text{bck}} \approx t_{\min}^{\text{rsp}}$. Значение σ характеризует вариабельность отсрочки. Рекомендуется использовать $\sigma \leqslant \theta/3$, поскольку в этом случае вероятность $0 < t_{j+1,0}^{\text{bck}} \leqslant 2\theta$ более 0,99.

Таким образом, вторая компонента управления — случайная отсрочка — активируется, когда первая компонента (адаптивная стратегия) приводит к недопустимо малым значениям задержки. В соответствии с выражением (3) происходит компенсация, что позволяет снизить нагрузку на СИБ. Такое увеличение задержки, естественно, приводит к увеличению потерь. В то же время процесс p^{ntf} по обмену актуальной информацией сохраняется, а не прекращается вследствие возникшей перегрузки СИБ, как это было показано в работе [7].

Использование параметров адаптивной стратегии (1), (2) и двухкомпонентной стратегии (3), (4) позволяет настраивать управление на заданный баланс между числом потерь, нагрузкой на СИБ и на сеть передачи данных. Далее рассмотрим несколько имитационных экспериментов, оценивающих эффект от использования случайной отсрочки (3) в сравнении с использованием только адаптивной стратегией (2) в ИИС с множеством ПА на слабопроизводительных ВУ.

Экспериментальное исследование возможностей двухкомпонентного управления

В эксперименте была использована следующая имитационная модель: работа СИБ в условиях вычислительных ограничений, что имитируется случайными задержками при обработке запросов от множества ПА.

Множество ПА состоит из n участвующих в ИИС агентов, каждый генерирует сенсорные данные для публикации в информационное хранилище с интенсивностью $\lambda^{\text{sns}} = \lambda_i^{\text{sns}}$ (интенсивность потока данных в процесс p^{sns}). Суммарная интенсивность публикации данных в СИБ определяется как $\lambda_{\text{tot}}^{\text{sns}} = n\lambda^{\text{sns}}$. Каждый ПА a_i публикует порцию данных d_i для $i = 1, 2, ..., n$. Каждая порция d_i включает в себя числовое значение и время публикации (таймстамп). Время между последовательными запросами на публикацию соответствует нормальному распределению с математическим ожиданием $1/\lambda^{\text{sns}}$ и среднеквадратическим отклонением как 10 % от математического ожидания.

Каждый ПА a_i подписывается на данные $\pi_i = \{d_e\}_{l=1}^{m}$, где $l \neq i$ (нельзя a_i подписать на самого себя). Эти m порций данных выбираются равновероятно. На СИБ формируется соответствующая таблица подписок. Сравниваются две стратегии для активного управления (AC — *Active Control*):

а) AC-AIMD: адаптивная стратегия с алгоритмом AIMD по правилу (2) для $\mu = 2$;

б) AC-CO: двухкомпонентное управление по правилам (3), (4) для $\vartheta = 2$ и $\beta = 0{,}5$ на основе AC-AIMD.

В двухкомпонентном управлении AC-CO параметры θ и σ подбираются так, чтобы

$$T = \mu\sqrt{\frac{2(\mu-1)}{\lambda\delta(\mu+1)}} \approx \frac{1}{\lambda_l},$$

где T — аналитическая оценка средней длины задержки стратегии (2), см. обоснование в [7]. Значение $t_{i,l}^{\text{sub}} = 1/\lambda_l$ соответствует минимальному среднему интервалу между последовательными изменениями в данных π_i.

В соответствии с этими стратегиями каждый ПА a_i реализует управление обменом информации по активной подписке на π_i. В экспериментах использовалось $n = 100$ ПА.

Измерение времени отклика t_j^{rsp} на запрос для дискретной эволюции $j = 0,1, \ldots$ определяется как время обработки запроса от ПА на уведомления на стороне СИБ. Измерение выполняется на стороне СИБ. На рис. 2 (см. вторую сторону обложки) показаны измерения t_j^{rsp} в секундах для стратегий AC-AIMD и AC-CO для двух значений минимального ожидаемого времени отклика $t_{\min}^{\text{rsp}} = 0{,}2$, 0,4 с. После краткого периода адаптации к условиям работы ИИС наблюдается выход на стационарный режим. Этот режим сохраняется, так как условия работы ИИС зафиксированы.

В целом увеличение времени задержки за счет случайной отсрочки приводит к уменьшению времени отклика. В стратегии со случайной отсрочкой минимальное ожидаемое время отклика $t_{\min}^{\text{rsp}}$ выступает параметром, позволяющим управлять временем отклика. В «чистой» стратегии AC-AIMD этот параметр оказывает незначительное воздействие (за счет выбора начальной задержки $t_0^{\text{AIMD}} = t_{\min}$).

Другой метрикой эффективности работы является q_j^{ntf} — длина очереди СИБ для запросов на уведомления, поступающих от множества ПА в ходе активного управления обменом информации (эволюция в дискретном времени $j = 0, 1, \ldots$). На рис. 3 (см. вторую сторону обложки) показаны измерения длины очереди для стратегий AC-AIMD и AC-CO для двух значений минимального ожидаемого времени отклика $t_{\min}^{\text{rsp}} = 0{,}2$, 0,4 с. После краткого периода адаптации к условиям работы ИИС наблюдается выход на стационарный режим. В конце очередь опустошается, так как ПА перестают отправлять запросы.

Аналогично предыдущей метрике, увеличение времени задержки за счет случайной отсрочки приводит к уменьшению длины очереди. В стратегии со случайной отсрочкой минимальное ожидаемое время отклика $t_{\min}^{\text{rsp}}$ выступает параметром, позволяющим управлять длиной очереди. В «чистой» стратегии AC-AIMD этот параметр значимого влияния не имеет.

Рассмотрим метрику v_j^{ntf}, определяемую как число уведомлений (о прошедших информационных изменениях), получаемых ПА за один запрос (эволюция в дискретном времени $j = 0,1, \ldots$). На рис. 4 (см. третью сторону обложки) показаны измерения на СИБ значений v_j^{ntf} для стратегий AC-AIMD и AC-CO для двух значений минимального ожидаемого времени отклика $t_{\min}^{\text{rsp}} = 0{,}2,\ 0{,}4$ с. После краткого периода адаптации к условиям работы ИИС наблюдается выход на стационарный режим.

Увеличение времени задержки за счет случайной отсрочки и увеличение минимального ожидаемого времени отклика $t_{\min}^{\text{rsp}}$ приводит к росту числа уведомлений. Это позволяет управлять производительностью системы, в том числе снижать уровень энергопотребления на ВС агентов.

Рассмотрим число потерь k_j, где потеря соответствует произошедшему информационному изменению, уведомления о котором не было доставлено некоторому ПА, см. также правила (1), (2) адаптивной стратегии. На рис. 5 (см. третью сторону обложки) показаны измерения на СИБ значений k_j для стратегий AC-AIMD и AC-CO для двух значений минимального ожидаемого времени отклика $t_{\min}^{\text{rsp}} = 0{,}2$, 0,4 с. После краткого периода адаптации к условиям работы ИИС наблюдается выход на стационарный режим.

Обе стратегии увеличивают время задержки, что приводит к ненулевому уровню потерь. Увеличение времени задержки за счет случайной отсрочки и уменьшение минимального ожидаемого времени отклика $t_{\min}^{\text{rsp}}$ приводят к росту потерь. Отметим, что наблюдаемый рост уровня потерь не оказывает практически значимого влияния на производительность функций доставки и обработки данных на уровне индивидуального ПА.

Выводы по экспериментальным результатам

В целом имитационные эксперименты подтверждают, что предлагаемое двухкомпонентное управление достаточно быстро адаптируется к условиям работы ИИС на интернет-периферии (за счет первой компоненты — адаптивной стратегии). Устанавливается сбалансированный интервал задержки между индивидуальными запросами от ПА. Баланс возникает между используемыми ресурсами (интенсивность запросов) и уровнем потерь (информационных изменений). Добавление второй компоненты позволяет дополнительно управлять вычислительными ресурсами ИИС, расходуемыми на обмен информацией, на основе следующих целевых показателей:

- времени отклика на обработку запроса от ПА;

- длины очереди запросов на обслуживание уведомлений от ПА к СИБ;
- числе передаваемых уведомлений в одном запросе от СИБ к ПА;
- уровне потерь уведомлений о произошедших информационных изменения для ПА.

Каждая из двух компонент имеет набор параметров, с помощью которых также можно влиять на управление обменом информацией. Оценка этого влияния, а также построение алгоритмов для настройки и адаптации значений этих параметров являются предметом дальнейших исследований.

Стратегии являются индивидуальными, т. е. прямое взаимодействие между ПА не предусмотрено. Тем не менее возникает необходимость синхронизации действий ПА по совместному использованию ресурсов СИБ. Повышение активности некоторого ПА может приводить к ухудшению положения других ПА, что в свою очередь ухудшает положение заданного ПА. Таким образом, перспективной задачей для дальнейшего исследования выступает задача синхронизации действий различных ПА, которая в предлагаемом методе двухкомпонентного управления частично решается на основе алгоритма случайной отсрочки.

Заметим, что интересы получателей уведомлений (ПА-подписчиков) находятся в противоречии с целями управления СИБ и сетевой инфраструктуры. Так, получатели заинтересованы в надежной своевременной доставке всех уведомлений без потерь. При этом нагрузка на СИБ и сетевую инфраструктуру не может превышать ее мощности. В противном случае при переполнении очередей маршрутизаторов и СИБ уровень потерь будет расти. Кроме того, фиксируя потери, ПА-подписчики будут увеличивать интенсивность запросов в силу адаптивной стратегии, что приведет к дальнейшему росту нагрузки на СИБ и сетевую инфраструктуру. Результатом такого развития событий могут стать сбой и появление системных ошибок в работе СИБ, а также перегрузка маршрутизаторов и каналов связи. Заметим также, что периферийные ВУ, на которых реализованы ПА-подписчики, как правило, являются автономными устройствами и, следовательно, требуют экономии заряда батареи.

Эксперименты показывают, что несмотря на отсутствие у ПА сведений об уровне загрузки сетевой инфраструктуры, предлагаемое двухкомпонентное управление позволяет установить сбалансированный интервал задержки t^{ntf} между индивидуальными запросами от ПА. Баланс трактуется как устойчивый компромисс между ресурсами элементов системы (интенсивность запросов, заряд батарей периферийных ВУ) и практически приемлемым уровнем потерь информационных изменений. При этом устанавливаемый интервал t^{adp} является стабильным, не подвержен случайным резким колебаниям и позволяет системе адаптироваться к изменениям мощности ресурсов системы, если такие изменения возникают.

Вторая компонента управления t^{bck}, ограничивающая интервал отсрочки снизу, позволяет дополнительно контролировать вычислительные ресурсы ИИС, расходуемые при обмене информацией. Также в ряде экспериментов можно наблюдать, что увеличение интервала отсрочки за счет случайной отсрочки позволяет ПА получать больше уведомлений за один запрос, что снижает расход заряда батареи.

Заключение

В работе исследованы возможности алгоритма случайной отсрочки для активного управления обменом информацией между множеством ПА в ИИС. Предложен новый метод двухкомпонентного управления. Первая компонента определяется известной адаптивной стратегией выбора времени отправки запросов, что позволяет каждому ПА индивидуально выходить на сбалансированный интервал задержки до следующего запроса. Вторая компонента определяется известным алгоритмом случайной отсрочки, что позволяет каждому ПА распознавать перегрузку СИБ и снижать свою активность для уменьшения будущих потерь из-за перегрузки. Проведенное экспериментальное исследование показывает, что предлагаемый метод позволяет контролировать следующие вычислительные ресурсы ИИС: время отклика СИБ на запрос, длину очереди запросов на стороне СИБ, число передаваемых уведомлений, уровень потерь уведомлений. Двухкомпонентное управление позволяет на уровне каждого ПА реализовать (в виде программы для работы на слабопроизводительном ВУ) такие свойства интеллектуальности управления обменом данных, как адаптация ПА к текущей нагрузке и снижение активности ПА при распознавании перегрузки в ИИС.

В качестве перспективной области для применения предложенного метода следует рассматривать ИИС, реализующие мониторинг в режиме реального времени на основе мобильных ВУ. В частности, системы мониторинга производственного оборудования и транспорта, а также системы мониторинга территории с помощью множества

роботов. Для повышения эффективности управления обменом информацией необходимо далее исследовать влияние введенных параметров (как адаптивной стратегии, так и алгоритма случайной отсрочки) на предмет автоматической настройки и адаптации значений этих параметров под текущее состояние нагрузки ПА и СИБ.

Исследование выполнено за счет гранта Российского научного фонда № 22-11-20040 (https://rscf.ru/project/22-11-20040/), проводимого совместно с Республикой Карелия с софинансированием из Фонда венчурных инвестиций Республики Карелия (ФВИ РК).

Список литературы

1. **Aboubakar M., Kellil M., Roux P.** A review of IoT network management: Current status and perspectives // Journal of King Saud University — Computer and Information Sciences. 2022. Vol. 34, Issue 7. P. 4163—4176. DOI: 10.1016/j.jksuci.2021.03.006.

2. **Korzun D., Balandina E., Kashevnik A., Balandin S., Viola F.** Ambient Intelligence Services in IoT Environments: Emerging Research and Opportunities. IGI Global, 2019. 199 p. DOI: 10.4018/978-1-5225-8973-0.

3. **Vodyaho A., Osipov V., Zhukova N., Chernokulsky V.** Data collection technology for ambient intelligence systems in internet of things // Electronics (Switzerland). 2020. Vol. 9, No. 11. P. 1—26. DOI: 10.3390/electronics9111846.

4. **Balicki J., Balicka H., Dryja P.** Big Data from Sensor Network via Internet of Things to Edge Deep Learning for Smart City // Lecture Notes in Computer Science. 2021. Vol. 12883 LNCS. P. 357—368. DOI: 10.1007/978-3-030-84340-3_29.

5. **Bogoiavlenskaia O., Vdovenko A., Korzun D. G., Kashevnik A.** Individual client strategies for active control of information-driven service construction in IoT-enabled smart spaces // International Journal of Distributed Systems and Technologies. 2019. Vol. 10, No. 2. P. 20—36. DOI: 10.4018/IJDST.2019040102.

6. **Bilal K.** Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers // Computer Networks. 2018. Vol. 130. P. 94—120. DOI: 10.1016/j.comnet.2017.10.002.

7. **Bogoiavlenskaia O., Korzun D., Kulakov K**. Random Backoff for Active Control of Information Updates in Smart Spaces // 24th Conference of Open Innovations Association, FRUCT. 2019. P. 47—53. DOI: 10.23919/FRUCT.2019.8711980.

8. **Bender M. A., Fineman J. T., Gilbert S., Young M.** Scaling exponential backoff: Constant throughput, polylogarithmic channel access attempts, and robustness // Journal of ACM. 2018. Vol. 66, No. 1. P. 6:1—6:33. DOI: 10.1145/3276769.

9. **Allman M., Paxson V., Blanton E.** RFC 5681: TCP Congestion Control. URL: http://datatracker.ietf.org/doc/rfc5681/

10. **Kuptsov D., Nechaev B., Lukyanenko A., Gurtov A.** How penalty leads to improvement: A measurement study of wireless backoff in IEEE 802.11 networks // Computer Networks. 2014. Vol. 75. Part A. P. 37—57. DOI: 10.1016/j.comnet.2014.09.008.

11. **Anderton W. C., Chakraborty T., Young M.** Windowed backoff algorithms for WiFi: theory and performance under batched arrivals // Distributed Computing. 2021. Vol. 34. P. 367—393. DOI: 10.1007/s00446-021-00403-9.

12. **Mouromtsev D.** Semantic reference model for individualization of information processes in IoT heterogeneous environment // Electronics (Switzerland). 2021. Vol. 10, No. 20. Article 2523. DOI: 10.3390/electronics10202523.

13. **Марченков С. А.** Автоматизация процессов программирования агентов на основе кодогенерации при построении семантических сервисов интеллектуальных пространств. Часть 1 // Программная инженерия. 2019. Том 10, № 6. С. 257—264. DOI: 10.17587/prin.10.257-264.

14. **Kulakov K.** An approach to efficiency evaluation of services with smart attributes // International Journal of Embedded and Real-Time Communication Systems. 2017. Vol. 8, No. 1. P. 64—83. DOI: 10.4018/IJERTCS.2017010105.

15. **Korzun D.**, **Gurtov A.** Structured peer-to-peer systems: Fundamentals of hierarchical or-ganization, routing, scaling, and security. Springer New York, NY, 2013. 366 p. DOI: 10.1007/978-1-4614-5483-0.

Applying the Random Backoff Algorithm for Active Information Exchange Control in Smart Internet Environment

D. G. Korzun, Adjunct Professor, Leading Research Scientist, dkorzun@cs.karelia.ru,
O. Yu. Bogoiavlenskaia, Associate Professor, olbgvl@cs.petrsu.ru,
K. A. Kulakov, Associate Professor, kulakov@cs.petrsu.ru, Department of Computer Science, Petrozavodsk State University, Petrozavodsk, 185910, Russian Federation

Corresponding author:

Dmitry G. Korzun, Adjunct Professor, Leading Research Scientist, Institute of Mathematics and Information Technology, Petrozavodsk State University, Petrozavodsk, 185910, Russian Federation
E-mail: dkorzun@cs.karelia.ru

Received on March 10, 2023
Accepted on April 04, 2023

The information exchange-as a way of interaction for participants in an Internet environment-is a fundamental function of a distributed system within the computing paradigms of Internet of Things, Cyber-Physical Systems, Ambient Intelligence etc. Participants in an Internet environment are represented by software agents running typically on low-performance edge computing devices. Interaction is implemented through exchange of information collected by the

participants in a shared information store that implements the «intelligence» of this environment. The organization of agents' access to the shared information and of their interaction uses a semantic information broker that manages information flows between participants. With an increase in the number, diversity, and activity of participants, the workload on the broker and the data transmission network increases. In this paper, a novel method is proposed to reduce the workload by delegating part of the control of the information exchange to the participants themselves. The subscription mechanism is modified for information changes in a shared information store. Instead of the passive option, when the broker must notify all agents who have subscribed to the information being changed, the active option is used, when agents themselves can check for information changes. A new two-component control method based on the well-known backoff algorithm in combination with the well-known adaptive strategy for choosing the time of sending the next request from an agent to a broker for the presence of information changes is proposed. The control capabilities are studied experimentally using a simulated Internet environment.

Keywords: *internet environment, information exchange, software agent, semantic broker, subscription, adaptive strategy, backoff*

Acknowledgements: *The research is implemented with financial support by Russian Science Foundation, project no. 22-11-20040 (https://rscf.ru/en/project/22-11-20040/) jointly with Republic of Karelia and funding from Venture Investment Fund of Republic of Karelia (VIF RK).*

For citation:

Korzun D. G., Bogoiavlenskaia O. Yu., Kulakov K. A. Applying the Random Backoff Algorithm for Active Information Exchange Control in Smart Internet Environment, *Programmnaya Ingeneria,* 2023, vol. 14, no. 5, pp. 207—216. DOI: 10.17587/prin.14.207-216.

References

1. **Aboubakar M., Kellil M., Roux P.** A review of IoT network management: Current status and perspectives. *Journal of King Saud University — Computer and Information Sciences*, 2022, vol. 34, issue 7, pp. 4163—4176. DOI. 10.1016/j.jksuci.2021.03.006.

2. **Korzun D., Balandina E., Kashevnik A., Balandin S., Viola F.** *Ambient Intelligence Services in IoT Environments: Emerging Research and Opportunities*, IGI Global, 2019, 199 p. DOI: 10.4018/978-1-5225-8973-0.

3. **VodyahoA., Osipov V., Zhukova N., Chernokulsky V.** Data collection technology for ambient intelligence systems in internet of things, *Electronics (Switzerland)*, 2020, vol. 9, no. 11, pp. 1-26. DOI 10.3390/electronics9111846.

4. **Balicki J.**, **Balicka H., Dryja P.** Big Data from Sensor Network via Internet of Things to Edge Deep Learning for Smart City, *Lecture Notes in Computer Science,* 2021, vol. 12883 LNCS, pp. 357—368. DOI: 10.1007/978-3-030-84340-3_29.

5. **Bogoiavlenskaia O., Vdovenko A., Korzun D. G., Kashevnik A.** Individual client strategies for active control of information-driven service construction in IoT-enabled smart spaces, *International Journal of Distributed Systems and Technologies*, 2019, vol. 10, no. 2, pp. 20—36. DOI: 10.4018/IJDST.2019040102.

6. **Bilal K.** Potentials, trends, and prospects in edge technologies: Fog, cloudlet, mobile edge, and micro data centers, *Computer Networks*, 2018, vol. 130, pp. 94—120. DOI: 10.1016/j.comnet.2017.10.002.

7. **Bogoiavlenskaia O., Korzun D., Kulakov K**. Random Backoff for Active Control of Information Updates in Smart Spaces, *24th Conference of Open Innovations Association, FRUCT,* 2019, pp. 47—53. DOI: 10.23919/FRUCT.2019.8711980.

8. **Bender M. A., Fineman J. T., Gilbert S., Young M.** Scaling exponential backoff: Constant throughput, polylogarithmic channel access attempts, and robustness, *Journal of ACM*, 2018, vol. 66, no. 1, pp. 6:1—6:33. DOI: 10.1145/3276769.

9. **Allman M., Paxson V., Blanton E.** RFC 5681: TCP Congestion Control, available at: http://datatracker.ietf.org/doc/rfc5681/

10. **Kuptsov D., Nechaev B., Lukyanenko A., Gurtov A.** How penalty leads to improvement: A measurement study of wireless backoff in IEEE 802.11 networks. *Computer Networks,* 2014, vol. 75, part A, pp. 37—57. DOI: 10.1016/j.comnet.2014.09.008.

11. **Anderton W. C., Chakraborty T., Young M.** Windowed backoff algorithms for WiFi: theory and performance under batched arrivals, *Distributed Computing*, 2021, vol. 34, pp. 367—393. DOI: 10.1007/s00446-021-00403-9.

12. **Mouromtsev D.** Semantic reference model for individualization of information processes in IoT heterogeneous environment, *Electronics (Switzerland),* 2021, vol. 10, no. 20, article 2523. DOI: 10.3390/electronics10202523.

13. **Marchenkov S. A.** Computer-Aided Programming of Software Agents Based on Code Generation in Constructing Semantic Services of Smart Spaces. Part 1, *Programmnaya Ingeneria,* 2019, vol. 10, no. 6, pp. 257—264. DOI: 10.17587/prin.10.257-264 (in Russian).

14. **Kulakov K.** An approach to efficiency evaluation of services with smart attributes, *International Journal of Embedded and Real-Time Communication Systems*, 2017, vol. 8, no. 1, pp. 64—83. DOI: 10.4018/IJERTCS.2017010105.

15. **Korzun D., Gurtov A.** *Structured peer-to-peer systems: Fundamentals of hierarchical organization, routing, scaling, and security*, Springer New York, NY, 2013, 366 p. DOI: 10.1007/978-1-4614-5483-0.

УДК 338.364.2
DOI: 10.17587/prin.14.217-224

Д. С. Пащенко, канд. техн. наук,
MBA, независимый консультант в области разработки программного обеспечения,
denpas@rambler.ru

Полностью удаленная разработка программного обеспечения как новый стандарт IT-отрасли: европейское исследование 2022—2023 гг.

Поступила в редакцию 03.02.2023
Принята к публикации 22.03.2023

Пандемия и становление COVID-экономики продолжают оказывать значительное влияние на организацию труда в отраслях «новой экономики», ярким представителем которой является отрасль информационных технологий (IT-отрасль). К 2023 г. значительное число российских, европейских и мировых IT-корпораций и небольших софтверных компаний существуют в режиме полностью удаленной работы: вне офисов и без личных встреч сотрудников даже внутри одной команды разработки. В статье приведены результаты европейского исследования (декабрь 2022 г. — январь 2023 г.), охватившего опыт 48 команд разработки программного обеспечения и IT-поддержки в компаниях со штаб-квартирами в странах Европы. Эксперты определили уровень востребованности новых организационно-производственных парадигм в разработке программного обеспечения (ПО) и представили свои идеи и прогнозы о развитии полностью удаленной разработки ПО как нового стандарта IT-отрасли в среднесрочной перспективе 2023—2024 гг. Результаты исследования приведены в виде решения соответствующих исследовательских задач и проверки поставленных гипотез об уровне закрепления новых парадигм в практике ведущих IT-компаний и об их влиянии на корпоративную культуру и бизнес-процессы IT-компаний.

Ключевые слова: *IT-отрасль, удаленная работа, информационные технологии, разработка ПО, COVID-экономика, программное обеспечение*

Для цитирования:

Пащенко Д. С. Полностью удаленная разработка программного обеспечения как новый стандарт IT-отрасли: европейское исследование 2022—2023 гг. // Программная инженерия. 2023. Том 14, № 5. С. 217—224. DOI: 10.17587/prin.14.217-224.

Постановка задачи и гипотезы исследования

К 2023 г. значительное число российских, европейских и мировых IT-корпораций и небольших софтверных компаний работают в полностью удаленном режиме — вне офисов и без личных встреч сотрудников даже внутри одной команды разработки. Многие предприятия и банки, располагающие солидными командами собственной (*in-house*) разработки, также выбрали данный режим или перешли на «гибридную» модель, в которой команды собираются в офисе лишь в определенные дни и/или неполным составом. Изучение организации полностью удаленной и «гибридной» моделей работы является актуальной и практически важной задачей, требующей финансовых инвестиций и управленческих усилий [1]. Пандемия COVID-19 и последующие волны социальной изоляции (локдауны) уже изменили IT-отрасль: от ускорения ее собственной цифровизации и виртуализации взаимодействия разработчиков до коренных изменений в организации труда и в производственных процессах. Трансформация производственных процессов в IT-отрасли и, в частности, в командах разработки ПО, приводящая к развитию новой производственно-организационной парадигмы полностью удаленной разработки ПО (FRM — *fully*

remote mode) является предметом изучения и авторских исследований с 2017 г. Первое исследование на этом направлении, посвященное распространенным в отрасли практикам географически распределенной разработки (GDD — *geographically distributed development* [2]) ПО, показало, что российские, мировые и европейские IT-лидеры к 2017 г. активно нанимали сотрудников в режиме «домашнего офиса». Таким образом, активно создавались команды разработки ПО, в которых часть сотрудников физически никогда не появлялись в центрах разработки лично и не имели регулярных очных встреч [3].

Другое авторское исследование 2020 г., посвященное адаптации мировых корпораций к первой волне COVID-19 [4], подтвердило довольно быстрые управляемые процессные изменения в IT-компаниях и их уверенное приспособление к новым условиям пандемии. В исследовании не было выявлено корреляций между переходом на полностью удаленную разработку ПО в условиях локдаунов по всему миру с долговременным снижением качества ПО или даже падением производительности команд. Также было установлено, что значительная часть компаний инвестировала управленческие усилия и ресурсы в управление соответствующими изменениями и добилась успехов. Однако с течением времени стали более заметными другие долгосрочные факторы в производственных и управленческих процессах, которые также нуждались в управляемых изменениях. Еще одно авторское исследование 2021 г. [5] позволило определить актуальность задач долгосрочной мотивации инженеров и их вовлеченности в проектные и корпоративные проблемы в условиях полностью удаленной разработки. Исследование [5] также подтвердило появление новых объективных сложностей в организации производственного процесса в IT-отрасли: от смешения рабочего и личного времени и пространства у специалистов до нетривиальных путей передачи знаний и опыта в командах, а также новых условий обучения и карьерного роста, рисков значительной десоциализации сотрудников. Решение отмеченных в 2021—2022 гг. сложностей позволяет судить о возможностях закрепления парадигмы полностью удаленной разработки как нового стандарта IT-отрасли.

В новом исследовании 2023 г., результаты которого приведены в настоящей статье, основное внимание акцентировано на понимании уровня закрепления полностью удаленной разработки / «гибридного» формата в практике различных компаний (от Казахстана и России до Германии и Франции). В данном исследовании автором выдвинуты **следующие гипотезы**.

1. Парадигма полностью удаленной разработки ПО и IT-сервисов стала новым стандартом IT-отрасли, а использование «гибридной» модели в большинстве случаев является вариантом решения вопросов трудовой дисциплины, мотивации и вовлечения инженеров в корпоративную жизнь, обучения и передачи знаний в командах. Даже после завершения пандемии значительная часть IT-компаний сохранит практики полностью удаленной разработки ПО или «гибридные» модели.

2. Парадигма полностью удаленной разработки уже нашла отражение во всех производственных процессах: от управления задачами и коммуникациями до проектного менеджмента. Все изменения уже проведены, производственные процессы оптимизированы на достаточно высоком уровне.

3. Социальное влияние данной парадигмы значительно, при этом европейские IT-компании и команды разработки ПО к концу 2022 г. уже нашли действенные приемы для борьбы с ее негативными рисками (такими как десоциализация специалистов, снижение продуктивности и мотивации и т. д.)

Сформулированные гипотезы позволяют определить цель и задачи исследования.

Цель, задачи и метод исследования

Цель исследования — изучить и проанализировать процесс многолетней адаптации проектных команд разработки ПО и поддержки IT-сервисов к полностью удаленному режиму работы как к новому стандарту IT-отрасли в Европе в 2022—2023 гг. Подлежащий изучению процесс включает в себя менеджмент основных рисков, реальную эффективность адаптации к изменениям и среднесрочное влияние на будущее IT-отрасли в перспективе 2023—2024 гг.

Среди **частных дополнительных задач** исследования следует выделить следующие:

1) определить уровень востребованности полностью удаленной разработки ПО и IT-сервисов и уровень востребованности «гибридной» модели, уточнить направления для их усовершенствования и необходимость инвестиций со стороны IT-компаний;

2) собрать аргументированные прогнозы и провести отбор идей по дальнейшему влиянию полностью удаленной разработки ПО на производственные и управленческие процессы, а также на корпоративную культуру компаний IT-отрасли.

Решение частных задач направлено на достижение поставленной цели и проверку трех отмеченных выше гипотез. Исследование проводилось в декабре 2022 г. — январе 2023 г. и охватило опыт 48 команд разработки ПО и IT-поддержки в компаниях со штаб-квартирами в странах Европы: Яндекс, Сбер, ВТБ, Deutschebank, Atos IT, Finastra и др. Некоторые из компаний являются софтверными и обладают длительным успешным опытом работы в отрасли, другие поддерживают внутреннюю разработку ПО (*in-house*), охватывающую тысячи специалистов в Европе и обеспечивающую IT-сервисами миллионы клиентов.

Применяемый метод исследования: анкетирование с помощью инструментария Google.Forms, в части случаев — дистанционные интервью. Исследование содержит два основных раздела:

- оценка уровня закрепления удаленной разработки ПО (без личных встреч, без общих офисов) и «гибридной» модели (редкие личные встречи, посещение офиса 1—2 раза в неделю) с учетом выявленных ранее основных проблем и рисков;
- общие прогнозы и представления о влиянии данного отраслевого стандарта на корпоративную культуру IT-компаний и на производственные процессы в перспективе 2023—2024 гг.

Каждая команда была представлена в этом исследовании 1—2 экспертами в ролях менеджера проекта, лидера команд, разработчика ПО, инженера поддержки или аналитика. Большинство экспертов (63 %) имеют восемь и более лет опыта разработки ПО/IT-услуг, другая часть экспертов (25 %) имеет 3—7 лет профессионального опыта в сфере IT. Существенная группа экспертов (33 %) является представителями внутренней разработки (*in-house*), в основном, крупных европейских банков (Sber, Deutschebank) и финтех-организаций, таких как Freedom Finance и Exness Cyprus, а еще большая часть экспертов (38 %) является представителями аутсорсинговых компаний (в основном, международных — Epam, Auriga, Atos IT и т. д.). В исследовании были задействованы эксперты, представившие опыт нескольких мировых поставщиков (вендоров) ПО — Яндекс, Finastra и др.

Все ответы были предоставлены в инструментарии Google.Forms. Значительная часть участников (88 %) также добавили свой прогноз на 2023—2024 гг. в части влияния полностью удаленной разработки на корпоративную культуру в IT-отрасли и возможности закрепления новой организационно-производственной парадигмы в качестве нового стандарта для IT-отрасли. После представления обобщенных результатов участникам значительная группа экспертов (23 %) добавила отзывы об исследовании, которые также повлияли на окончательные результаты.

Выбранный метод исследования подразумевает обобщение полученных мнений экспертов при проверке гипотез и в решении поставленных задач. Основные результаты исследования приведены в следующем разделе.

Основные результаты

Рост востребованности полностью удаленной разработки ПО и IT-сервисов очевиден: около 58 % опрошенных команд к концу 2022 г. работают вообще без офисов и личных встреч. Авторские научные исследования с 2020 г. охватили 100 IT-компаний (в основном, российских и европейских) и показали уверенный рост удельного веса команд в отрасли, для которых полностью удаленная разработка ПО стала организационно-производственной парадигмой: от 31 % в 2020 г. [4] до 58 % в конце 2022 г.

Популярность «гибридной» модели также выросла до 29 % для опрошенных команд в 2022 г., а доля IT-компаний, чьи сотрудники работают полную рабочую неделю в офисах, значительно снизилась, даже несмотря на решение нескольких крупных европейских и российских банков вернуть своих специалистов на работу в офисы в 2023 г. При этом обязательность возврата в офисы в 2023 г. для части респондентов является долгосрочным демотивирующим фактором, в том числе одной из причин для следующей смены работы. Уверенный рост востребованности полностью удаленной разработки / «гибридной» модели подтверждает гипотезу 1, с оговоркой, что возврат частью европейских банков сотрудников в офисы представляется вынужденной мерой (организации не смогли решить набор сопутствующих сложностей в области информационной безопасности, мотивации сотрудников, организации производственных процессов без реальных офисов). Таким образом, задача 1 близка к решению: для значительного числа европейских IT-компаний новая организационно-производственная парадигма является востребованной.

Исследование продемонстрировало, что в компаниях, принявших парадигму полностью удаленной разработки ПО, производственные процессы в основном закрепились в практике, а все необходимые процессные и технологические изменения уже внедрены. Более того, в 2/3 опрошенных команд все улучшения уже были сделаны еще до 2022 г., а к настоящему времени лишь некоторые детали и небольшие изменения дорабатываются на

уровне отдельных команд. При этом треть экспертов отметили, что ранее сделанные инвестиции в развитие производственных процессов полностью удаленной разработки ПО к концу 2022 г. уже недостаточны: новые долгосрочные риски и проблемы требуют дополнительных усилий и вложений.

Одним из важных вопросов исследования в рамках решения задачи 1 является оценка влияния полностью удаленной разработки на значимые параметры производимого ПО и IT-сервисов. Проведенное исследование позволяет сделать вывод о том, что полностью удаленная разработка ПО уже вызвала серьезные изменения во всех областях производственных процессов в IT-компаниях и оказывает влияние на основные параметры программного продукта. Так, для 63 % команд уровень результативности при переходе на полностью удаленную (или «гибридную») парадигму не изменился, а в каждой пятой команде существенно вырос. Более 60 % экспертов также не отмечают прямой корреляции между сдвигом в организационной парадигме разработки ПО в сторону полностью удаленных процессов и долгосрочным уровнем качества разрабатываемого ПО (в 2020 г. на фоне локдаунов уверенность экспертов была выше — 93 %). По мнению автора, это связано с тем, что тенденции роста или падения уровня качества ПО в отдельных компаниях и командах в 2020—2022 гг. носили намного более комплексный характер, а влияние фактора организационно-производственной парадигмы не является доминирующим. Более того, современные высокотехнологичные софтверные компании уже автоматизировали свои процессы обеспечения качества и практически исключили человеческий фактор из этого вопроса [6].

Переход на полностью удаленную разработку / «гибридную» модель совпал с бурным (с 2017 г.) ростом популярности практик DevOps и CI/CD (упрощенно — методологии автоматизации технологических процессов сборки, настройки и развертывания ПО) во всем мире и частично стал катализатором этого процесса [7]. Новый уровень автоматизации в рутинных производственных операциях при создании ПО (включая непрерывные сборки, развертывание и интеграцию ПО, интегрированное автотестирование и анализ кода, развитие инструментов и сред разработки (SDE — *software development environment*) и политик управления релизами) сделал производство ПО еще более обезличенным, а его последующая виртуализация уже не требует личных встреч в офисах, что уменьшило общие усилия команды в каждом проекте. Вместе с тем процессы информационной безопасности в парадигме полностью удаленной разработки ПО стали более сложными и более вариативными, прежде всего в части организации производства ПО. Отчасти это связано с субъективным восприятием информационной безопасности отдельными руководителями (например, в крупных банках), а отчасти — с объективными опасениями быстрой потери лояльности и вовлеченности сотрудников при переходе на полностью удаленную разработку ПО. Эксперты отметили, что доступ к общим информационным системам и средам общей разработки ПО в течение 2021—2022 гг. усложнился.

Следует заметить, что в 2022 г. для 95 % команд не произошло никаких существенных изменений в проектном управлении на фоне закрепления полностью удаленного формата работы в софтверных проектах. Таким образом, все изменения в процессах проектного управления уже были завершены ранее. Исследования 2020—2021 гг. [4, 5] также подтверждали, что полностью удаленная разработка ПО без присутствия в офисах опирается на инженерный опыт команд в организации удаленных и географически распределенных производственных процессов. При этом из работ [4, 5] следует, что изменение организационно-производственной парадигмы в разработке ПО не оказывает существенного долгосрочного влияния на процессы проектного управления и управления качеством программных продуктов.

В данном исследовании также ставился вопрос о взаимосвязи полностью удаленной разработки ПО с ценностями доминирующей в Европе методологии разработки ПО — SCRUM (итерационная «гибкая» командная методология разработки ПО, ставящая разработанный продукт в центр усилий кросс-функциональной команды разработчиков). Так, существует общеизвестное противоречие между идеями SCRUM-разработки и организацией полностью удаленных производственных процессов — это необходимый высокий уровень общности SCRUM-команд: от частых встреч с обсуждением общих проблем и рисков до физического размещения рабочих мест разработчиков рядом друг с другом. По результатам исследования 2022 г. более 70 % команд смогли преодолеть традиционную сложность, связанную с необходимостью «работать как можно ближе» членам SCRUM-команд в сочетании с полностью удаленной разработкой ПО без физических офисов. Разработчики продолжают тесно взаимодействовать, но в виртуальном пространстве с помощью современных средств коммуникаций.

Задача обеспечения эффективных коммуникаций между разработчиками в проектах является центральной в обсуждении развития полностью удаленной разработки ПО без физических офисов [8]. Исследования 2020—2021 гг. [4, 5] показали, что мировые лидеры IT-рынка ищут различные способы повышения эффективности электронных коммуникаций. Вместе с этим опыт Европы в конце 2022 г., собранный в исследовании, подтверждает, что для данного региона большая часть необходимых изменений уже внедрена. Так, эксперты примерно из 70 % команд отметили, что не проводили никаких серьезных изменений в коммуникациях с клиентами и партнерами в 2022 г. Примерно для 30 % команд настройка данных процессов продолжалась и в 2022 г. Коммуникации стали более формальными, появились специальные регламенты взаимодействия разработчиков, были проведены настройки и кастомизации инструментальных средств для электронных каналов коммуникаций. Разработчики ПО, их клиенты и партнеры в Европе почти полностью завершили перестроение коммуникационных процессов в новых условиях. Таким образом, подтверждена гипотеза 2: к концу 2022 г. парадигма полностью удаленной разработки в принявших ее компаниях уже нашла свое отражение во всех производственных процессах: от управления задачами и коммуникациями до проектного менеджмента.

В рамках исследования, результаты которого представлены в настоящей статье, проведен ретроспективный анализ уже принятых в отрасли решений в наиболее сложные периоды времени, к которым относятся волны пандемии COVID-19: первая — начало 2020 г., вторая — начало 2021 г. Эксперты выделили набор наиболее важных факторов, позволивших их командам и IT-компаниям выполнить успешный переход к модели полностью удаленной разработки ПО / «гибридной» модели. К числу таких относятся:

- предыдущий опыт менеджеров и инженеров IT-компаний в удаленной разработке до пандемии (более 58 % респондентов);
- быстрое внедрение соответствующих производственных изменений как реакция на изменения в экономике (около 40 % респондентов).

Эти факторы стали конкурентными преимуществами команд и компаний в 2020—2021 гг. и сохраняют весомый потенциал в конкурентной борьбе и в 2023 г. Эксперты отметили также наиболее часто встречающиеся причины, которые привели к задержкам и экономическим потерям при переходе к полностью удаленной разработке ПО или «гибридной» модели:

- медленная скорость реакции и низкая вовлеченность топ-менеджмента в необходимые корпоративные изменения;
- излишняя уверенность в технологиях (в качестве и стабильности Интернета вне офиса, в функциональности мессенджеров, порталов и корпоративных программных продуктов и т. п.), которая не подтвердилась на практике и потребовала дополнительных корректирующих действий и инвестиций.

Таким образом, решена задача 1 исследования — определены высокий уровень востребованности полностью удаленной разработки ПО и IT-сервисов и значимый уровень востребованности «гибридной» модели, уточнены направления для их усовершенствования и определена необходимость дальнейших инвестиций со стороны менеджмента IT-компаний.

Еще одна гипотеза и задача 2 исследования посвящены отбору идей и прогнозов по дальнейшему влиянию полностью удаленной разработки ПО на производственные и управленческие процессы, на значимые риски десоциализации инженеров, проблемы смешения личного и рабочего времени/пространства и в целом на корпоративную культуру IT-компании. Очевидно, что парадигма полностью удаленных процессов разработки ПО и IT-сервисов оказывает не только экономическое или производственное, но и социальное влияние на вовлеченных в нее специалистов, имеет долгосрочный эффект на весь комплекс процессов HR-управления [8, 9]. В более ранних исследованиях [4, 5] обсуждались различные выявленные аспекты:

- жесткая зависимость эффективности работы членов команд от обеспечения рабочих условий (семья — фактор стресса и фактор поддержки, возможность комфортной работы из дома: шум, свет, рабочее место и т. п.);
- снижение социальной активности сотрудников, как вынужденное в течение карантина, так и последующее после пандемии (при сохранении удаленных рабочих мест);
- для каждого сотрудника и членов его семьи — смешение рабочего времени и пространства с личным.

В данном исследовании наиболее значимым вопросом был определен аспект смешения личного и рабочего времени и пространств для сотрудников (и членов их семей) при полностью удаленной работе в IT-отрасли. Более 63 % экспертов в исследовании указали, что несмотря на значительное влияние парадигмы полностью удаленной разработки ПО на баланс рабочего / личного времени и пространства специалистов, в их командах сотрудники самостоятельно решают подобные сложности. Только в 16 % команд / IT-компаний были

разработаны внутренние инструкции и регламенты, чтобы формализовать организацию производственных процессов в данном аспекте. Типовые процессы обучения, обмена опытом и карьерного роста также стали виртуализированными, однако при этом их прозрачность заметно снизилась: ни в 2021 г., ни в 2022 г. экспертные панели не предложили консолидированных решений для преодоления этой трудности, указывая на необходимость дополнительных инвестиций со стороны менеджмента IT-компаний. Эксперты подтверждают также, что независимо от состояния пандемии и мер по борьбе с ней компаниям следует вкладывать больше денежных средств и внимания в социализацию и вовлечение разработчиков в корпоративную жизнь.

В этом смысле вопрос инвестиций со стороны менеджмента IT-компаний в данные организационно-производственные парадигмы является ключевым. Если в более ранних исследованиях подчеркивалась роль инвестиций в обучение навыкам полностью удаленной работы, то к 2023 г. запрос сместился в сторону технологических инноваций: инструментов цифровой коммуникации, технологий дистанционного обучения и средств поддержки разработки ПО в различных фазах его жизненного цикла. Таким образом, гипотеза 3 не может быть подтверждена полностью: хотя команды и менеджмент ищут возможности и инвестиции на преодоление всех негативных рисков и проблемных вопросов полностью удаленной разработки, но многие сложности еще только предстоит разрешить.

Эксперты также отметили, что полностью удаленные процессы работы в IT-компаниях уже стали частью корпоративной культуры и стремительно вносят разнообразные изменения в соответствующие бизнес-процессы. Среди наиболее часто встречающихся запросов следует выделить:

1) необходимость искать новые и эффективные пути повышения мотивации и вовлечения инженеров в проекты (совместные онлайн- и офлайн-встречи, неформальные модели коммуникаций, встречи в формате «один-на-один»);

2) снижение трудозатрат на изменения в области информационной безопасности (усложнение правил эксплуатации рабочего оборудования, авторизации и т. п.).

В завершении основных результатов в рамках решения задачи 2 исследования приведены некоторые прогнозы и квалифицированные идеи на период 2023—2024 гг. о будущем развития парадигмы полностью удаленной разработки ПО. Значительная часть экспертов (88 %) подтверждают, что полностью удаленная разработка является перспективным (и даже действующим) стандартом в IT-отрасли. При этом следует учитывать перечисленные далее замечания.

1. Во многом переход и закрепление данной организационно-производственной парадигмы в IT-компаниях связан с профессиональной и психологической готовностью к этому топ-менеджмента данных компаний, что накладывает некоторые ограничения на рост ее востребованности.

2. Несмотря на то, что переход к полностью удаленной разработке ПО ослабляет регулярный контроль над трудовой дисциплиной специалистов, он является самым мощным «притягивающим» фактором при найме. Лучшие европейские разработчики не готовы вернуться в офис даже в случае серьезного повышения денежной компенсации.

3. Процессы полностью удаленной работы в IT-компаниях нуждаются в дальнейшей инвестиционной поддержке, которая должна быть направлена:

— на технологическое совершенствование производственных процессов (от коммуникаций, развертывания сред и хостов до автоматических сборок и автотестирования);

— на повышение прозрачности и эффективности в процессах соблюдения трудовой дисциплины, чтобы часы совместной работы команд были обязательными и продуктивными;

— на повышение мотивации и вовлеченности инженеров в общие проекты.

4. Неодинаковые условия работы в офисе и дома для значительной части сотрудников означают необходимость компаний предусмотреть «гибридные» формы работы, т. е. сочетать полностью удаленные процессы с возможностью выбора работником варианта с присутствием в офисе в определенные дни или фазы проекта, спринта, календарного года.

Дальнейшее развитие парадигмы полностью удаленной разработки ПО потребует и других адаптационных шагов по преодолению сложных рисков (например, офлайн-социализация инженеров, рост неполной занятости инженеров) [10], но мировые IT-лидеры уже сделали свой выбор в пользу данной парадигмы. Их HR-процессы перестроены так, что теперь созданные команды по продуктам уже почти невозможно собрать в физических офисах на регулярной основе. Очевидно, что европейские IT-компании в большинстве своем пойдут в этом же фарватере организационного развития, что подтверждает реализацию цели данного исследования.

Выводы

Более ранние исследования [4, 5] показали, что переход на полностью удаленную разработку ПО поддерживается всеми вовлеченными сотрудниками. Такой подход оказывает позитивное мотивационное влияние и рассматривается всеми участниками процесса как новый организационный стандарт даже после завершения пандемии. Представленные в статье результаты исследования убедительно подтверждают, что во всем мире полностью удаленные процессы разработки уже стали частью глобальной корпоративной культуры IT-индустрии и организационно-производственной парадигмой для значимого числа европейских IT-лидеров.

Все соответствующие изменения в производственных процессах и в постановке соответствующих целей в IT-компаниях были завершены, причем в основном еще до 2022 г. Практически решены задачи обеспечения качества, управления проектами, HR-процессов, информационной безопасности, технической поддержки. Более сложные вопросы, такие как прозрачное корпоративное обучение, обмен инженерным опытом, развитие карьеры по-прежнему требуют дополнительных инвестиций и усилий со стороны IT-компаний. Тем не менее парадигма полностью удаленной разработки становится все более востребованной.

Начавшийся более десяти лет назад с географически-распределенной разработки переход на полностью удаленные процессы еще не завершился. Однако он уже стал одним из ключевых конкурентных преимуществ в профессиональном найме разработчиков и хедхантинге [11]. По мере преодоления выявленных в исследованиях сложностей в этой парадигме данное конкурентное преимущество будет становиться все более значительным.

В среднесрочной перспективе 2023—2024 гг. команды позитивно воспринимают парадигму полностью удаленной разработки ПО. Исследование показывает, что управление изменениями при адаптации к новым организационно-производственным парадигмам является регулярным в практике лидеров команд и руководителей IT-компаний.

Переход на полностью удаленные процессы или «гибридную» модель уже оказал значительное влияние на корпоративную культуру компаний, изменил значительную часть бизнес-процессов. При этом со стороны команд разработчиков сохраняется значительный запрос на рост инвестиций в данную организационно-производственную парадигму со стороны IT-компаний: от необходимости развития технологических инструментов и ПО до офлайн-событий для повышения сплоченности команд и социализации их участников.

Список литературы

1. **Казаков А.** Гибридный офис: как создать пространство мечты// VC.RU. 25.03.2022. URL: https://vc.ru/office/387324-gibridnyy-ofis-kak-sozdat-prostranstvo-mechty (дата обращения 03.02.2023).

2. **Espinosa A., Slaughter S. A., Kraut R. E., Hersleb J. D.** Team Knowledge and Coordination in Geographically Distributed Software Development // Journal of Management Information Systems. 2007. Vol. 24, No. 1. P. 135—169. DOI:10.2753/MIS0742-1222240104.

3. **Пащенко Д. С.** Географически распределенные команды: естественные и организационные особенности проектов разработки программного обеспечения // Программная инженерия. 2017. Том 8, № 2. С. 88—95. DOI: 10.17587/prin.8.88-95.

4. **Pashchenko D.** Fully Remote Software Development Due to COVID Factor: Results of Industry Research // International Journal of Software Science and Computational Intelligence. 2020. Vol. 13, No. 3. P. 64—70. DOI: 10.4018/IJSSCI.2021070105.

5. **Пащенко Д. С.** Российский опыт организации полностью удаленной разработки программного обеспечения: отраслевое исследование // Программная инженерия. 2021. Том 12, № 6. С. 311—318. DOI: 10.17587/prin.12.311-318.

6. **Zhai J., Yang Q., Yang Y., Xiao J., Wang Q., Li M.** Automated Process Quality Assurance for Distributed Software Development / K. Berkling, M. Joseph, B. Meyer, M. Nordio (eds) // Software Engineering Approaches for Offshore and Outsourced Development. SEAFOOD 2008. Lecture Notes in Business Information Processing, Springer, Berlin, Heidelberg. 2009. Vol. 16. P. 196—210. DOI: 10.1007/978-3-642-01856-5_14.

7. **Zeller M.** Towards Continuous Safety Assessment in Context of DevOps // SAFECOMP 2021: Computer Safety, Reliability, and Security. SAFECOMP 2021 Workshops, 2021. Lecture Notes in Computer Science book series. Vol. 12853. P. 145—157. DOI: 10.1007/978-3-030-83906-2_11.

8. **Sunil P.** How COVID-19 is impacting HR practices in APAC: Pay freezes, cautious hiring, and more. 27.03.2020. URL: https://www.humanresourcesonline.net/how-covid-19-is-impacting-hr-practices-in-apac-pay-freezes-cautious-hiring-and-more (дата обращения 03.02.2023).

9. **Narain S.** Post COVID-19 pandemic: Hybrid-work model in the new-normal. 10.09.2020. URL: https://www.downtoearth.org.in/blog/governance/post-covid-19-pandemic-hybrid-work-model-in-the-new-normal-73313 (дата обращения 03.02.2023).

10. **Gilbert N**. 17 Remote Work Trends for 2023: Current Forecasts You Should Know. 2022. URL: https://financesonline.com/remote-work-trends/ (дата обращения 03.02.2023).

11. **Гурова И. М.** Дистанционная работа как тренд времени: результаты массового опыта // МИР (Модернизация. Инновации. Развитие). 2020. Том 11, № 2. С. 128—147. DOI: 10.18184/2079-4665.2020.11.2.128-147.

Fully Remote Software Development as a New Standard in the IT Industry: European Study 2022—2023

D. S. Pashchenko, denpas@rambler.ru,
Moscow, 125368, Russian Federation

Corresponding author:

Denis S. Pashchenko, Moscow, 125368, Russian Federation
E-mail: denpas@rambler.ru

Received on February 03, 2023
Accepted on March 22, 2023

The pandemic and the emergence of the COVID-economy continue to have a significant impact on the organization of labor in the industries of the "new economy", of which the information technology industry is a prominent representative. By 2023, a significant number of Russian, European and global IT corporations and small software companies are working in a completely remote mode — outside the offices and without personal meetings of employees even within the same software development team. This article presents the results of a European study (Dec. 2022 — Jan. 2023), covering the experience of 48 software development and IT support teams in companies headquartered in Europe: Yandex, Sberbank, VTB to Deutsche Bank, ATOS IT, Finastra, etc. The experts determined the level of demand for new organizational and production paradigms in software development and presented their ideas and forecasts on the development of fully remote software development as a new IT industry standard in the medium term 2023—2024. The results of the study are presented in the form of solving the relevant research problems and testing the hypotheses about the level of consolidation of new paradigms in the practice of leading IT companies and their impact on the corporate culture and business processes of IT companies.

Keywords: *IT industry, remote work, information technology, software development, COVID economy, software*

For citation:

Pashchenko D. S. Fully Remote Software Development as a New Standard in the IT Industry: European Study 2022—2023, *Programmnaya ingeneria,* 2023, vol. 14, no. 5 pp. 217—224. DOI: 10.17587/prin.14.217-224.

References

1. **Kazakov A.** Gibridnyi ofis: kak sozdat' prostranstvo mechty, *VC.RU,* 25.03.2022, available at: https://vc.ru/office/387324-gibridnyy-ofis-kak-sozdat-prostranstvo-mechty (date of access 03.02.2023) (in Russian).

2. **Espinosa A., Slaughter S. A., Kraut R. E., Hersleb J. D.** Team Knowledge and Coordination in Geographically Distributed Software Development, *Journal of Management Information Systems*, 2007, vol. 24, no. 1, pp. 135—169. DOI: 10.2753/MIS0742-1222240104.

3. **Pashchenko D. S.** Research in CEE-region: Changes Implementation in Software Production, *Programmnaya Ingeneria*, 2017, vol. 8, no. 2, pp. 88—95. DOI: 10.17587/prin.8.88-95 (in Russian).

4. **Pashchenko D.** Fully Remote Software Development Due to COVID Factor: Results of Industry Research, *International Journal of Software Science and Computational Intelligence,* 2020, vol. 13, no. 3, pp. 64—70. DOI: 10.4018/IJSSCI.2021070105.

5. **Pashchenko D. S.** Russian Experience in Organizing Fully Remote Software Development: an Industry Study of 2021, *Programmnaya Ingeneria*, 2021, vol. 12, no. 6, pp. 311—318. DOI: 10.17587/prin.12.311-318 (in Russian).

6. **Zhai J., Yang Q., Yang Y., Xiao J., Wang Q., Li M.** Automated Process Quality Assurance for Distributed Software Development, *Software Engineering Approaches for Offshore and Outsourced Development. SEAFOOD 2008* /K. Berkling, M. Joseph, B. Meyer, M. Nordio (eds), Lecture Notes in Business Information Processing, Springer, Berlin, Heidelberg. 2009, vol. 16, pp. 196—210. DOI: 10.1007/978-3-642-01856-5_14.

7. **Zeller M.** Towards Continuous Safety Assessment in Context of DevOps, *SAFECOMP 2021: Computer Safety, Reliability, and Security. SAFECOMP 2021 Workshops*, 2021, Lecture Notes in Computer Science book series, vol. 12853, pp. 145—157. DOI: 10.1007/978-3-030-83906-2_11.

8. **Sunil P.** How COVID-19 is impacting HR practices in APAC: Pay freezes, cautious hiring, and more, 27.03.2020, available at: https://www.humanresourcesonline.net/how-covid-19-is-impacting-hr-practices-in-apac-pay-freezes-cautious-hiring-and-more (date of access 03.02.2023).

9. **Narain S.** Post COVID-19 pandemic: Hybrid-work model in the new-normal, 10.09.2020, available at: https://www.downtoearth.org.in/blog/governance/post-covid-19-pandemic-hybrid-work-model-in-the-new-normal-73313 (date of access 03.02.2023).

10. **Gilbert N.** 17 Remote Work Trends for 2023: Current Forecasts You Should Know, 2022, available at: https://financesonline.com/remote-work-trends/ (date of access 03.02.2023).

11. **Gurova I. M.** Remote Work as a Trend of Time: Results of Mass Testing, *MIR (Modernizatsiia. Innovatsii. Razvitie),* 2020, vol. 11, no. 2, pp. 128—147. DOI: 10.18184/2079-4665.2020.11.2.128-147 (in Russian).

УДК 004.23, 004.4'24, 004.4'42
DOI: 10.17587/prin.14.225-231

П. Н. Советов, канд. техн. наук, доц. кафедры, sovetov@mirea.ru,
Федеральное государственное бюджетное образовательное учреждение высшего образования «МИРЭА — Российский технологический университет», Москва

Алгоритмы улучшения автоматически синтезированного набора команд расширяемого процессора

Поступила в редакцию 05.03.2023
Принята к публикации 23.03.2023

Процессоры, имеющие расширяемую систему команд, сегодня все чаще используются в качестве программируемых аппаратных ускорителей для различных предметных областей. При расширении RISC-V и других подобных процессорных архитектур возникает потребность в проектировании специализированных команд. Эта задача может быть решена автоматически с помощью применения алгоритмов синтеза команд. В настоящей работе рассмотрены алгоритмы, которые могут использоваться в дополнение к известным подходам и улучшающие набор синтезированных команд за счет перевычисления общих операций (результат которых востребован несколькими операциями) программы в укрупненных синтезированных командах, а также за счет выявления избыточных (имеющих эквиваленты среди других команд) синтезированных команд. Представлены экспериментальные оценки для улучшенных реализаций алгоритмов из областей криптографии, а также трехмерной графики, демонстрирующие, в частности, что для теста на основе шифра AES алгоритм кластеризации общих операций позволяет сократить размер скомпилированного кода на 10 %, а алгоритм поглощения функций позволяет сократить набор синтезированных команд в 2,5 раза.

Ключевые слова: *синтез набора команд, расширяемые процессоры, RISC-V, высокоуровневое проектирование, анализ графа зависимостей, кластеризация графа, SMT-решатель*

Для цитирования:

Советов П. Н. Алгоритмы улучшения автоматически синтезированного набора команд расширяемого процессора // Программная инженерия. 2023. Том 14, № 5. С. 225—231. DOI: 10.17587/prin.14.225-231.

Введение

В настоящее время широко используются программируемые аппаратные ускорители, специально спроектированные для отдельных классов задач. За счет специализации своей аппаратной части такие ускорители могут быть более энергоэффективными по сравнению с процессорами общего назначения. Многообразие имеющихся в настоящее время программируемых аппаратных ускорителей, нацеленных на решение таких классов задач, как, например, криптография, обработка изображений или машинное обучение, позволяет говорить о наступлении «нового золотого века компьютерной архитектуры» [1].

Одним из подходов к проектированию архитектуры программируемого аппаратного ускорителя является специализация тракта данных процессора общего назначения с расширением набора команд предметно-ориентированными инструкциями. Во многих случаях задача проектировщика состоит во введении минимального числа новых команд, обеспечивающих требуемое ускорение на алгоритмах целевого класса. Существует ряд процессорных архитектур с расширяемым набором команд (расширяемых процессоров). К числу таких архитектур относятся: Tensilica Xtensa, ARC, Arm Custom Instructions, RISC-V. Существует также ряд САПР для расширяемых процессоров, в которых проектирование осуществляется с использованием языков описания предметно-ориентированного набора команд. Примерами таких языков являются: TIE (Xtensa processor generator), nML (ASIP Designer), CodAL (Codasip Studio).

Предполагается, что проектировщик создает описание необходимого набора команд вручную, т. е. задача автоматического синтеза специализированных команд в этих САПР не решается.

Автоматический синтез специализированных команд расширяемого процессора может быть основан на анализе кода алгоритмов для решения целевого класса задач в целях извлечения подграфов вычислений — наиболее перспективных участков аппаратного ускорения на разных уровнях программных конструкций с учетом зависимости от заданных архитектурных ограничений. Такие типовые подграфы являются кандидатами в специализированные команды для расширяемого процессора. Существует ряд алгоритмов для синтеза специализированных команд [2] на основе анализа программного кода уровня линейного участка.

В настоящей работе рассмотрены алгоритмы, которые могут использоваться в дополнение к известным подходам и позволяющие добиться улучшения набора синтезированных команд расширяемого процессора. Это улучшение достигается за счет укрупнения синтезированных команд с помощью перевычисления общих операций, а также за счет выявления избыточных синтезированных команд, которые являются частными случаями вычислений других, более общих команд. Приведенные в работе экспериментальные оценки на основе анализа программного кода из областей криптографии, а также трехмерной графики демонстрируют сокращение размера скомпилированных программ и сокращение общего числа синтезированных команд, полученные с использованием разработанных алгоритмов.

Синтез набора команд с его последующим улучшением

Процесс синтеза команд состоит в общем случае из перечисленных далее этапов.

1. Извлечение подграфов команд из графа зависимостей (ГЗ) программы при заданных архитектурных ограничениях.

2. Фильтрация множества извлеченных подграфов в целях избавления от функционально эквивалентных дубликатов и получения минимального набора наиболее предпочтительных кандидатов с точки зрения потенциала аппаратного ускорения.

Синтезированные команды обычно имеют следующие наиболее распространенные формы:

- подграф с одним выходом (*Multiple-Input Single-Output*, MISO);
- подграф с несколькими выходами (*Multiple-Input Multiple-Output*, MIMO);
- SIMD-MIMO — несвязный вариант MIMO, ориентированный на SIMD-вычисления.

Наиболее производительные алгоритмы синтеза команд позволяют извлечь множество MISO-подграфов на уровне линейного участка. Одним из самых простых среди подобных алгоритмов является жадный алгоритм MaxMISO, имеющий линейную вычислительную сложность [3].

На рис. 1 (см. четвертую сторону обложки) показан ГЗ раунда шифра Магма. Серым цветом обозначены подграфы-кластеры, найденные с помощью алгоритма MaxMISO. Можно заметить, что операция сложения, отмеченная голубым цветом (двойным кругом), не попала ни в один из найденных кластеров, поскольку является общей операцией.

Общей операцией в ГЗ является такая операция, исходящие из которой ребра идут на вход как минимум двух других операций. Иными словами, результатом вычисления общей операции пользуются как аргументами две или более иных операций.

Если бы удалось избавиться от общей операции сложения, показанной на рис. 1 (см. четвертую сторону обложки), то это не только сократило бы размер скомпилированного кода, но и дало возможность избавиться от лишнего такта вычислений при использовании параллелизма команд. Такого эффекта можно добиться за счет репликации общих операций в синтезируемых командах.

На рис. 2 (см. четвертую сторону обложки) показаны результаты дополнения алгоритма MaxMISO работой алгоритма кластеризации общих операций, рассматриваемого в следующем разделе. Видно, что кластеры синтезированных команд теперь укрупнены, а размер скомпилированного кода раунда шифрования для RISC-архитектуры сократился на одну команду. Этот подход может использоваться не только с MaxMISO, но и с другими известными алгоритмами извлечения MISO-подграфов.

Из данных рис. 2 (см. четвертую сторону обложки) можно сделать вывод, что для ускорения раунда шифрования Магма требуется добавить в набор команд расширяемого процессора три новых синтезированных инструкции: ci19, ci34 и ci37. При этом ci34 является частным случаем вычислений ci19:

```
ci19(a, b, c, d):= (((a + b) >> c) & 255) + d
   ci34(a, b, c):= ((a + b) & 255) + c
```

Таким образом, для ускорения раунда шифрования Магма достаточно иметь две синтезирован-

ные команды, если имеется возможность задать константные аргументы для ci19:

```
ci19(a, b, 0, c) = ci34(a, b, c)
```

Для выявления избыточных синтезированных команд, являющихся частными случаями других команд, разработан алгоритм поглощения функций, изложенный далее.

Алгоритм кластеризации общих операций

Основная идея итеративного алгоритма кластеризации общих операций состоит в том, что покрытие ГЗ программы подграфами может оказаться более полным, если включать, пока это возможно, в эти подграфы копии общей для них операции.

Псевдокод рассматриваемого алгоритма показан на рис. 3. Основными его этапами являются:

- создание кластеров из одного узла (строки 21—22);
- обход кластеров в порядке топологической сортировки (строки 10—17);
- объединение MISO-кластера с его «пользователями» (строка 14);
- повторение, пока есть изменения (строки 23—26).

В функции IS_LEGAL_MISO, определенной в строках 1—2, осуществляется проверка «синтезируемости» MISO-подграфа: подграф должен быть выпуклым и удовлетворять набору архитектурных ограничений.

Этот алгоритм используется на заключительном этапе извлечения подграфов команд, после применения одного из известных алгоритмов извлечения MISO-подграфов. Разработанный алгоритм также можно использовать и в качестве единственного алгоритма извлечения подграфов команд, но в общем случае этот алгоритм порождает избыточное число кластеров по сравнению с MaxMISO.

Алгоритм поглощения функций

Функция f поглощает функцию g, если кортеж аргументов f можно составить из аргументов g и констант из кортежа c таким образом, что f (аргументы которой составлены из элементов кортежа v) и g становятся эквивалентными по входам-выходам:

$$\exists p_1 \in \{1,\ldots,n\} \ldots \exists p_n \in$$
$$\in \{1, \ldots, n\}.\left(\forall x_1 \ldots \forall x_m . f\left(v_{p_1},\ldots,v_{p_n}\right) = g\left(x_1, \ldots, x_m\right)\right),$$
$$v = \left(x_1, \ldots, x_m, c_1,\ldots, c_{n-m}\right), n \geqslant m.$$

Таким образом, задача определения, поглощает ли функция f функцию g, может быть решена путем подбора соответствующих имен аргументов и значений констант. Эта задача сводится к задаче SMT (*Satisfiability Modulo Theories* — выполнимость формул в теориях) и в ее решении может быть использован SMT-решатель.

На рис. 4 показан псевдокод алгоритма поглощения функций. В этом алгоритме используется техника индуктивного синтеза, управляемого

```
1:  function IS_LEGAL_MISO(G, S)
2:      return (∀i ∈ S \ root(S). ∀u ∈ uses(G[i]). u ∈ S) ∧ arch_constraints(G, S)
3:  function COMBINE_SUBGRAPHS(G, u, i)
4:      return cluster((inputs(G[u])∪inputs(G[i]))\{i}, nodes(G[u])∪nodes(G[i]))
5:  function CAN_COMBINE(G, u, i)
6:      return is_legal_miso(G, nodes(G[u]) ∪ nodes(G[i]))
7:  function COMBINE(G)
8:      G' ← ∅
9:      M ← ∅
10:     for each i ∈ toposort(G, i_end) do
11:         if i ∉ M then
12:             if ∀u ∈ uses(i). can_combine(G, u, i) then
13:                 for each u ∈ uses(i) do
14:                     G'[u] ← combine_subgraphs(G, u, i)
15:                     M[u] ← 1
16:             else
17:                 G'[i] ← G[i]
18:     return G'
19: function CLONE_AND_COMBINE(G)
20:     G' ← ∅
21:     for each i ∈ G do
22:         G'[i] ← cluster(inputs(G[i]), nodes(G[i]))
23:     repeat
24:         G ← G'
25:         G' ← combine(G)
26:     until G = G'
27:     return G
```

Рис. 3. Алгоритм кластеризации общих операций

1: **function** SYNTH(f, g, T)
2: $p \leftarrow (p_1 \in [1..n], \ldots, p_n \in [1..n])$
3: $c \leftarrow (c_1, \ldots, c_{n-m})$
4: **for each** $(x, y) \in T$ **do**
5: $v \leftarrow (v_1, \ldots, v_n)$
6: $\text{SMT.add}(\bigwedge_{i=1}^{n} \left(\bigvee_{j=1}^{m} (p_i = j \wedge v_i = x_j) \vee \bigvee_{j=m+1}^{n-m} (p_i = j \wedge v_i = c_{j-m}) \right))$
7: $\text{SMT.add}(\bigwedge_{i=1}^{m} \bigvee_{j=1}^{n} x_i = v_j)$
8: $\text{SMT.add}(f(v_1, \ldots, v_n) = g(x_1, \ldots, x_m))$
9: **if** SMT.check() = *sat* **then**
10: **return** SMT.model(), **true**
11: **return** $\perp$, **false**
12: **function** VERIFY(f, g, m)
13: $x \leftarrow (x_1, \ldots, x_m)$
14: $v \leftarrow \text{get\_v}(m.p, x, m.c)$
15: $\text{SMT.add}(y = f(v_1, \ldots, v_n) \wedge y \neq g(x_1, \ldots, x_m))$
16: **if** SMT.check() = *sat* **then**
17: **return** (x, y), **false**
18: **return** $\perp$, **true**
19: **function** SUBSUME(f, g)
20: $T \leftarrow \varnothing$
21: **loop**
22: $m, err \leftarrow \text{synth}(f, g, T)$
23: **if** *err* **then**
24: **return** $\perp$, **false**
25: $t, err \leftarrow \text{verify}(f, g, m)$
26: **if** $\neg err$ **then**
27: **return** m, **true**
28: $T \leftarrow T \cup \{t\}$

Рис. 4. Алгоритм поглощения функций на основе синтеза операндов

контрпримерами CEGIS (*Counter-Example Guided Inductive Synthesis*) [4].

Основными элементами рассматриваемого алгоритма являются:

- функция SUBSUME, реализующая CEGIS-цикл синтеза и верификации очередного синтезированного решения (строки 21—28);
- функция SYNTH для синтеза операндов с учетом набора сгенерированных с помощью функции VERIFY тестов (строки 1—11).

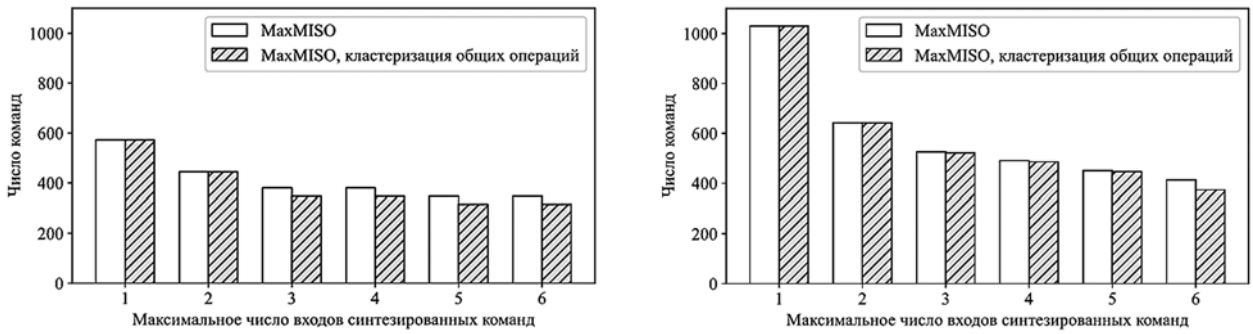
Очередная синтезированная SMT-модель содержит выбранные индексы p имен аргументов функции f, а также значения констант c.

Экспериментальные оценки разработанных алгоритмов

Тестирование разработанных алгоритмов проведено на программных реализациях блочных шифров Магма и AES. С использованием методов, изложенных в работе [5], реализованы варианты перенацеливаемого компилятора с порождением кода для программной модели процессора, имеющего архитектуру RISC-V RV32I, расширенную синтезированными командами. В качестве базового алгоритма извлечения подграфов синтезированных команд использован MaxMISO с архитектурными ограничениями на максимальное число входов извлеченного подграфа в диапазоне 1—6.

На рис. 5 приведены данные по размеру скомпилированного кода для программных реализации двух шифров: Магма и AES.

В тесте Магма кластеризация общих операций сокращает размер кода, начиная с варианта с синтезированными командами, ограниченными тремя входами. Максимальное сокращение размера кода наблюдается на варианте с ограничением на шесть входов и составляет 9 %.



а) *б*)

Рис. 5. Результаты кластеризации общих узлов для программных реализаций шифров:
а — Магма; *б* — AES

Таблица 1

Результаты работы алгоритма поглощения функций для программных реализаций шифров Магма и AES

Максимальное число входов	Сокращение синтезированного набора команд, раз	
	Магма	AES
2	1,7	2,5
3	1,7	1,2
4	1,7	2,3
5	2	1,2
6	2	1,3

Для шифра AES кластеризация общих операций также сокращает размер кода, начиная с варианта, использующего команды, ограниченные тремя входами. Максимальное сокращение размера кода получено на варианте с ограничением на шесть входов и составляет 10 %.

Данные по применению алгоритма поглощения функций для шифров Магма и AES сведены в табл. 1. Максимально набор синтезированных команд для шифра Магма сокращается в 2 раза, а для шифра AES — в 2,5 раза.

В работе [6] приведены примеры результатов работы алгоритма синтеза MISO-команд. В частности, в этой работе показано расширение из пяти синтезированных команд, извлеченных из программного теста Volume Ray-Casting для трехмерной графики. При дополнительном использовании алгоритма поглощения функций число синтезированных команд, как можно видеть из данных табл. 2, сокращено всего до двух команд.

Обзор известных алгоритмов синтеза команд

В табл. 3 приведены характеристики некоторых известных алгоритмов синтеза команд, действующих на уровне линейного участка. С помощью этих алгоритмов из ГЗ программы извлекаются выпуклые подграфы, без общих для нескольких подграфов операций. В работе [7] общие операции учитываются только в специальных условиях: на этапе выбора команд, для исправления результатов алгоритма порождения подграфов с наложениями.

На этапе фильтрации известными алгоритмами синтеза команд не учитывается функциональная эквивалентность подграфов и не определяются случаи, когда один подграф является частным случаем

Таблица 2

Результаты работы алгоритма поглощения функций для теста Volume Ray-Casting [6]

Синтезированная команда	Результат поглощения
`fmadd(A, B, C) := ((A * B) + C)`	`fdot(1, C, B, A, 0, 0)`
`f2madd(A, B, C, D) := ((A * B) + (C * D))`	`fdot(A, B, D, C, 0, 0)`
`f3madd(A, B, C, D, E) := (((A * B) * C) * D) + E`	`Команда не поглощена`
`fdot(A, B, C, D, E, F) := ((A * B) + (C * D)) + (E * F)`	`Команда не поглощена`
`fmmul(A, B, C) := ((A * B) * C)`	`f3madd(C, 1, B, A, 0)`

Таблица 3

Алгоритмы синтеза команд на уровне линейного участка

Источник	Форма подграфа	Ограничения	Извлечение	Фильтрация
Alippi [3]	MISO, без общих операций	Максимальный размер	Жадный алгоритм инкрементальной кластеризации (MaxMISO)	Нет
Atasu [11]	Произвольная, без общих операций	Архитектурные	Инкрементальная кластеризация на основе задачи ЦЛП	На основе изоморфизма
Pozzi [12]	Произвольная, без общих операций	Архитектурные	Оптимальное разбиение графа	Нет
Pulli [13]	Связный MIMO, без общих операций	Нет	Нахождение максимального общего подграфа	Изоморфизм, задача покрытия графа
Xiao [14]	Связный MIMO, без общих операций	Нет	Перечисление всех подграфов	Нет

другого подграфа. В работах [8, 9] используются эвристики, применимые только для специальных случаев, а техника слияния трактов данных [10] требует дополнительного синтеза мультиплексоров.

По данным табл. 3 можно сделать вывод, что известные по публикациям подходы не обладают характеристиками представленных в статье алгоритмов. Тем не менее эти алгоритмы нуждаются в доработке для улучшения синтеза MIMO-команд, что является темой отдельного исследования.

Заключение

Рассмотрены алгоритмы решения задачи синтеза команд расширяемого процессора, представлен краткий обзор известных алгоритмов синтеза команд. Показано, что улучшение работы этих алгоритмов может быть достигнуто за счет перевычисления общих операций в укрупненных синтезированных командах, а также за счет выявления избыточных синтезированных команд, которые являются частными случаями вычислений других команд. Представлены разработанные автором алгоритмы, дополнительно улучшающие работу известных алгоритмов извлечения MISO-графов, а именно алгоритм кластеризации общих операций и алгоритм поглощения функций.

Получены экспериментальные оценки разработанных алгоритмов, демонстрирующие, в частности, что для программной реализации шифра AES алгоритм кластеризации общих операций позволяет сократить размер скомпилированного кода на 10 %, а алгоритм поглощения функций для того же шифра AES сокращает набор синтезированных команд в 2,5 раза.

Список литературы

1. **Hennessy J. L., Patterson D. A.** A new golden age for computer architecture // Communications of the ACM. 2019. Vol. 62, No. 2. P. 48—60. DOI: 10.1145/3282307.

2. **Galuzzi C., Bertels K.** The instruction-set extension problem: A survey // ACM Transactions on Reconfigurable Technology and Systems (TRETS). 2011. Vol. 4, No. 2. P. 1—28. DOI: 10.1145/1968502.1968509.

3. **Alippi C., Fornaciari W., Pozzi L., Sami M.** A DAG-based design approach for reconfigurable VLIW processors // Proceedings of the conference on design, automation and test in Europe. Association for Computing Machinery, NY, USA, 1999. P. 57—es. DOI: 10.1145/307418.307504.

4. **Solar-Lezama A.** Program sketching // International Journal on Software Tools for Technology Transfer. 2013. Vol. 15, No. 5. P. 475—495. DOI: 10.1007/s10009-012-0249-7.

5. **Sovetov P.** Development of DSL compilers for specialized processors // Programming and Computer Software. 2021. Vol. 47, No. 7. P. 541—554. DOI: 10.1134/s0361768821070082.

6. **Nery A. S., Nedjah N., França F. M., Jozwiak L., Corporaal H.** Automatic complex instruction identification for efficient application mapping onto ASIPs // 2014 IEEE 5th latin american symposium on circuits and systems. IEEE, 2014. P. 1—4. DOI: 10.1109/lascas.2014.6820291.

7. **Cong J., Fan Y., Han G., Zhang Z.** Application-specific instruction generation for configurable processor architectures // Proceedings of the 2004 ACM/SIGDA 12th international symposium on field programmable gate arrays. 2004. P. 183—189. DOI: 10.1145/968280.968307.

8. **Peymandoust A., Pozzi L., Ienne P., De Micheli G.** Automatic instruction set extension and utilization for embedded processors // Proceedings IEEE international conference on application-specific systems, architectures, and processors. ASAP 2003. IEEE, 2003. P. 108—118. DOI: 10.1109/asap.2003.1212834.

9. **Clark N. T., Zhong H., Mahlke S. A.** Automated custom instruction generation for domain-specific processor acceleration // IEEE Transactions on Computers. 2005. Vol. 54, No. 10. P. 1258—1270. DOI: 10.1109/tc.2005.156.

10. **Pothineni N., Brisk P., Ienne P., Kumar A., Paul K.** A high-level synthesis flow for custom instruction set extensions for application-specific processors // 2010 15th Asia and South Pacific design automation conference (ASP-DAC). IEEE, 2010. P. 707—712. DOI: 10.1109/aspdac.2010.5419795.

11. **Atasu K., Dündar G., Özturan C.** An integer linear programming approach for identifying instruction-set extensions // Proceedings of the 3rd IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis. 2005. P. 172—177. DOI: 10.1145/1084834.1084880.

12. **Pozzi L., Atasu K., Ienne P.** Exact and approximate algorithms for the extension of embedded processor instruction sets // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. IEEE, 2006. Vol. 25, No. 7. P. 1209—1229. DOI: 10.1109/tcad.2005.855950.

13. **Pulli A., Galuzzi C., Gaydadjiev G.** Towards domain-specific instruction-set generation / 2014 24th international conference on field programmable logic and applications (FPL). IEEE, 2014. P. 1—4. DOI: 10.1109/fpl.2014.6927423.

14. **Xiao C., Wang S., Liu W., Wang X., Casseau E.** An optimal algorithm for enumerating connected convex subgraphs in acyclic digraphs // IEEE Transactions on Circuits and Systems II: Express Briefs. — IEEE, 2020. Vol. 68, No. 1. P. 261—265. DOI: 10.1109/TCSII.2020.3000297.

Algorithms for Improving the Automatically Synthesized Instruction Set of an Extensible Processor

P. N. Sovietov, Associated Professor, sovetov@mirea.ru,
MIREA — Russian Technological University, Moscow, 119454, Russian Federation

Corresponding author:

Petr N. Sovietov, Associated Professor,
MIREA — Russian Technological University, Moscow, 119454, Russian Federation
E-mail: sovetov@mirea.ru

Received on March 05, 2023
Accepted on March 23, 2023

Processors with extensible instruction sets are often used today as programmable hardware accelerators for various domains. When extending RISC-V and other similar extensible processor architectures, the task of designing specialized instructions arises. This task can be solved automatically by using instruction synthesis algorithms. In this paper, we consider algorithms that can be used in addition to the known approaches and improve the synthesized instruction sets by recomputing common operations (the result of which is consumed by multiple operations) of a program inside clustered synthesized instructions (common operations clustering algorithm), and by identifying redundant (which have equivalents among the other instructions) synthesized instructions (subsuming functions algorithm).

Experimental evaluations of the developed algorithms are presented for the tests from the domains of cryptography and three-dimensional graphics. For Magma cipher test, the common operations clustering algorithm allows reducing the size of the compiled code by 9 %, and the subsuming functions algorithm allows reducing the synthesized instruction set extension size by 2 times. For AES cipher test, the common operations clustering algorithm allows reducing the size of the compiled code by 10 %, and the subsuming functions algorithm allows reducing the synthesized instruction set extension size by 2.5 times. Finally, for the instruction set extension from Volume Ray-Casting test, the additional use of subsuming functions algorithm allows reducing problem-specific instruction extension set size from 5 to only 2 instructions without losing its functionality.

Keywords: *instruction set synthesis, extensible processors, RISC-V, high-level design, dependency graph analysis, graph clustering, SMT solver*

For citation:

Sovietov P. N. Algorithms for Improving the Automatically Synthesized Instruction Set of an Extensible Processor, *Programmnaya ingeneria,* 2023, vol. 14, no. 5, pp. 225—231. DOI: 10.17587/prin.14.225-231.

References

1. **Hennessy J. L., Patterson D. A.** A new golden age for computer architecture, *Communications of the ACM* 2019, vol. 62, no. 2, pp. 48—60. DOI: 10.1145/3282307.

2. **Galuzzi C., Bertels K.** The instruction-set extension problem: A survey,*ACM Transactions on Reconfigurable Technology and Systems (TRETS),* 2011, vol. 4, no. 2, pp. 1—28. DOI: 10.1145/1968502.1968509.

3. **Alippi C., Fornaciari W., Pozzi L., Sami M.** A DAG-based design approach for reconfigurable VLIW processors, *Proceedings of the conference on design, automation and test in Europe*. Association for Computing Machinery, NY, USA, 1999, pp. 57—es. DOI: 10.1145/307418.307504.

4. **Solar-Lezama A.** Program sketching, *International Journal on Software Tools for Technology Transfer*, 2013,vol. 15, no. 5, pp. 475—495. DOI: 10.1007/s10009-012-0249-7.

5. **Sovetov P.** Development of DSL compilers for specialized processors, *Programming and Computer Software*, 2021, vol. 47, no. 7, pp. 541—554. DOI: 10.1134/s0361768821070082.

6. **Nery A. S., Nedjah N., França F. M., Jozwiak L., Corporaal H.** Automatic complex instruction identification for efficient application mapping onto ASIPs, *2014 IEEE 5th latin american symposium on circuits and systems,* IEEE, 2014, pp. 1—4. DOI: 10.1109/lascas.2014.6820291.

7. **Cong J., Fan Y., Han G., Zhang Z.** Application-specific instruction generation for configurable processor architectures, *Proceedings of the 2004 ACM/SIGDA 12th international symposium on field programmable gate arrays*, 2004, pp. 183—189. DOI: 10.1145/968280.968307.

8. **Peymandoust A., Pozzi L., Ienne P., De Micheli G.** Automatic instruction set extension and utilization for embedded processors, *Proceedings IEEE international conference on application-specific systems, architectures, and processors*, *ASAP 2003.* IEEE, 2003, pp. 108—118. DOI: 10.1109/asap.2003.1212834.

9. **Clark N. T., Zhong H., Mahlke S. A.** Automated custom instruction generation for domain-specific processor acceleration, *IEEE Transactions on Computer* 2005, vol. 54, no. 10, pp. 1258—1270. DOI: 10.1109/tc.2005.156.

10. **Pothineni N., Brisk P., Ienne P., Kumar A., Paul K.** A high-level synthesis flow for custom instruction set extensions for application-specific processors, *2010 15th Asia and South Pacific design automation conference (ASP-DAC)*,IEEE, 2010, pp. 707—712. DOI: 10.1109/aspdac.2010.5419795.

11. **Atasu K., Dündar G., Özturan C.** An integer linear programming approach for identifying instruction-set extensions, *Proceedings of the 3rd IEEE/ACM/IFIP international conference on hardware/software codesign and system synthesis,* 2005, pp. 172—177. DOI: 10.1145/1084834.1084880.

12. **Pozzi L., Atasu K., Ienne P.** Exact and approximate algorithms for the extension of embedded processor instruction sets, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2006, vol. 25, no. 7, pp. 1209—1229. DOI: 10.1109/tcad.2005.855950.

13. **Pulli A., Galuzzi C., Gaydadjiev G**. Towards domain-specific instruction-set generation, *2014 24th international conference on field programmable logic and applications (FPL)*, IEEE, 2014, pp. 1—4. DOI: 10.1109/fpl.2014.6927423.

14. **Xiao C., Wang S., Liu W., Wang X., Casseau E.** An optimal algorithm for enumerating connected convex subgraphs in acyclic digraphs, *IEEE Transactions on Circuits and Systems II: Express Briefs.,* IEEE, 2020, vol. 68, no. 1, pp. 261—265. DOI: 10.1109/TCSII.2020.3000297.

УДК 004.432
DOI: 10.17587/prin.14.232-244

В. Н. Марков, д-р техн. наук, проф., vinitar@yandex.ru,
Кубанский государственный технологический университет, Краснодар

Аргументы-перечислители как языковый инструмент определения циклов

Поступила в редакцию 13.02.2023
Принята к публикации 27.02.2023

В рамках разработки новых языков высокого уровня и совершенствования существующих языков предложена новая синтаксическая конструкция определения циклов обработки коллекций и рядов значений в виде выражений. Ее суть заключается в использовании перечислителей индексов в качестве аргументов операторов и функций обработки коллекций. Такие аргументы-перечислители позволяют выражать операторы и функции обработки среза/всей коллекции, которые превосходят функции второго порядка в лаконичности и читаемости исходного кода, и не уступают операторам списковых включений.

Ключевые слова: *перечислитель, итератор, индексатор, читабельность кода, лаконичность кода; сечение коллекции, функции высшего порядка, списковые включения*

Для цитирования:

Марков В. Н. Аргументы-перечислители как языковый инструмент определения циклов // Программная инженерия. 2023. Том 14, № 5. С. 232—244. DOI: 10.17587/prin.14.232-244.

Введение

Развитие *языков высокого уровня* (ЯВУ) идет в направлении совершенствования парадигм программирования, организации и обработки структур данных и улучшения выразительной способности языков в целом [1—3]. В этой области языковые инструментальные средства обработки различных видов коллекций являются благодатной почвой для воплощения новых идей в теории языков программирования. Примером тому может служить ставшее классикой теории языков программирования лямбда-исчисление [4], воплотившееся в ЯВУ в виде анонимных функций, функций высших порядков, замыкания и каррирования. Теория множеств, со своей стороны, обогатила ЯВУ нотацией построения коллекций путем указания свойств включаемых элементов с помощью языковой абстракции списковых включений [5, 6]. Такие инструменты объектно-ориентированного программирования, как инкапсуляция и полиморфизм, повышающие декларативность и гибкость исходного кода обработки коллекций [2, 3, 7, 8], стали неотъемлемой частью современных языков. Последним существенным дополнением к инструментам обработки коллекций послужили итераторы [9, 10].

В рамках разработки новых языков программирования и совершенствования существующих в настоящей статье предложена новая синтаксическая конструкция определения циклов обработки коллекций. Ее суть заключается в использовании описываемой ниже нотации перечислителя в качестве *аргумента* коллекции или функции ее обработки. Такой *аргумент-перечислитель* позволяет использовать краткую форму доступа к элементам среза/всей коллекции. В качестве критерия оценки синтаксической конструкции аргумента-перечислителя выбрана очевидность семантики как золотая середина между лаконичностью и читабельностью исходного кода [2].

Источниками предлагаемой нотации перечислителей являются арифметические последовательности Haskell [11] и диапазоны Python [6]. Идея неявного описания циклической обработки коллекций с помощью аргументов-перечислителей навеяна недетерминированными предикатами Prolog [12] и недетерминированными функциями Visual Prolog [13], работающими на откатах. Укороченный оператор присваивания языка APL [14]

лег в основу упрощенной нотации модификации итерируемых элементов коллекций.

В настоящей статье рассмотрены одномерные коллекции, индексация которых по умолчанию начинается с нуля. Трансляция описываемых синтаксических конструкций выполняется в псевдокод, подобный коду на языке C#, без указания типов данных в случаях, где это не имеет значения для раскрытия замысла. Для оценки очевидности семантики и лаконичности операторов представлено сравнение кода предлагаемых синтаксических конструкций с операторами списковых включений и функциями второго порядка.

1. Ряды значений

Введем понятие *ряда* значений, хранящего параметры для вычисления элементов ряда и возвращающего их по одному. Ряды могут быть представлены двумя синтаксически различными способами: с условием продолжения вычисления значений элементов и с фиксированной длиной ряда.

Ряды с условием продолжения (табл. 1) определены тремя параметрами, расположенными в соответствующих полях и заключенными в квадратные скобки с двумя различными разделителями:

```
[init; next .. cond]
```

Здесь `init` – начальное значение члена ряда;
`next` – выражение следующего значения;
`cond` – условие продолжения.

Выражениями по умолчанию являются: ноль для параметра `init`; инкремент `# + 1` для параметра `next`; условие # <= Bound для параметра `cond` при любой явно указанной его правой части Bound.

Форма Бэкуса—Наура (БНФ) ряда с условием опирается на тривиальные БНФ арифметического и логического выражений:

```
<ряд _ усл> → [<выражение> ; <выражение>
.. <лог _ выражение>]
```

Квадратные скобки выбраны в целях повторения обозначения индексов коллекций, но могут быть заменены другими в конкретном ЯВУ. Символы разделителей выбраны также достаточно условно, но различаются, чтобы однозначно вос-

Таблица 1

Примеры рядов с условием продолжения

Полное и сокращенное представления ряда	Значения членов ряда #	Примечание
`[1; #+2 .. # <= 10]` `[1; +2 .. <= 10]`	1 3 5 7 9	От 1 с шагом + 2 пока <= 10. Местоимение #, занимающее левые позиции в выражениях, можно опустить
`[2; # + ln(#) .. ln(#) < Eps]` `[2; + ln(#) .. ln(#) < Eps]`	2 2,7 3,7 4,99	$x_{i+1} = x_i + \ln x_i$, пока $\ln x_i <$ Eps. Значение Eps = 1,6 захвачено извне
`['a'; + 2 .. 'f']`	'a' 'c' 'e'	Использование кодов символов
Использование значений по умолчанию		
`[1; + 2 .. <= 10]` `[1; + 2 .. 10]`	1 3 5 7 9	Условие по умолчанию <= можно опустить
`[2; + 1 .. <= 6]` `[2 .. 6]`	2 3 4 5 6	Инкремент можно опустить, так как 2 < 6
`['a'; + 1 .. 'f']` `['a' .. 'f']`	a b c d e f	Инкремент можно опустить, так как 'a' < 'f'
`[6; - 1 .. 2]` `[6 .. 2]`	6 5 4 3 2	Декремент можно опустить, так как 6 > 2
`[0; + 2 .. 9]` `[+2 .. 9]`	0 2 4 6 8	Нулевое начальное значение можно опустить
`[0; + 1 .. 4]` `[.. 4]`	0 1 2 3 4	Ноль и инкремент можно опустить, так как 0 < 4
`[0; - 1 .. - 4]` `[.. - 4]`	0 −1 −2 −3 −4	Ноль и декремент можно опустить, так как 0 > −4

становить опущенные параметры в полную форму записи ряда.

Ряды могут содержать *местоимения* идентификаторов членов ряда #, ## и т. д. Местоимения введены потому что при сокращенной записи ряда он не будет содержать имя члена ряда. Поэтому неким универсальным именем члена ряда в программном окружении является его местоимение, область видимости которого — оператор, содержащий этот ряд. Символ местоимения члена ряда условен и может быть выбран в конкретном ЯВУ иным уникальным среди идентификаторов языка именем, чем указанное в статье.

Ряд с местоимением `[1; # + 2 .. # <= 10]` надо понимать так: «первый член ряда равен 1, для следующей итерации *его* (#) надо увеличить на 2, пока *он* (#) меньше или равен 10».

Ряд с условием продолжения [`init, next .. cond`] и возможным местоимением # транслируется по следующим правилам.

1. Опущенные выражения восстанавливаются выражениями по умолчанию.

2. Все вхождения местоимения # заменяются во время трансляции именем переменной цикла, которое возвращает служебная функция уникальных имен. Пусть эта функция вернула имя, например, *i*. Тогда следует выполнить подстановку # → *i* в параметрах `next` и `cond`.

3. Ряд с восстановленными выражениями и уникальными именами переменных ряда заменяется одним из двух вариантов кода.

А. Универсальным итератором, например,

```
IEnumerable<> P(init, Func<> next,
Predicate<> cond)
    for (i = init; cond(i); i = next(i))
        yield return i;
```

Вызов итератора

```
foreach (i in P(2,(i) => i+2, (i) => i <= 9))
   Console.Write(i); // 2 4 6 8
```

Типы данных играют второстепенную роль при демонстрации императивной семантики ряда и потому не указаны. Выражение `next` передается в итератор как анонимная функция, условие `cond` — как анонимный предикат.

Б. Итеративным циклом с подставленными параметрами, например,

```
for (i = 2; i <= 9; i = i+2)
    Console.Write(i); // 2 4 6 8
```

Вариант А прост в реализации генератора кода, однако уступает в скорости выполнения варианту Б. Например, в задаче перебора всех пар элементов двух рядов длины *n,* вариант А

```
foreach (i in P(0, i => i+1, i => i < n))
    foreach (j in P(0, j => j+1, j => j < n))
        k++;
```

выполняется медленнее варианта Б

```
for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
        k++;
```

на процессоре AMD Ryzen 5 1600X 6-Core, 3,6 ГГц (табл. 2) из-за накладных расходов на организацию итератора P и обработку анонимных сущностей.

Ряды с фиксированной длиной (табл. 3) заданы также тремя параметрами. Отличием от рассмотренного выше ряда является использование двоеточия в качестве второго разделителя, а также замена третьего поля длиной ряда `len`:

```
[init; next : len]
```

БНФ ряда с фиксированной длиной имеет следующий вид:

```
<ряд _ фикс _ длин> → [<выражение> ; <выражение> :
<выражение>]
```

Ряд с фиксированной длиной `[init; next : count]` и возможным местоимением # транслируется по следующим правилам.

1. Опущенные выражения восстанавливаются выражениями по умолчанию.

2. Все вхождения местоимения # заменяются именем переменной цикла, например, *i* посредством подстановки # → *i* в параметре `next`.

3. Ряд с восстановленными выражениями и уникальными именами переменных заменяется одним из двух вариантов кода.

Таблица 2

Время (с) перебора всех пар элементов двух рядов длины *n*

n	Вариант А	Вариант Б
10000	2,4	0,2
20000	10,2	0,8
30000	23,1	1,8
40000	41,4	3,3

Таблица 3

Примеры рядов с фиксированной длиной

Полное и сокращенное представления ряда	Значения членов ряда #	Примечание
`[1; # + 2 : 5]` `[1; + 2 : 5]`	1 3 5 7 9	От 1 с шагом + 2 всего 5 элементов. Местоимение #, стоящее в левой позиции, можно опустить
`[2; # + ln(#) : 4]` `[2; + ln(#) : 4]`	2 2,7 3,7 4,99	$x_{i+1} + \ln x_i$, всего 4 элемента. Местоимение #, стоящее в левой позиции, можно опустить
`[rand(8); rand(8) : 5]`	4 0 2 4 3	Пять случайных чисел
`['a'; + 2 : 4]`	'a' 'c' 'e' 'g'	Четыре строчных символа
Использование значений по умолчанию		
`[2; + 1 : 4]` `[2 : 4]`	2 3 4 5	Инкремент можно опустить
`[2; - 1 : 4]`	2 1 0 −1	Декремент опустить нельзя
`[0; + 2 : 4]` `[+ 2 : 4]`	2 4 6 8	Нулевое начальное значение можно опустить
`[0; -1 : 3]` `[-1 : 3]`	−1 −2 −3	Ноль опускаем, декремент опустить нельзя
`[0; + 1 : 4]` `[: 4]`	0 1 2 3	Ноль и инкремент можно опустить

А. Универсальным итератором с добавленным счетчиком `len`

```
Enumerable< > P(init, Func<> next, count)
   for (i = init, len = 0; len < count;
i = next(i), len++)
       yield return i;
```

Вызов перечислителя

```
foreach (i in P(2,(i) => i+2, 4))
Console.WriteLine(i); // 2 4 6 8
```

Б. Итеративным циклом с подставленными параметрами, например

```
for (i = 2, count = 0; count < 4; i = i+2,
count++)
```

Каждый вид ряда может одновременно получать элементы нескольких математических рядов. Для этого в первом поле через запятую перечисляются начальные значения членов ряда, во втором поле также через запятую указываются выражения для итерации переменных в том же порядке. Такие ряды могут использовать местоимения, например, # — для первой переменной ряда, ## — для второй переменной и т. д. (табл. 4).

Ряд с условием продолжения и с двумя переменными типа `int`

```
[init1, init2; next1, next2 .. cond]
```

транслируется в один из двух вариантов кода:

а) универсальный итератор, например,

```
IEnumerable<(int,int)> P(int init1, int init2,
            Func<int,int> next1, Func<int,
int> next2,
            Predicate<(int,int)> cond)
{
    for (int i = init1, j = init2; cond((i,j));
```

Таблица 4

Примеры рядов с двумя переменными

Полное и сокращенное представления ряда	Значения членов ряда # и ##	Примечание
`[3, 1; # + 3, ## + 1 .. # < = 10]` `[3, 1; + 3, + 1 .. # < = 10]`	(3,1) (6,2) (9,3)	Первая переменная # изменяется от 3 с шагом +3 пока она не больше 10. Вторая переменная ## изменяется от 1 с шагом +1 пока первая переменная не больше 10
`[1, 1; ##, # + ## : 4]`	(1,1) (1,2) (2,3) (3,5)	Четыре пары смежных членов ряда Фибоначчи

```
        i = next1(i), j = next2(j))
        yield return (i,j);
}
```

вызов такого итератора тривиальный, например,

```
foreach ((int,int) T in P(2,5,(i) => i+2, (i)
=> i+1, (T) => T.Item1<T. Item2))
         Console.WriteLine(T.ToString()); // (2,
5) (4, 6) (6, 7)
```

б) итеративный цикл с подставленными параметрами, например,

```
for (int i = 2, j = 5; i < j; i = i+2,
j = j+1)
    Console.WriteLine($"({i}, {j})"); // (2, 5)
(4, 6) (6, 7)
```

Ряд с фиксированной длиной и с двумя переменными, а также ряды с тремя и более переменными транслируются аналогично.

Трансляцию на основе универсальных итераторов допускается использовать там, где важны унификация и скорость разработки, например, в интеллектуальных решателях задач, в интерпретаторах, в средствах автоматического построения программ по спецификациям, включающим аргументы-перечислители, а также при разработке прототипов программ. Трансляцию в итеративные циклы без использования анонимных сущностей предлагается использовать там, где важна скорость выполнения кода.

2. Сечения коллекций на основе рядов индексов

Используя ряд в качестве *перечислителя* индексов элементов коллекций, можно получать сечения коллекций. Пусть *M*[*i*] — *i*-й элемент коллекции с произвольным доступом или элемент ссылочной коллекции, хранящийся по *i*-й ссылке в последовательности ссылок коллекции. Подставим описанный выше ряд с условием продолжения `[init; next.. cond]` вместо индекса элемента коллекции. Получим запись `M[init; next .. cond]`. Такая нотация обозначает *сечение* коллекции с последовательным доступом к ее элементам по индексам, возвращаемым указанным рядом (табл. 5). Аналогичным образом поступим с сечением коллекции посредством ряда индексов с фиксированной длиной `M[init; next : len]`.

Сечения коллекций используют следующие местоимения: # — для индекса элемента коллекции; @ — для позиции индекса в ряде. Выражение # <= −1 является условием по умолчанию для продолжения итераций.

Подставляя в обозначение элемента двумерной коллекции *M*[*i, j*] вместо индексов *i*, *j* ряд с двумя переменными, получим обозначение ее сечения, например `M[init1, init2; next1, next2 ... cond]`. Сечения двумерных коллекций в этой статье далее не рассматриваются по причине ограничения ее объема.

Реальное значение индекса вычисляется в кольце по модулю (*Length* + *n*) mod *Length*, равному

Таблица 5

Примеры сечений коллекции *M* = {4, 3, 2, 1, 0, −1, −2, −3, −4, −5, −6}

Эквивалентные обозначения сечений	Элементы сечения `M[#]`	Примечание
`M[1; + 2 .. <= 7]` `M[1; + 2 .. 7]`	3 1 −1 −3	От индекса 1 с шагом 2 пока индекс <= 7
`M[0; + 2: 5]` `M[+ 2 : 5]`	4 2 0 −2 −4	Первые пять элементов, стоящих на четных позициях
`M[0; + 2 .. - 1]` `M[+ 2 ..]`	4 2 0 −2 −4 −6	Все элементы с четными индексами
`M[10; - 1 : 4]` `M[- 1; - 1 : 4]`	−6 −5 −4 −3	Последние четыре элемента
`M[8; + 1 .. - 1]` `M[8 ..]`	−4 −5 −6	Последние элементы, начиная с индекса 8
`M[3; - 1 .. 0]` `M[3; - 1 ..]`	1 2 3 4	Первые четыре элемента в обратном порядке
`M[3; + 1 : 4]` `M[3 : 4]`	1 0 −1 −2	Четыре смежных элемента, начиная с индекса 3
`M[2; - 1 : 5]`	2 3 4 −6 −5	От индекса 2 с шагом -1 длиной 5
`M[8; + 1: 5]`	−4 −5 −6 4 3	От индекса 8 с шагом + 1 длиной 5

длине коллекции *Length*, подобно подходу, применяемому, например в Python 15]. Отрицательный индекс $-n$ указывает на элемент коллекции, имеющий реальный индекс (*Length* − *n*) mod *Length*. Последний элемент коллекции *M* может быть записан в виде `M[-1]`.

Перечислители всех элементов коллекции имеют следующий вид:

- `[..]` — перечислитель от первого элемента к последнему;
- `M[-1 .. 0]` — перечислитель от последнего элемента к первому.

3. Операторы с аргументами-перечислителями

Используя ряды как аргументы выражений и/или операторов, можно лаконично выражать различные виды циклической обработки коллекций и рядов. В качестве сокращения будем использовать название «α-операторы» для упомянутых выше сущностей. Такой α-оператор может быть оператором присваивания, ветвления или вызовом функции/процедуры. α-оператор является областью видимости местоимений ряда.

Приводимые примеры (табл. 6) используют укороченный синтаксис оператора присваивания [14] для случая модификации переменных. α-операторы, аналогичные предикатам второго порядка обработки связных списков [16], имеют очевидную семантику и лаконичный синтаксис.

α-оператор `if (M[..] > 1) Q[#] + 5;` следует читать так: «если очередной элемент коллекции *M* больше 1, то элемент коллекции *Q* с *таким же* индексом # увеличить на 5».

4. Трансляционная семантика α-операторов обработки коллекций с произвольным доступом

Пусть некоторый α-оператор задан в общем виде `Op(Iter1, Iter2, #, ##)` и содержит два ряда индексов `Iter1, Iter2` элементов коллекций *M, Q* соответственно, а также все возможные местоимения #, ##. Пусть каждый ряд `Iter1, Iter2` определен в виде `[init1, next1 .. cond], [init2, next2 : len]`

Таблица 6

Примеры α-операторов

α-оператор	Результат	Примечание
`if ([1; + 2 : 5] % 5 > 2) write(#);`	3 9	Вывод элементов ряда 1 3 5 7 9, удовлетворяющих условию
`if ([1; + 2 : 5] % 5 > 2) write(@);`	1 4	Вывод индексов элементов ряда 1 3 5 7 9, удовлетворяющих условию
`M.Add([3; - 1 .. - 1]);`	{3,2,1,0,−1}	Построение коллекции *M*
Обработка элементов коллекции *M* = {3, 2, 1, 0, —1}		
`s = 0;` `s - M[..];`	−5	Левоассоциативный Fold относительно вычитания
`s = 0;` `s - M[-1 .. 0];`	1	Правоассоциативный Fold относительно вычитания
Модификация элементов коллекции *M* = {3, 2, 1, 0, —1}		
`M[..] + 5;`	{8,7,6,5,4}	Модификация всей коллекции
`M[1; + 3 : 2] + 5;`	{3,7,1,0,4}	Модификация сечения коллекции
`if (M[+ 2 .. ] > 0) M[#] + 5;`	{8,2,6,0,4}	Условная модификация сечения коллекции
Получение новых коллекций *R* и *Q* из коллекции *M* = {3, 2, 1, 0, -1}		
`R.Add(M[..] + 5);`	{8,7,6,5,4}	Отображение Map
`if (M[..] > 0) R.Add(M[#]);`	{3,2,1}	Фильтрация Filter
`if (M[..] > 1) R.Add(M[#])` `else Q.Add(M[#]);`	{3,2 } и {1,0,−1}	Фильтрация *M* на две коллекции *R* и *Q*
`if (M[..] > 0) R.Add(M[#] + 5);`	{8,7,6}	Отображение FilteredMap
Получение новой коллекции *R* из коллекций *M* = {3, 2, 1, 0, —1} и *Q* = {1, 5, 10}		
`R.Add(M[..] + Q[..]);`	{4,7,11}	Отображение ZipMap
`if (M[..] < Q[..]) R.Add(M[#] + Q[##]);`	{7,11}	Отображение FilteredZipMap

соответственно. В случае обработки коллекций с произвольным доступом ряды `Iter1, Iter2` являются, по сути, индексаторами.

Отмеченного выше вида α-оператор транслируется в итеративный цикл в соответствии с перечисленными далее шагами.

Шаг 1. Опущенные выражения восстанавливаются выражениями по умолчанию.

Шаг 2. Все вхождения местоимений #, ## заменяются уникальными переменными, например, *i, j* соответственно.

Шаг 3. Все вхождения элементов коллекций *M*[#], *Q*[##] заменяются именами *M*[*i*], *Q*[*j*] соответственно.

Шаг 4. Из параметров двух рядов [`init1, next1 .. cond`] и [`init2, next2 : len`] строится заголовок цикла, в котором все шесть параметров заменяются явными выражениями

```
for (i=init1, j=init2, count=0;
cond && count<len;
     i=next1, j=next2, count++)
```

Шаг 5. В полученном α-операторе `Op(Iter1, Iter2, i, j)` выполняется замена рядов `Iter1, Iter2` именами переменных цикла *i*, *j* соответственно. Сечения коллекций `M[init1, next1 .. cond]` и `Q[init2, next2 : len]` заменяются элементами коллекций *M*[*i*] и *Q*[*j*] соответственно, и следовательно, α-оператор становится оператором над элементами коллекций `Op(M[i], Q[j], i, j)`, который размещается в теле цикла

```
for (i=init1, j=init2, count=0;
cond && count<len;
     i=next1, j=next2, count++)
{    Op(M[i], Q[j], i, j);}
```

Таким образом, параметры рядов определяют заголовок цикла. Сам оператор, претерпев замену местоимений программно генерируемыми именами переменных, размещается в теле цикла.

Пример 1. Трансляция условного α-оператора, строящего коллекцию *R* из сумм элементов исходных коллекций *M* = {2, 4, 6, 8, 10, 12} и *Q* = {2, 10, 20}, удовлетворяющих условию *M*[*i*] < *Q*[*i*]

```
   if (M[..] < Q[#]) R.Add(M[#] + Q[#]);
```

в перечислитель, имеет вид

```
foreach (i in P(0,(i) => i+1,
(i) => i<M.Length && i<Q.Length))
    if (M[i] < Q[i]) R.Add(M[i] + Q[i]); // 14 26
```

Здесь итератор *P* определен кодом

```
IEnumerable<int> P(init, Func<int,int>
next, Predicate<int> cond)
    for (int i = init; cond(i); i = next(i))
        yield return i;
```

Трансляция в итеративный цикл имеет вид

```
for (i = 0; i<M.Length && i<Q.Length; i++)
    if (M[i] < Q[i]) R.Add(M[i] + Q[i]); // 14 26
```

Пример 2. В случае двух различных аргументов-перечислителей в задаче примера 1

```
if (M[+ 2 ..] < Q[..]) R.Add(M[#] + Q[##]);
```

α-оператор транслируется в итератор

```
foreach ((x,y) in P(0, 0, (i) => i+2, (j) => j+1,
(T) => T.Item1 < M.Length &&
T.Item2 < Q.Length))
    if (M[x] < Q[y]) R.Add(M[x] + Q[y]); // 16 30
```

Трансляция в итеративный цикл имеет вид

```
for (i = 0, j = 0; i < M.Length && j <
Q.Length; i = i+2, j++)
    if (M[i] < Q[j]) R.Add(M[i] + Q[j]); // 16 30
```

5. Трансляционная семантика α-операторов обработки ссылочных коллекций

Аргумент-перечислитель с условием продолжения `[init; next .. cond]` для ссылочной коллекции RefCollection возвращает в общем случае элементы ее сечения. С помощью итератора (вариант А) он вызывается тривиально:

```
customIter1 = RefCollection.GetIteratorCond
(init, next, cond);
```

Этот итератор передвигает внутренний указатель ссылочной коллекции вначале на позицию `init` при выполнении условия `cond`, а при следующих итерациях — на позицию, определяемую выражением `next`, до тех пор, пока выполняется условие `cond`, например,

```
public IEnumerable<Node<T>>
GetIteratorCond(int init, Func<int, int>
next, Predicate<int> cond)
{
   Node<T> Temp = First;
   int count = 0; // номер позиции
```

```
  if (Size > init)
    for (; count < init; count++)
      Temp = Temp.Next; // к позиции init
  else
      yield break;
  while (cond(count))
  {
    yield return Temp; // к следующей позиции
    int d = next(count);
    for (; count < d; count++)
        if (Temp == null) yield break;
        else Temp = Temp.Next;
  }
}
```

Реализация аргумента-перечислителя в виде итеративного цикла (вариант Б) является, по сути, повторением тела функции `GetIteratorCond` с подставленными вместо анонимных сущностей `next` и `cond` выражениями. В задаче перебора пар элементов, стоящих на четных позициях двух связных списков длины *n*, варианты А и Б сопоставимы по скорости работы (табл. 7). Причиной отличия являются накладные расходы на организацию и обработку анонимных сущностей `next` и `cond`.

Аргумент-перечислитель с фиксированной длиной `[init; next: len]` реализуется и вызывается аналогично

```
customIter2 = RefCollection.GetIteratorLen
(init, next, len);
```

Указанный итератор передвигает внутренний указатель ссылочной коллекции вначале на позицию `init`, если значение `len` больше нуля, а при следующих итерациях — на позицию, определяемую выражением `next`. Всего возможно не более чем `len` итераций в случае, если не встретится пустая ссылка.

Рассмотрим случай одновременной обработки двух ссылочных коллекций одного вида. Пусть некоторый α-оператор `Op(Iter1, Iter2, #, ##)`

Таблица 7

Время (с) перебора пар элементов сечений связных списков длины *n*

n	Вариант А	Вариант Б
10000	0,9	0,5
20000	3,7	2,0
30000	8,5	4,4
40000	14,8	7,9

имеет два перечислителя `Iter1`, `Iter2` элементов коллекций *M* и *Q* соответственно, и возможные местоимения #, ##. И пусть перечислители `Iter1`, `Iter2` пробегают ряд значений видов `[init1, next1 .. cond]` и `[init2, next2 : len]` соответственно.

Такой α-оператор транслируется в цикл, который получает пары элементов (`Item1, Item2`) коллекций *M* и *Q* с помощью перечислителя `Iterator` двух коллекций класса RefCollection и выполняет оператор над ними:

```
foreach ((Item1, Item2) in RefCollection.Iterator
(M, init1, next1, cond, Q, init2, next2,
len))
    Op(Item1, Item2);
```

В случае одновременной обработки двух коллекций *разного* вида создается код перечислителя, который использует предикат `MoveNext` [17] для продвижения к следующему элементу среза/коллекции, а также геттер и сеттер для доступа к элементам по чтению/записи [8].

Пример 3. Условный α-оператор отображения результата фильтрации среза односвязного списка *M* = {6, 3, 2, 8, 7, 4, 9, 1, 5, 15} в коллекцию *R*

```
if (M[+ 2 .. < 8] > 5) R.Add(M[#] + 3);
```

транслируется в псевдокод

```
foreach (m in M.IteratorCond(0, x => x + 2,
y => y < 8))
    if (m.Value > 5)
       R.Add(m.Value + 3); // 9 10 12
```

Пример 4. Условный α-оператор отображения в коллекцию *R* суммы элементов двух односвязных списков *M* = {6, 3, 2, 8, 7, 4, 9, 1, 5, 15} и *Q* = {1, 2, 4, 5, 8}, просматриваемых двумя различными перечислителями

```
if (M[+ 2 ..] > Q[1, + 1 : 3]) R.Add(M[#] +
Q[##]);
```

транслируется в псевдокод

```
foreach ((t1, t2) in SinglyLinkedList.Iterator3
(M,0, x => x + 2, y => y < M.Size,Q,1,
x => x + 1, 3))
    if (t.Value > t2.Value)
        R.Add(t1.Value + t2.Value); // 8 12
```

Здесь статический перечислитель `Iterator3` класса односвязных списков SinglyLinkedList получает параметры ряда `[0; x => x + 2 .. y => y < M.Size]`

для итераций коллекции *M* и параметры ряда `[1; x => x + 1 : 3]` для итераций коллекции *Q*. Итерации этих коллекций выполняются одновременно, возвращая в переменной `t1` очередной элемент коллекции *M*, а в переменной `t2` очередной элемент коллекции *Q*.

6. Циклы с именованными значениями

Принятый α-оператор над двумя и более коллекциями не позволяет их обрабатывать по принципу «каждый элемент одной коллекции с каждым элементом другой коллекции». Следующий код вместо декартового произведения выведет пары элементов, занимающих одинаковые позиции в своих коллекциях:

```
W = {1, 2, 3};
Q = {10,20};
Write($"{Q[..]},{W[..]}; "); // 1,10; 2,20;
```

Для обработки элементов коллекций по принципу «каждый с каждым» необходимо получать элементы одной коллекции во внешнем цикле и передавать их имена во внутренний цикл, просматривающий элементы другой коллекции.

Используем инфиксный оператор <– [1] для получения именованных значений членов ряда

```
i <- [0; + 1 .. 4]; // 0 1 2 3 4
```

или значений пары членов

```
i,j <- [3,1; + 2, + 3 : 4]; // 3,1 5,4 7,7 9,10
```

или значений элементов коллекции

```
x <- M[0; + 2 .. 8]; // M[0] M[2] M[4] M[6] M[8]
```

Именованные значения передаются в качестве параметров во вложенный цикл.

Пример 5. α-оператор получения в коллекции *P* декартова произведения коллекций *Q* = {10, 20} и *W* = {1, 2}

```
foreach (x <- Q)
    P.Add((x, W[..]));
```

транслируется в псевдокод с вложенным циклом

```
foreach (x in Q)
    foreach (y in W)
        P.Add((x, y)); // {(10,1), (10,2), (20,1),
(20,2)}
```

Пример 6. α-оператор получения фильтрованного декартова произведения коллекций *Q* = {3, 7} и *W* = {1, 5, 9} в коллекции *P*

```
foreach (x <- Q)
if (x > W[..])
P.Add((x, W[#]));
```

транслируется в псевдокод с вложенным циклом

```
foreach (x in Q)
    foreach (y in W)
        if (x > y)
            P.Add((x, y)); // {(7,1), (7,5)}
```

7. Сравнение α-операторов и операторов списковых включений

Списковые включения будем записывать в виде, подобном нотации включения в Haskell [1] и в Visual Prolog [6]:

```
M = {элемент : итератор; итератор; ... условие};
```

Пример 7. Построение оператором спискового включения коллекции простых чисел в диапазоне от 3 до 11 с использованием предиката проверки простоты числа `isPrime(x)` имеет вид

```
P = {x : x <- [3, + 2 .. 11]; isPrime(x)};
```

и записывается с помощью α-оператора как

```
if (isPrime([3, + 2 .. 11])) P.Add(#);
//{3, 5, 7, 11}
```

Пример 8. Построение zip-коллекции {(2, "red"), (5, "green")} на основе коллекций *Q* = {2, 5} и *W* = = {"red", "green", "blue"} с помощью замыкания

```
P = {(x,y) : x,y <- Q,W};
```

записывается с помощью α-оператора как

```
P.Add((Q[..], W[..]));
```

Пример 9. Пересечение коллекций *Q* = {2, 5} и *W* = {1, 2, 3} с помощью оператора замыкания и инфиксного предиката *in,* определяющего вхождение значения в коллекцию, строится в виде

```
P = {x : x <- Q; x in W};
```

и с помощью α-оператора записывается как

```
if (Q[..] in W) P.Add(Q[#]);  // {2}
```

Пример 10. Построение разности коллекций $Q = \{2, 5\}$ и $W = \{1, 2, 3\}$ с помощью замыкания имеет вид

```
P = {x : x <- Q; !(x in W)};
```

и с помощью α-оператора записывается как

```
if (!(Q[..] in W)) P.Add(Q[#]);  // {5}
```

Как видно из приведенных примеров 7—10, α-оператор не уступает в лаконичности и читабельности оператору списковых включений.

8. Сравнение α-операторов и функций второго порядка

Сравним читаемость и лаконичность известных функций второго порядка, вызванных и определенных в псевдокоде, и эквивалентных им α-операторов.

Пример 11. Функция `map` отображения коллекции $P = \{1, 2, 3\}$ в коллекцию Q вызывается и определяется в псевдокоде в виде

```
Q = P.map((x) => x+5);  // вызов функции
SomeCollection<T> map(P, Function(T, T) F)
{   foreach (x in P) Q.Add(F(x)); // определение
// функции
    return Q;}
```

α-оператор позволяет определить и вызвать такой код одним движением:

```
Q.Add(P[..] + 5);
```

Пример 12. Функция `filter` фильтрации коллекции $P = \{1, 2, 3, 4, 5\}$ возвращает новую коллекцию $Q = \{4, 5\}$:

```
Q = P.filter((x) => x>3); // вызов
SomeCollection<T> filter(P, Predicate(T) P)
// определение
{   foreach (x in P) if (P(x)) Q.Add(x);
    return Q;}
```

Эквивалентный α-оператор имеет вид

```
if (P[..]>3) Q.Add(P[#]);
```

Пример 13. Процедура фильтрации `filter` коллекции $M = \{1, 2, 3, 4, 5\}$ возвращает коллекцию `Pos` = {4, 5}, удовлетворяющую условию фильтрации, и коллекцию `Neg` = {1, 2, 3}, неудовлетворяющую условию фильтрации:

```
M.filter((x) => x>3, Pos, Neg);  // вызов
filter(M, Predicate(T) P, out T Pos, out T
Neg) // определение
    foreach (x in M) if (P(x)) Pos.Add(x);
else Neg.Add(x);
```

Аналогичный α-оператор имеет вид

```
if (M[..]>3) Pos.Add(M[#]); else Neg.Add(M[#]);
```

Пример 14. Функция фильтрующего отображения `filteredMap` коллекции $M = \{1, 2, 3, 4, 5\}$ в коллекцию $Q = \{9, 10\}$, выполняет отображение тех элементов исходной коллекции M, которые удовлетворяют заданному условию

```
Q = P.filteredMap((x) => x>3,(x) => x+5); //
вызов
SomeCollection<T> filteredMap(M,
Predicate(T) P, Function(T,T) F)
{   foreach (x in M) if (P(x)) Q.Add(F(x));
// определение
return Q;}
```

α-оператор такого отображения определяется и вызывается в виде

```
if (M[..]>3) Q.Add(M[#]+5);
```

Пример 15. Функция `foldl` собирает элементы коллекции $M = \{1, 2, 3\}$ с числом 10 с помощью анонимной функции, просматривая коллекцию слева направо

```
y = M.foldl((s,x) => s+x, 10);  // вызов
T foldl(M, Function(T,T) F, s)   // определение
{   y = s;
    foreach (x in M) y = F(y,x);
    return y;} // ((10+1)+2)+3
```

определяется и вызывается α-оператором

```
s = 10;
s + M[..];
```

Пример 16. Функция `unfold` создает коллекцию элементов $M = \{25, 16, 9, 4, 1\}$ по заданной функции из исходного скаляра 5. Она выполняет возведение в квадрат чисел, лежащих на отрезке $5 \geqslant x \geqslant 1$. Для упрощения псевдокода кортеж объявляется выражением (T, T):

```
M = unfold(5, (y) => (y*y, y-1), (z) => z>=1);
// вызов
SomeCollection<T> unfold(x, Function(T, (T,T)) F,
Predicate(T) P)
```

```
{    while (P(x)) // определение
         {(y,x) = F(x);
         M.Add(y);}
     return M;}
```

С помощью α-оператора эта функция определяется и вызывается в виде

```
M.Add([5; -1 .. 1]^2);
```

Здесь циркумфлекс ^ обозначает возведение в степень.

Пример 17. Функция `zipMap` отображает коллекции $Q = \{1, 2, 3\}$ и $W = \{10, 20, 30\}$ в новую коллекцию $P = \{11, 22, 33\}$, содержащую суммы элементов исходных коллекций, стоящих на одинаковых позициях:

```
P = zipmap(Q,W,(x,y) => x+y);  // вызов
SomeCollection<T> zipmap(Q, W,
Function((T,T),T) F) // определение
{    foreach ((x,y) in Q,W) P.Add(F(x,y));
     return P;}
```

Эквивалентный этому коду α-оператор имеет простой вид

```
P.Add(Q[..] + W[..]);
```

Рассмотренные примеры 11—17 показывают, что α-оператор превосходит в лаконичности и читаемости ряд известных функций второго порядка.

9. Поразрядная сортировка с использованием α-операторов

Модифицируем функцию языка C# поразрядной сортировки коллекции *m*, вторым аргументом которой является маска разряда *p*:

```
static List<int> RadixSort(List<int> m, uint p)
{
    if (m.Count <= 1 || p == 0) return m;
    List<int> a = new List<int>();
    List<int> b = new List<int>();
    foreach (int e in m)  // цикл разделения
       if ((e & p) == 0) a.Add(e);
       else b.Add(e);
    p >>= 1;
    List<int> a1 = RadixSort(a, p);
    List<int> b1 = RadixSort(b, p);
    foreach (int e in b1)  // цикл соединения
       a1.Add(e);
    return a1;
}
```

Цикл разделения списка заменяем условным α-оператором

```
if ((m[..] & p) == 0) a.Add(m[#]);
else b.Add(m[#]);
```

Цикл соединения списков заменяем безусловным α-оператором

```
a1.Add(b1[..]);
```

В результате получаем новый код:

```
static List<uint> RadixSort(List<uint> m,
uint p)
   {
   if (m.Count <= 1 || p == 0) return m;
   List<uint> a = new List<uint>();
   List<uint> b = new List<uint>();
   if  ((m[..]  &  p)  ==  0)  a.Add(m[#]);
// цикл разделения
   else b.Add(m[#]);
   p >>= 1;
   List<uint> a1 = RadixSort(a, p);
   List<uint> b1 = RadixSort(b, p);
   a1.Add(b1[..]); // цикл соединения
   return a1;
}
```

Для облегчения восприятия α-операторов достаточно выделять их аргументы-перечислители цветом в редакторе кода.

Заключение

Достоинствами предложенной синтаксической конструкции являются:

- относительно легкая для восприятия семантика;
- замена циклов выражениями, что уменьшает размер кода;
- стимуляция программиста к высокоуровневому представлению обработки коллекций, что отодвигает манипуляцию индексами на второй план.

Недостатком является снижение скорости обработки коллекций при использовании универсальных итераторов. Эффективным решением являются преобразования каждого аргумента-перечислителя в итеративный цикл без использования анонимных сущностей. Цена такого эффекта — повышение времени оптимизации и генерации кода компилятором.

Предложенная в статье идея может быть использована для построения аргументов-перечис-

лителей узлов бинарных деревьев. Параметр этих перечислителей описывается числом, двоичные разряды которого являются рангами ссылок, начиная от корня дерева. Для деревьев арности k рангами являются показатели степеней полинома в k-й системе счисления, который определяет положение узла в дереве.

Предметом будущих исследований является использование *композиции* аргументов-перечислителей. Например, при сортировке подсчетом коллекции M можно использовать α-оператор $Q[M[..]] + 1$, который заполняет массив Q значениями счетчиков элементов коллекции M. Вместе с тем конструкцию «цикл в цикле» с тремя уровнями вложенности можно описать выражением, например [[5..9]; + [1:10].. [100..200]].

Описанный подход может быть использован разработчиками трансляторов ЯВУ и препроцессоров.

Список литературы

1. **Кнут Д. Э.** Искусство программирования для ЭВМ. Том 1. Основные алгоритмы: пер. с англ. М.: Мир, 1976. 735 с.
2. **Себеста Р. У.** Основные концепции языков программирования. 5-е изд.: пер. с англ. М.: Вильямс, 2001. 672 с.
3. **Пратт Т., Зелковиц М.** Языки программирования. Разработка и реализация / под общ. ред. А. Матросова: пер. с англ. СПб.: Питер, 2002. 688 с.
4. **Пирс Б.** Типы в языках программирования: пер. с англ. М.: Лямбда пресс: Добросвет, 2011. 656 с.
5. **Филд А., Харрисон П.** Функциональное программирование: пер. с англ. М.: Мир, 1993. 637 с.
6. **Python** 3.11.2. Documentation. The Python Language Reference. URL: https://docs.python.org/3/reference/expressions.html (дата обращения 06.02.2023).
7. **Бен-Ари М.** Языки программирования. Практический сравнительный анализ: пер. с англ. М.: Мир, 2000. 366 с.
8. **Мартин Р.** Чистый код: создание, анализ и рефакторинг: пер. с англ. СПб.: Питер, 2013. 464 с.
9. **Солтер Н. А., Клепер С. Дж.** C++ для профессионалов: пер. с англ. М.: Вильямс, 2006. 912 с.
10. **Watt S. M.** A Technique for Generic Iteration and Its Optimization // Computer Science. 2006. URL: https://www.csd.uwo.ca/~watt/pub/reprints/2006-wgp-jflow.pdf (дата обращения 06.02.2023).
11. **A Gentle** Introduction to Haskell. URL: https://www.haskell.org/tutorial/goodies.html (дата обращения 06.02.2023).
12. **Стерлинг Л., Шапиро Э.** Искусство программирования на языке Пролог: пер. с англ. М.: Мир, 1990. 235 с.
13. **Visual** Prolog. Fundamental Prolog Part 2. URL: https://wiki.visual-prolog.com/index.php?title=Fundamental_Prolog_Part_2#Functors_and_Predicates (дата обращения 6.02.2023).
14. **Dyalog.** The tool of thought for software solutions. URL: https://www.dyalog.com/aplx.htm (дата обращения 06.02.2023).
15. **Python** 3.11.2. Documentation. The Python Standard Library. URL: https://docs.python.org/3/library/stdtypes.html (дата обращения 06.02.2023).
16. **Марков В. Н.** Современное логическое программирование на языке Visual Prolog 7.5: уч. СПб: БХВ-Петербург, 2016 544 с.
17. **Microsoft**.NET. C# documentation. Iterators. URL: https://learn.microsoft.com/en-us/dotnet/csharp/iterators (дата обращения 06.02.2023).
18. **Microsoft**.NET. C# documentation. Using Properties. URL: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-properties (дата обращения 06.02.2023).

Enumerator-Arguments as a Language Tool for Defining Loops

V. N. Markov, D. Sc. (Engineering), Professor, vinitar@yandex.ru,
Kuban State Technological University, Krasnodar, 350000, Krasnodar region, Russian Federation

Corresponding author:

Vitaly N. Markov, D. Sc. (Engineering), Professor, Kuban State Technological University, Krasnodar, 350000, Krasnodar Region, Russian Federation
E-mail: vinitar@yandex.ru

Received on February 13, 2023
Accepted on February 27, 2023

The new syntactic construction for defining processing loops of collections and series of values in the form of expressions is proposed as part of the development of new programming languages and the improvement of existing languages. Its essence is to use index enumerators as arguments to operators and collection processing functions. Such enumerator-arguments allow you to express operators and functions for processing a slice/entire collection.

Translational semantics of enumerator-arguments for indexed and referenced collections is given. A number of examples of using enumerator-arguments for comparison with higher-order functions and list comprehension operators are shown. It is concluded that enumerator-arguments surpass second-order functions in conciseness and readability of the source code, and are not inferior to list comprehensions operators.

Further research is aimed at constructing enumerator-arguments of binary trees nodes using a parameter that is expressed as a number. Its binary digits are link ranks and uniquely position a node in the tree.

Keywords**:** *enumerator, iterator, indexator, readability of source code, conciseness of source code, slice of collection, higher-order function, list comprehension*

For citation:

Markov V. N. Enumerator-Arguments as a Language tool for Defining Loops, *Programmnaya ingeneria,* 2023, vol. 14, no. 5 pp. 232—244. DOI: 10.17587/prin.14.232-244.

References

1. **Knuth D. E.** *The Art of Computer Programming,* vol. 1: Fundamental Algorithms, 3rd ed. Addison-Wesley, 1997, 650 p.

2. **Sebesta R. W.** *Concepts of Programming Languages,* 5th ed., Addison-Wesley, 2001, 720 p.

3. **Pratt T. W., Zelkowitz M. V.** *Programming Languages: Design and Implementation,* 4th ed., Pearson, 2000, 672 p.

4. **Pierce B. C.** *Types and Programming Languages,* MIT, 2002, 648 p.

5. **Field A. J., Harrison P. G.** *Functional Programming*, Addison-Wesley, 1988, 616 p.

6. **Python** 3.11.2. Documentation. The Python Language Reference, available at: https://docs.python.org/3/reference/expressions.html (date of access 06.02.2023).

7. **Ben-Ari M.** *Understanding programming Languages*,Wiley, 1996, 376 p.

8. **Martin R. C.** *Clean Code: A Handbook of Agile Software Craftsmanship*, Pearson, 2008, 464 p.

9. **Solter N. A., Kleper S. J.** *Professional C++,* Wrox, 2005, 864 p.

10. **Watt S. M.** A Technique for Generic Iteration and Its Optimization. *Computer Science,* 2006, available at: https://www.csd.uwo.ca/~watt/pub/reprints/2006-wgp-jflow.pdf. (date of access 06.02.2023).

11. **A Gentle** Introduction to Haskell, available at: https://www.haskell.org/tutorial/goodies.html (date of access 06.02.2023).

12. **Sterling L., Shapiro E.** *The Art of Prolog: Advanced Programming Techniques*, MIT, 1986, 437 p.

13. **Visual** Prolog. Fundamental Prolog Part 2, available at: https://wiki.visual-prolog.com/index.php?title=Fundamental_Prolog_Part_2#Functors_and_Predicates (date of access 06.02.2023).

14. **Dyalog.** The tool of thought for software solutions, available at: https://www.dyalog.com/aplx.htm (date of access 06.02.2023).

15. **Python 3.11.2**. Documentation. The Python Standard Library, available at: https://docs.python.org/3/library/stdtypes.html date of access 06.02.2023).

16. **Markov V. N.** *Modern logic programming in Visual Prolog 7.5*: textbook, SPb, BHV-Peterburg, 2016, 544 p. (in Russian).

17. **Microsoft.**NET. C# documentation. Iterators, available at: https://learn.microsoft.com/ru-ru/dotnet/csharp/iterators (date of access 06.02.2023).

18. **Microsoft.**NET. C# documentation. Using Properties, available at: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/using-properties (date of access 06.02.2023).

ИНФОРМАЦИЯ

УДК 004.056

DOI: 10.17587/prin.14.245-253

Д. И. Назаров, исследователь безопасности, dmitry.nazarov@solidwall.io,
ООО «Солидсофт», Москва,
Д. А. Сигалов, мл. науч. сотр. кафедры информационной безопасности,
asterite@seclab.cs.msu.ru,
Д. Ю. Гамаюнов, доц. кафедры информационной безопасности,
gamajun@seclab.cs.msu.su,
Московский государственный университет имени М. В. Ломоносова

Поиск информации о принимаемых сервером запросах в закомментированном клиентском коде веб-приложений

Поступила в редакцию 04.03.2023
Принята к публикации 22.03.2023

Рассмотрена задача обнаружения информации о принимаемых сервером запросах путем ее извлечения из закомментированного клиентского кода веб-приложений. Предложен и программно реализован алгоритм, извлекающий закомментированные вызовы. Проведен эксперимент более чем на миллионе страниц в сети Интернет, относящихся более чем к 50 тыс. веб-приложений из списка Alexa Top 1 Million. По его результатам было выяснено, что закомментированные вызовы действительно встречаются в реальных веб-приложениях. В данном случае они были обнаружены примерно в 2,78 % из всех исследованных веб-приложений. Кроме того, около 40 % из них были отмечены как «живые», т. е. имеющие конечную точку на сервере. Проведенный беглый анализ также показал, что такие запросы нередко оказываются уязвимыми.

Ключевые слова: *информационная безопасность, веб-приложение, статический анализ, поиск уязвимостей, HTTP-запрос, закомментированный код*

Для цитирования:

Назаров Д. И., Сигалов Д. А., Гамаюнов Д. Ю. Поиск информации о принимаемых сервером запросах в закомментированном клиентском коде веб-приложений // Программная инженерия. 2023. Том 14, № 5. С. 245—253. DOI: 10.17587/prin.14.245-253.

Введение

Задача обнаружения программных уязвимостей в веб-приложениях на основе модели «черного ящика», т. е. когда программный код серверной стороны приложения недоступен, и пользователь может взаимодействовать с ней, только отправляя запросы и анализируя ответы, — это одна из наиболее актуальных задач анализа защищенности современных информационных систем. Типовое современное веб-приложение (рис. 1) представляет собой распределенную программу из клиентской части на языке JavaScript и серверной части на произвольном языке программирования, которые взаимодействуют между собой по протоколу HTTP и производным от него.

В настоящее время JavaScript является самым распространенным языком для клиентской части веб-приложений, а его поддержка встроена во все современные браузеры. В рассматриваемой модели «черного ящика» исходный код клиентской части приложения всегда доступен для анализа, так как он всегда передается в браузер клиента для последующего исполнения, возможно, в минифицированном и обфусцированном (т. е. преобразованным способом, сохраняющим функциональность программы, но затрудняющим ее чтение и анализ) виде. Некоторые виды уязвимостей веб-приложений можно обнаружить на основе анализа только клиентской части приложения, например, утечки пользовательских данных [1] или уязвимости класса DOM Based XSS [2]. Кроме того, анализ

Рис. 1. Типовое современное веб-приложение

клиентского кода веб-приложения позволяет извлечь информацию о структуре и о возможных значениях HTTP-запросов, отправляемых на сервер с клиентской стороны веб-приложения [3]. Эта информация позволяет найти серверные точки входа и некоторые начальные приближения входных данных для динамического анализа серверной стороны приложения методом «черного ящика». Целью при этом является поиск уязвимостей уже на серверной стороне веб-приложения, например, таких как SQL Injection, Reflected XSS и др.

Любое улучшение методов извлечения информации о серверных точках входа в клиентской части повышает вероятность обнаружения уязвимостей в серверной части. В работе, результаты которой представлены в настоящей статье, исследована гипотеза о том, что закомментированный JavaScript-код клиентской части веб-приложения может содержать полезную информацию о серверных точках входа, недоступную в «живом» коде клиентской части. В языке JavaScript, как и в любом другом современном языке программирования, есть возможность оставлять комментарии. Как правило, комментарии используют для написания пояснений к исходному тексту программы, но это не всегда так. Нередко программисты неиспользуемую или устаревшую со временем часть кода вместо удаления прячут внутри блока комментариев. Поскольку клиентский код и серверный код исполняются изолированно друг от друга, то возможна такая ситуация, что часть клиентского кода окажется закомментированной, при этом с сервера соответствующая данному коду функциональная возможность удалена не будет. Предложенный в данной работе подход к извлечению информации о серверных точках входа с помощью анализа закомментированного клиентского кода является новым. Авторы не обнаружили работ, которые исследовали бы такую задачу. По этой причине, кроме реализации алгоритма анализа закомментированного кода, в рамках данного исследования авторы провели эксперименты для оценки распространенности закомментированного JavaScript-кода в реальных веб-приложениях, а также сделали анализ обнаруженных в них серверных точек входа, в том числе на наличие уязвимостей.

Основная часть исследования

В процессе изучения инструментальных средств были рассмотрены такие часто упоминаемые в литературе статические анализаторы, как TAJS [4], SAFE [5] или WALA [6]. Ни в одном из них не было найдено возможностей для анализа комментариев.

Далее был рассмотрен статический анализатор [7], который в данный момент находится в стадии разработки. В отличие от перечисленных выше, он выявляет точки взаимодействия с сервером путем анализа клиентского JavaScript-кода. Однако в нем также отсутствует возможность получения закомментированных вызовов.

Следует отметить, что анализ комментариев является достаточно сложным направлением по причине того, что сами комментарии в большей

```
var isGuest =true;

function goBackOnly()
{
    googleActionEvent('Eski Sürüm','Eskiye Dön', 'V1 e geçti');
    setCookie('backtothefuture', '2', 365);
    window.location.reload();

    //$.ajax({
    //    type: "POST",
    //    url: "/data/ReturnBackDeleteCookie",
    //    success: function (response) {
    //        if(response)
    //        {
    //            window.location.reload();
    //        }
    //    }
    //});
}

function readResponse(form, response) {
    form.find('.uyari').hide();
    if (response.HasError) {
        if (response.Message) {
            form.find('.uyari.kirmizi').html('<strong>Hata! </strong>' + response.Message).
```

Рис. 2. Пример закомментированного кода с сайта https://www.donanimhaber.com/

степени представляют собой текстовые блоки на естественном языке, не имеющие обязательного формата помимо синтаксических разделителей. Из этого можно сделать вывод, что все алгоритмические решения данной задачи будут носить эвристический характер. На рис. 2 и 3 представлены реальные примеры закомментированных вызовов, встречающиеся в сети Интернет.

Описание алгоритма

Программная реализация алгоритма для решения поставленной задачи не может быть использована как самостоятельное инструментальное средство. Для обработки полученных с его помощью результатов необходимо его интегрировать со статическим анализатором. Из рассмотренных

```
        } else {
            alert('Не добавлено изображение или фото оттиска штампа!');
        }
    }

    /* function carouselAjax(type, count) {
        $.post(
                "/ajax.php",
                {
                    method: 'carouselajax',
                    type: type,
                    count: count
                },
                    function(data) {
                        // alert(data);
                        $('#' + type + '_carousel .slidered').append(data);

                        $("#ooo_carousel").jCarouselLite('reload', {
                            animation: 'slow'
                        });
                }
        );
    } */
```

Рис. 3. Пример закомментированного кода с сайта https://master-stampov.ru/assets/js/scripts.js?v=78

```
1: function ExtractCommentedCode(Scripts)
2:     ScriptsAST ← []
3:     for all script s ∈ Scripts do
4:         s = RemoveBlankLines(s)
5:         mergedComments = MergeComments(s)
6:         for all blockComment ∈ mergedComments do
7:             ScriptsAST = ScriptsAST ∪ ParseComment(blockComment)
8:     StatAnalyzer(ScriptsAST)
9: function ParseComment(BlockComment)
10:    while True do
11:        ast ← ParseCode(BlockComment)
12:        if ast is nil then
13:            textComment ← DeleteBadSymbol(BlockComment)
14:        else
15:            return ast
16:    return nil
```

Рис. 4. Псевдокод алгоритма извлечения закомментированного кода

инструментальных механизмов для интеграции был выбран статический анализатор [7] по причине наличия в нем поиска точек взаимодействия с сервером.

На вход алгоритм (рис. 4) получает текст JavaScript-кода страницы. На первом этапе происходит удаление пустых строк. Они не влияют на синтаксическую корректность, но некоторые разработчики добавляют их для повышения читаемости кода. Поскольку на одном из следующих этапов необходимо будет объединить близлежащие однострочные комментарии в блочные, то пустые строки, встречающиеся между ними, могут повлиять на синтаксическую корректность уже объединенных блоков. Для наглядности приведен реальный пример (рис. 5), который показывает, что без удаления пустых строк после объединения будет получено несколько некор-

```
// $(window).on('scroll', function () {
//  if ($('.load-more').offset().top <= $(window).scrollTop() + $(window).height()) {
//     console.log('bottom');
//     $.ajax({
//       type: "GET",
//       url: 'http://www.tmpo.co/ajax?tipe=beritaterkini&limit=10&start=10',
//       success: function (result) {
//         $(".divContent").append(result);

//         sIndex = sIndex + offSet;
//         isPreviousEventComplete = true;

//         if (result == '') //When data is not available
//             isDataAvailable = false;

//         $(".LoaderImage").css("display", "none");
//       },
//       error: function (error) {
//           alert(error);
//       }
//     });
// }
// });
```

Рис. 5. Пример кода с разделением однострочных комментариев пустыми строками. Если не сгруппировать код в один блок, то нельзя получить синтаксически верную конструкцию, содержащую запрос

ректных блоков с JavaScript-кодом вместо одного корректного.

На втором этапе происходит синтаксический разбор кода страницы с помощью библиотеки Babel Parser, результатом которого является AST-дерево. Из полученного дерева извлекается массив закомментированных строк и блоков. После чего этот массив подается на вход функции, которая объединяет находящиеся на соседних строках комментарии в блоки.

После слияния начинается этап парсинга. Аналогично второму этапу, с помощью Babel Parser проводится синтаксический разбор каждого из полученных блоков. Если парсер сообщает об ошибке, то необходимо удалить вызывающий ее символ и повторить этап заново. Если по результатам некоторого числа итераций удается представить комментарий в виде AST-дерева, он будет добавлен в область для последующего анализа.

Эксперимент

После успешной интеграции модуля с анализатором был проведен эксперимент на ресурсах из базы данных, которая основана на 24212 наиболее и 26072 наименее популярных сайтах из списка Alexa Top 1 Million. Сама база была собрана с помощью программы, осуществляющей автоматический обход страниц веб-приложения по ссылкам (*web crawler*), написанной на основе библиотеки Go-Golly. Для каждого ресурса в базе данных хранится его уникальный идентификатор (порядковый номер), реальный URL и содержимое. Также в базе данных хранится информация о связях между HTML-страницами и подключаемыми ими JavaScript- и CSS-файлами.

Для каждой страницы формировался архив в формате TAR, содержащий ее и все подключенные к ней ресурсы. Анализатор запускался на локальной копии страницы в двух режимах: с включенной опцией анализа закомментированного кода и без него. Результаты каждого из запусков сохранялись независимо друг от друга.

Следующим этапом необходимо было выделить запросы, являющиеся уникальными для закомментированного кода. Сама по себе задача, какие запросы считать уникальными, является нетривиальной. Ведь, с одной стороны, значения параметров могут находиться в компоненте path URL [8, 9]. В таком случае, пара запросов, отличающихся значением этого параметра, а соответственно, и URL, скорее всего, будут обработаны сервером одинаково. С другой стороны, возможно использование параметров query string, отвечающих за маршрутизацию в приложении. По этой причине даже идентичные запросы, имеющие разные значения такого параметра, могут обрабатываться сервером по-разному. Исходя из этого считаем одинаковыми запросы, у которых совпадают методы, path-компоненты URL, наборы параметров в query string и/или в теле запроса, значения заголовков Content-Type и Host и значения параметров. Если значения одноименных параметров расходятся, но относятся к числовому типу, то такие запросы также считаются одинаковыми.

Далее было проведено сравнение результатов двух запусков анализатора по следующему алгоритму. Для каждого запроса с пометкой о том, что он был найден в закомментированном коде, проводился поиск аналогичного запроса из результатов запуска анализатора без включенного модуля на той же странице. Если соответствующий запрос не был найден, то закомментированный вызов считался уникальным и попадал в финальную выборку.

Следующим шагом выполнялась проверка существования соответствующих конечных точек на сервере для найденных запросов с использованием описанной далее процедуры. Все закомментированные вызовы дедуплицировались по описанному ранее алгоритму уже между собой, после чего подсчитывалось общее число уникальных вызовов. Затем по каждому из них формировался и отправлялся на сервер запрос. Сразу после этого отправлялся аналогичный запрос на новый адрес, который получался добавлением в компонент path оригинального URL строки, сгенерированной из случайных символов. Полученный таким образом адрес, скорее всего, обрабатывался сервером как несуществующий. Далее сравнивались коды ответов. Если они одинаковы, то с большой вероятностью проверяемая конечная точка уже отключена. Если коды ответов разные, то, скорее всего, с сервера еще не была удалена соответствующая функциональная возможность. В свою очередь, код ответа может означать либо полностью найденный корректный запрос к серверу, либо наличие некоторых значений параметров, которые анализатором не были найдены или были найдены некорректно. Вне зависимости от кода ответа в дальнейшем найденную конечную точку можно анализировать на предмет уязвимостей.

На последнем этапе был выполнен сбор статистики по категориям точности среди всех найденных уникальных запросов, а также оценено процентное соотношение успешных запросов ко всем. Таким образом, было выделено три категории точности и полноты запросов. Первая катего-

рия — полностью известные: в нее входят запросы, у которых анализатор смог определить URL и все параметры с их значениями. Вторая категория включает в себя запросы, у которых полностью известны URL и имена параметров, но какие-то значения параметров оказались не определены. К третьей категории были отнесены все остальные запросы.

Результаты

Было проведено два эксперимента:

- на полной выборке;
- только на наименее популярных сайтах (исходя из предположения о том, что они имеют большее количество самописного кода, а значит, возрастает вероятность увидеть закомментированные запросы).

В общей сложности было проанализировано более миллиона страниц.

Полная выборка (случайные страницы из всего списка)

- Всего проанализировано 580 056 страниц из базы данных, относящихся к 50 291 веб-приложению.
- AJAX-вызовы, встречающиеся только в закомментированном коде, оказались на 9948 страницах (около 1,72 % от общего количества) или в 1396 веб-приложениях (около 2,78 % от общего количества).
- Всего найдено 14 395 закомментированных вызовов. Из них оказалось 3923 уникальных (около 27,25 %).
- Соответствующие конечные точки на сервере имели 1536 вызовов (около 39,15 % среди уникальных), относящиеся к 1232 ресурсам (около 0,21 % от общего количества) или 739 различным веб-приложениям (около 1,47 % от общего количества).

На диаграммах (рис. 6 и 7) можно увидеть сравнительную статистику по кодам ответов для всех уникальных закомментированных запросов и запросов, которые были сочтены успешными, т. е. имеющими конечную точку на сервере, исходя из предположений, описанных в предыдущем разделе.

Как видно на графиках, большая часть успешных запросов имеет коды ответов, равные 200, 400, 403, 500, а процент ответов с кодом 404 минимален, что свидетельствует о верности выдвинутых предположений для проверки конечной точки на сервере.

Статистика по запросам, посчитанных успешными, в зависимости от того, насколько точно анализатору удалось определить, как выглядит отправляемый запрос:

- среди 1961 полностью известного запроса 728 имеют конечную точку на сервере, что соответствует 37,12 %.
- из 1682 запросов с неизвестными значениями параметров 773 были определены как успешные, что составляет 45,96 %.
- запросов с неизвестными именами параметров или частью URL было найдено 280, среди которых 35 (12,5 %) имеют соответствующую функциональность на сервере.

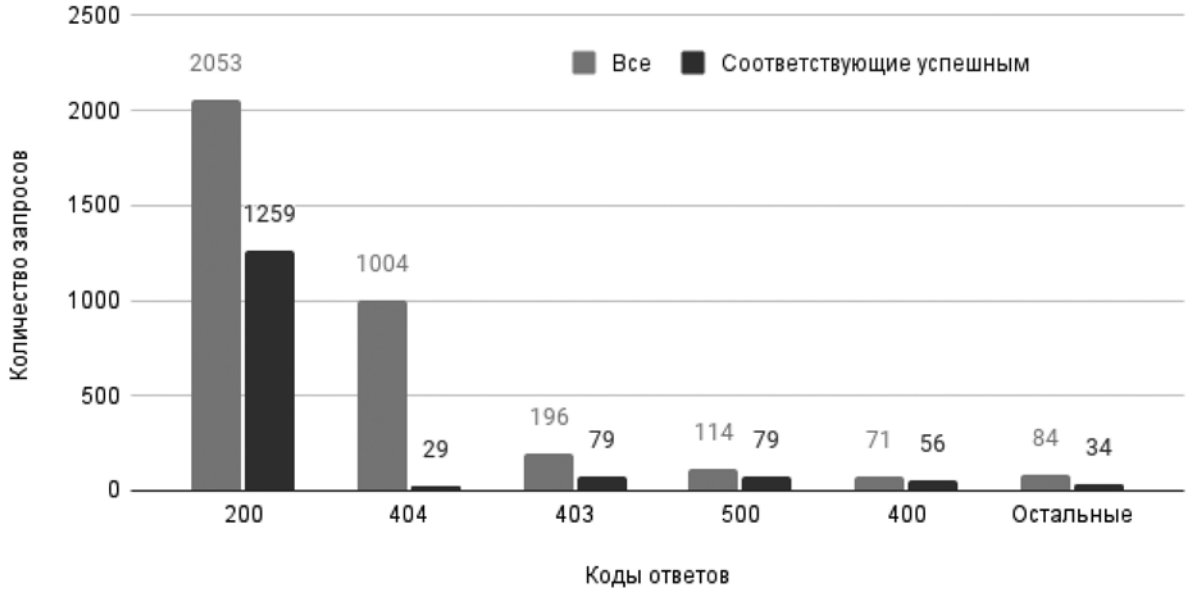


Рис. 6. Коды ответов для полной выборки (успешные — имеющие отличные от случайных запросов коды ответов)

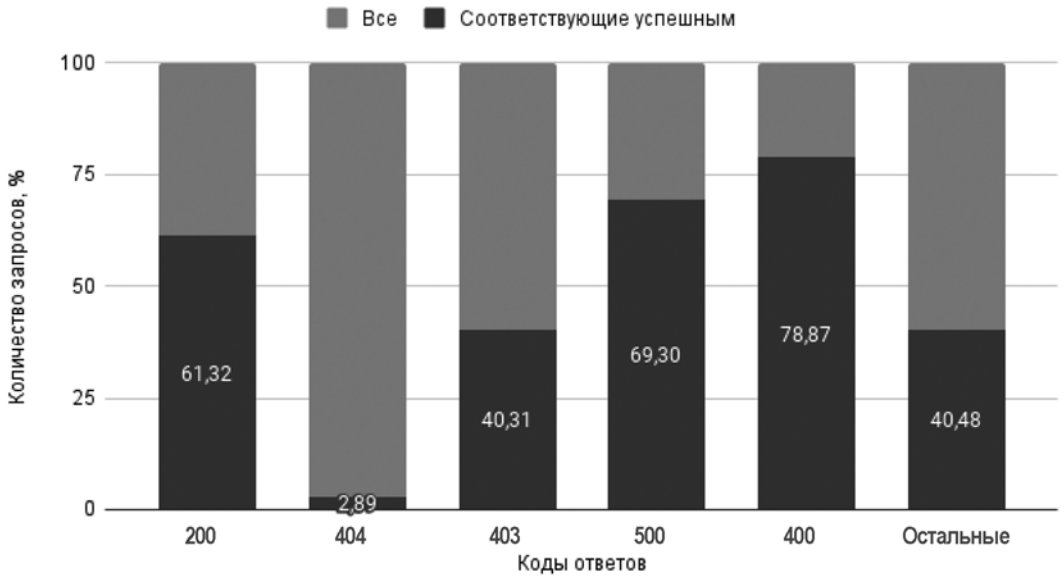


Рис. 7. Процентное соотношение кодов ответа для полной выборки

Наименее популярные сайты

- Всего проанализирована 568 421 страница из базы данных, относящаяся к 26 072 веб-приложениям.
- AJAX-вызовы, встречающиеся только в закомментированном коде, оказались на 7112 страницах (1,25 % от общего количества) или в 477 веб-приложениях (1,83 % от общего количества).
- Всего найдено 9202 закомментированных вызова. Из них оказалось 1646 уникальных (17,89 %).
- Соответствующие конечные точки на сервере имели 570 вызовов (34,63 % среди уникальных), относящиеся к 500 ресурсам (0,09 % от общего количества) или 282 различным веб-приложениям (1,08 % от общего количества).

Далее представлена аналогичная полному запуску статистика по кодам ответов и конкретности запросов (рис. 8 и 9).

Точность результатов анализатора для найденных успешных запросов:

- среди 833 полностью известных запросов 234 имеют конечную точку на сервере, что соответствует 28,09 %;
- из 646 запросов с неизвестными значениями параметров 326 были определены как успешные, что составляет 50,46 %;

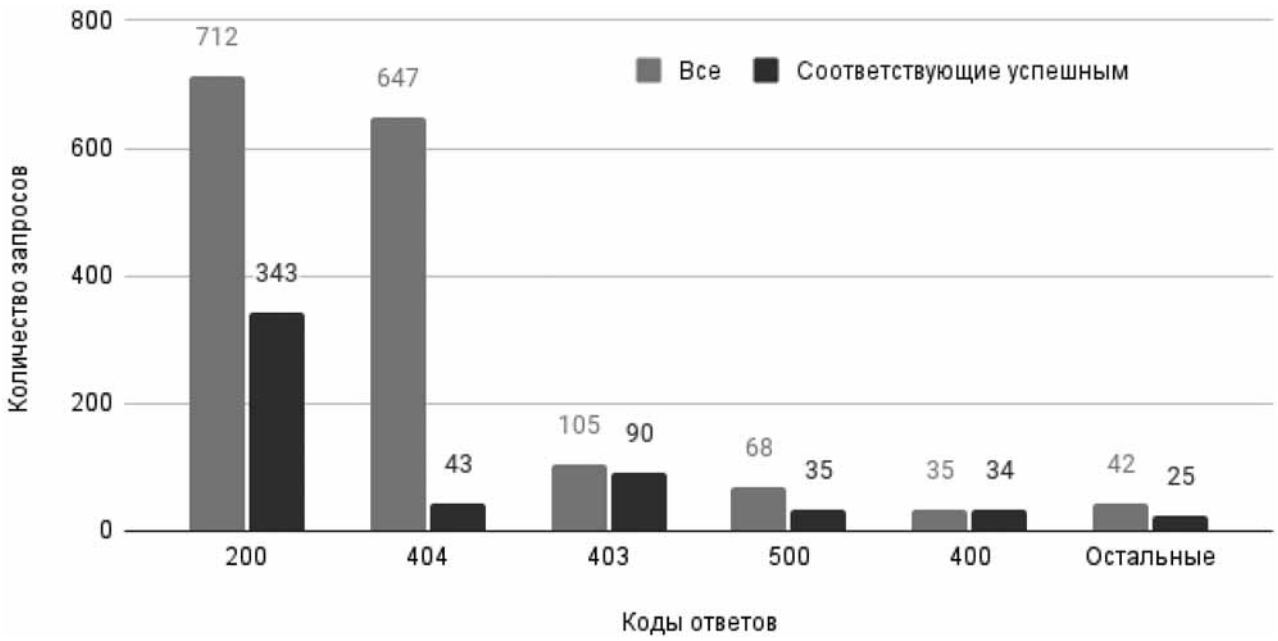


Рис. 8. Коды ответов для выборки из наименее популярных сайтов

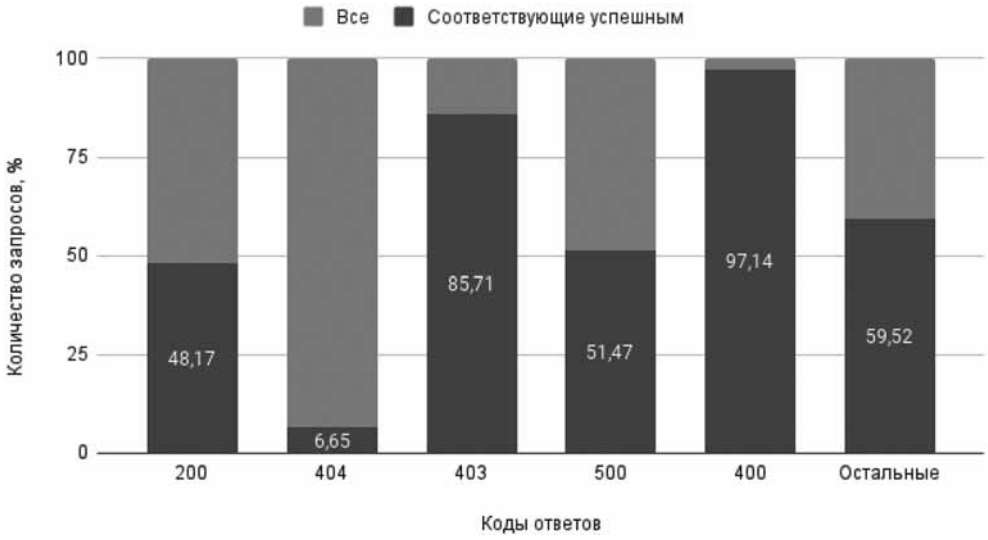


Рис. 9. Процентное соотношение кодов ответа для выборки из наименее популярных сайтов

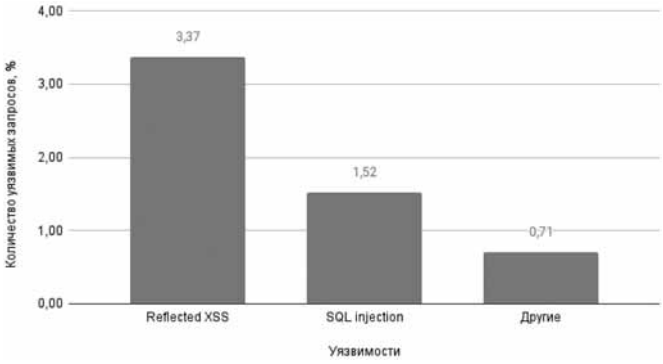


Рис. 10. Процентное соотношение уязвимых запросов ко всем закомментированным, имеющим конечную точку на сервере

• запросов с неизвестными именами параметров или частью URL было найдено 167, среди которых 10 (5,99 %) имеют соответствующую функциональность на сервере.

Уязвимости

Закомментированные запросы с оставленной функциональностью на сервере были исследованы на наличие уязвимостей. Из них 5,6 % оказались уязвимы. Самые часто встречающиеся виды уязвимостей — Reflected XSS и SQL Injection. На рис. 10 представлена диаграмма, демонстрирующая процентное соотношение найденных уязвимостей конкретного типа по отношению ко всем «живым» (имеющим работающую конечную точку на сервере) закомментированным запросам.

Заключение

Решена задача обнаружения информации о принимаемых сервером запросах путем ее извлечения из закомментированного клиентского кода веб-приложений на языке JavaScript, реализован алгоритм, извлекающий закомментированные вызовы функций, отправляющих HTTP-запросы на сервер. Экспериментальное исследование на приложениях из рейтинга Alexa Top 1 Million позволяет сделать вывод, что закомментированные запросы с оставленной функциональностью на сервере действительно встречаются в реальном мире. В проведенном эксперименте они нашлись примерно в 2,78 % от исследованных веб-приложений, более того, около 40 % из них имеют соответствующую «живую» точку входа на сервере. Выборочный ручной анализ показал, что подобные запросы нередко оказываются уязвимыми: в «забытых» серверных API были найдены такие уязвимости, как Reflected XSS, SQL Injection, PHP code injection и др. Также следует отметить, что подобный анализ может быть реализован только с помощью статического анализа клиентского кода приложения, так как для инструментов динамического анализа закомментированный программный код является недоступным.

Разработанный модуль можно использовать в составе сканера веб-безопасности для получения более полной информации о серверной части веб-приложения методом «черного ящика».

Список литературы

1. **Hedin D., Sabelfeld A.** Information-Flow Security for a Core of JavaScript // IEEE 25th Computer Security Foundations Symposium. 2012. P. 3—18. DOI: 10.1109/CSF.2012.19.

2. **OWASP** Foundation. DOM Based XSS. URL: https://owasp.org/www-community/attacks/DOM_Based_XSS (дата обращения 02.03.2023).

3. **Guha A., Krishnamurthi S., Jim T.** Using static analysis for Ajax intrusion detection // Proceedings of the 18th international conference on World wide web. 2009. P. 561—570. DOI: 10.1145/1526709.1526785.

4. **Jensen S. H., Møller A., Thiemann P.** Type Analysis for JavaScript // International Static Analysis Symposium. 2009. P. 238—255. DOI: 10.1007/978-3-642-03237-0_17.
5. **Lee H., Won S., Jin J., Cho J., Ryu S.** Safe: Formal specification and implementation of a scalable analysis framework for ecmascript // In International Workshop on Foundations of Object-Oriented Languages. 2012. URL: https://junheecho.com/assets/papers/fool12.pdf (дата обращения 02.03.2023).
6. **The T. J. Watson** Libraries for Analysis (WALA). URL: http://wala.sourceforge.net/ (дата обращения 02.03.2023).
7. **Сигалов Д. А., Хашаев А. А., Гамаюнов Д. Ю.** Обнаружение серверных точек взаимодействия в веб-приложениях на основе анализа клиентского JavaScript-кода // Прикладная дискретная математика. 2021. № 53. С 32—54. DOI: 10.17223/20710410/53/3.
8. **Berners-Lee T., Fielding R., Masinter L.** Uniform Resource Identifier (URI): Generic Syntax, document RFC 3986. 2005. URL: https://www.ietf.org/rfc/rfc3986.txt (дата обращения 02.03.2023).
9. **The URL** standard. URL: https://url.spec.whatwg.org/ (дата обращения 02.03.2023).

Mining Server HTTP Endpoints from Commented-Out Client-Side Code of Web Applications

D. I. Nazarov, Security Researcher, dmitry.nazarov@solidwall.io,
LTD "SolidSoft", Moscow, 117312, Russian Federation,
D. A. Sigalov, Junior Researcher, asterite@seclab.cs.msu.ru,
D. Yu. Gamayunov, Associate Professor, gamajun@seclab.cs.msu.su,
Lomonosov Moscow State University, Moscow, 119991, Russian Federation

Corresponding author:

Dmitry I. Nazarov, Security Researcher, LTD "SolidSoft",
Moscow, 117312, Russian Federation
E-mail: dmitry.nazarov@solidwall.io

Received on March 04, 2023
Accepted on March 22, 2023

In this paper we consider the problem of detecting information about HTTP endpoints on the server by extracting requests from the commented-out client code of web applications. The algorithm that extracts commented-out requests is proposed and implemented. The developed module was integrated into a static analyzer. An experiment was conducted on more than a million web pages belonging to more than 50 thousand web applications from the Alexa Top 1 million list. Requests unique to the commented code were selected for each page. Then a check was made for the existence of an associated endpoint on the server for each of them. According to the results of the experiment, it was found that commented-out requests actually occur in real web applications, they were detected in ~2.78 % of all explored sites. In addition, ~40 % of them were marked as "live", that is having an endpoint on the server. Also, a cursory analysis has shown that such endpoints often turn out to be vulnerable. The module can be used as part of a web security scanner to obtain more complete information about the server side of a web application using the black box method.

Keywords: *information security, web application, static analysis, search for vulnerabilities, HTTP-request, commented-out code.*

For citation:

Nazarov D. I., Sigalov D. A., Gamayunov D. Yu. Mining Server HTTP Endpoints from Commented-Out Client-Side Code of Web Applications, *Programmnaya Ingeneria,* 2023, vol. 14, no. 5, pp. 245—253. DOI: 10.17587/prin.14.245-253.

References

1. **Hedin D., Sabelfeld A.** Information-Flow Security for a Core of JavaScript, *IEEE 25th Computer Security Foundations Symposium,* 2012, pp. 3—18. DOI: 10.1109/CSF.2012.19.
2. **OWASP** Foundation. DOM Based XSS, available at: https://owasp.org/www-community/attacks/DOM_Based_XSS (date of access 02.03.2023).
3. **Guha A., Krishnamurthi S., Jim T.** Using static analysis for Ajax intrusion detection, *Proceedings of the 18th international conference on World wide web,* 2009, pp. 561—570. DOI: 10.1145/1526709.1526785.
4. **Jensen S. H., Møller A., Thiemann P.** Type Analysis for JavaScript, *International Static Analysis Symposium,* 2009, pp. 238—255. DOI: 10.1007/978-3-642-03237-0_17.
5. **Lee H., Won S., Jin J., Cho J., Ryu S.** Safe: Formal specification and implementation of a scalable analysis framework for ecmascript, *International Workshop on Foundations of Object-Oriented Languages,* 2012, available at: https://junheecho.com/assets/papers/fool12.pdf (date of access 02.03.2023).
6. **The T. J. Watson** Libraries for Analysis (WALA), available at: http://wala.sourceforge.net/ (date of access 02.03.2023).
7. **Sigalov D. A., Khashaev A. A., Gamayunov D. Yu.** Detecting server-side endpoints in web applications based on static analysis of client-side JavaScript code, *Prikladnaya Diskretnaya Matematika,* 2021, no. 53, pp. 32—54. DOI: 10.17223/20710410/53/3 (in Russian).
8. **Berners-Lee T., Fielding R., Masinter L.** Uniform Resource Identifier (URI): Generic Syntax, document RFC 3986. 2005. available at: https://www.ietf.org/rfc/rfc3986.txt (date of access 02.03.2023).
9. **The URL** standard, available at: https://url.spec.whatwg.org/ (date of access 02.03.2023).

УДК 004.42
DOI: 10.17587/prin.14.254-260

Н. А. Бабич, аспирант, nickware@mail.ru,
Институт проблем машиноведения Российской академии наук, Санкт-Петербург

Программная платформа для чтения, обработки и анализа данных ЭЭГ

Поступила в редакцию 14.03.2023
Принята к публикации 04.04.2023

Данные электроэнцефалограмм (ЭЭГ) могут быть использованы во множестве различных областей: для диагностики заболеваний головного мозга, в нейромашинных интерфейсах, для проведения различных исследований и др. Для применения данных ЭЭГ необходим большой набор различных алгоритмов предобработки и анализа этих данных. В настоящей статье описана программная платформа, содержащая набор средств для автоматизированной обработки сигналов ЭЭГ и их анализа, в том числе методами машинного обучения. Платформа имеет гибкую архитектуру и состоит из модулей, что позволяет применять ее при исследовании данных ЭЭГ для различных целей. Данные могут быть получены как из файлов, так и напрямую от электроэнцефалографа в режиме реального времени. Графический интерфейс предоставляет удобный способ конфигурирования модулей платформы. Программный интерфейс клиентских приложений дает возможность применения этой платформы для создания прототипов устройств, использующих для своей работы данные ЭЭГ.

Ключевые слова: *анализ сигналов, анализ ЭЭГ, машинное обучение, индикаторы ритмов мозга, фильтрация сигналов, ритмы мозга, программная платформа*

Для цитирования:

Бабич Н. А. Программная платформа для чтения, обработки и анализа данных ЭЭГ // Программная инженерия. 2023. Том 14, № 5. С. 254—260. DOI: 10.17587/prin.14.254-260.

Введение

Электроэнцефалография — метод записи электрограммы спонтанной электрической активности головного мозга [1, 2]. Биосигналы, обнаруженные с помощью электроэнцефалографии, представляют собой постсинаптические потенциалы пирамидных нейронов (нейронов, вызывающих возбуждение активности мозга млекопитающих) в коре головного мозга. Электроэнцефалография является неинвазивным методом, электроды размещаются вдоль кожи головы с использованием международной системы «10-20» или ее вариантов. Клиническая интерпретация записей ЭЭГ чаще всего осуществляется путем визуального осмотра записи или количественного анализа ЭЭГ.

Колебания напряжения, измеряемые электродами, позволяют оценить активность мозга. Поскольку электрическая активность, наблюдаемая с помощью ЭЭГ, возникает в нейронах нижележащей ткани головного мозга, записи, сделанные электродами на поверхности скальпа, различаются в зависимости от их ориентации и расстояния до источника активности. Кроме того, записанные значения искажаются промежуточными тканями и костью, которые действуют подобно резисторам и конденсаторам в электрической цепи. Это означает, что не все нейроны вносят одинаковый вклад в сигнал ЭЭГ, при этом ЭЭГ преимущественно отражает активность нейронов коры вблизи электродов на коже головы. Глубокие структуры мозга, расположенные дальше от электродов, не будут вносить непосредственный вклад в ЭЭГ.

Данные электроэнцефалограмм могут быть использованы во множестве различных областей. Например, для диагностики заболеваний головного мозга, в нейромашинных интерфейсах [3], для проведения различных исследований и др.

Таким образом, в прикладных задачах при цифровой обработке эффективность использования «сырых» данных ЭЭГ низкая. Для применения данных ЭЭГ необходим большой набор различных алгоритмов предобработки и анализа этих данных.

Для решения этой задачи была разработана специальная программная платформа. Она содержит набор средств для автоматизированной обработки сигналов ЭЭГ и их анализа, в том числе методами машинного обучения.

Архитектура платформы схематически представлена на рис. 1.

На схеме отражены модули управления, чтения данных, предобработки и анализа, а также вспомогательные компоненты. Модуль управления обеспечивает взаимодействие всех компонентов платформы. Модуль чтения данных позволяет выполнять чтение данных ЭЭГ из разных форматов, в том числе из потока напрямую с электроэнцефалографа (при чтении используется поток *Lab Streaming Layer*, LSL-поток). Состав остальных компонентов подробно описан в следующих разделах.

Алгоритмы предобработки ЭЭГ

Под алгоритмами предобработки обычно понимают различные фильтры и другие методы подготовки данных для последующего анализа. Измеряемый сигнал ЭЭГ всегда содержит артефакты, которые могут повлиять на анализ данных. Артефакт — это любой измеренный сигнал,

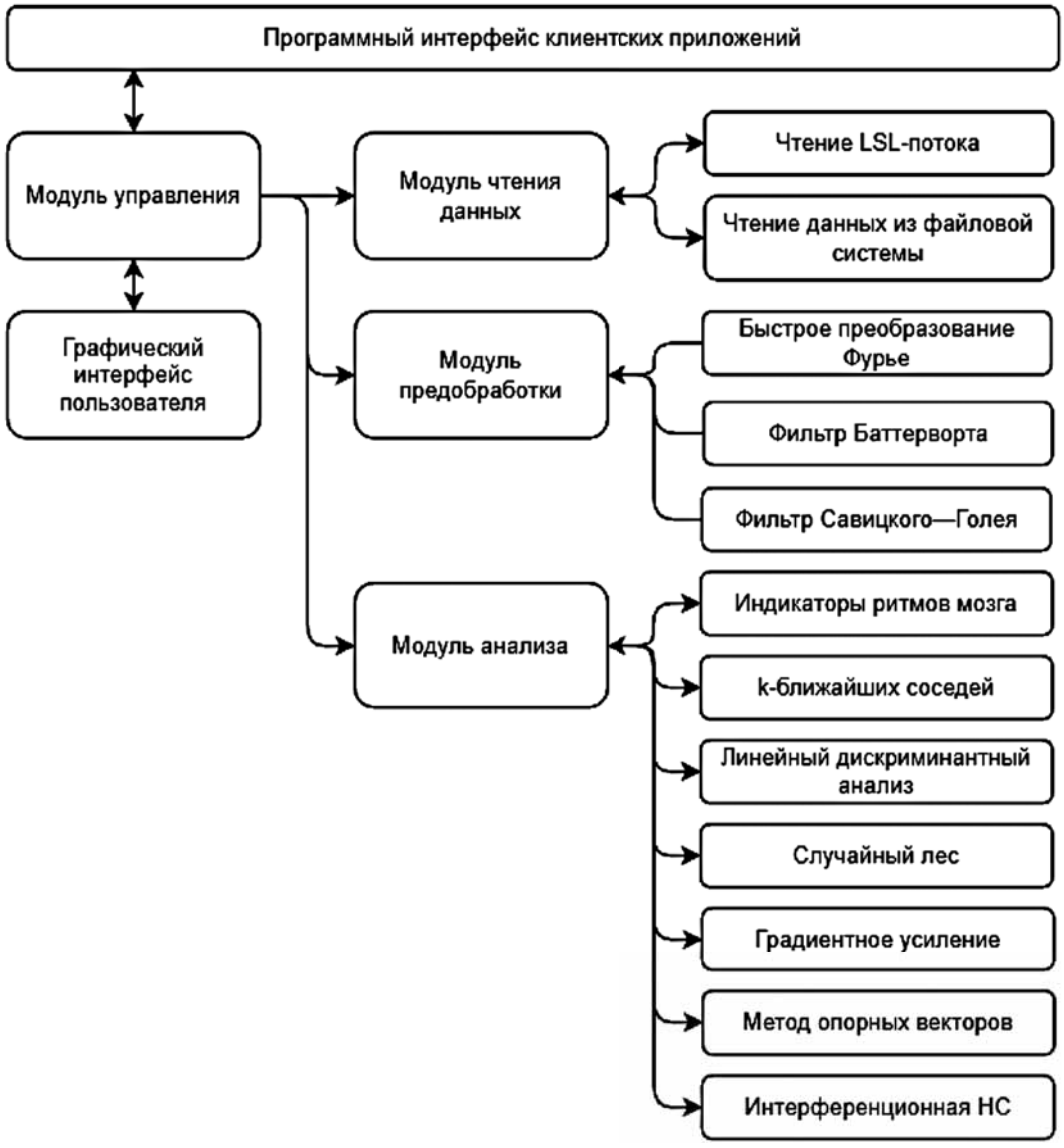


Рис. 1. Архитектура программной платформы

который не исходит из мозга. Хотя существует большое число алгоритмов удаления артефактов, проблемный вопрос о том, как с ними бороться, остается открытым. Источником артефактов могут быть особенности, связанные с прибором, такие как неисправные электроды, линейный шум или высокое сопротивление электродов. Они также могут быть связаны с физиологией записываемого субъекта. Например, такие особенности могут включать моргание и движение глаз, сердечную и мышечную активность. Такие типы артефактов сложнее удалить. Артефакты могут искажать визуальную интерпретацию данных ЭЭГ, поскольку некоторые из них могут имитировать когнитивную деятельность, что может повлиять на диагностику таких случаев, как болезнь Альцгеймера или нарушения сна. Таким образом, удаление таких артефактов в данных ЭЭГ, используемых для практических приложений, имеет первостепенное значение.

Опишем доступные в программном комплексе методы предобработки данных ЭЭГ.

Фильтр Баттерворта. Фильтр Баттерворта — это тип фильтра обработки сигналов, предназначенный для получения как можно более плоской частотной характеристики в полосе пропускания [4]. Он не имеет пульсаций в полосе пропускания и в полосе задерживания, поэтому его иногда называют максимально плоским фильтром. Фильтр Баттерворта позволяет достигать плоскостности за счет относительно широкой области перехода от полосы пропускания к полосе задерживания со средними переходными характеристиками.

Фильтр Савицкого—Голея. Фильтр Савицкого—Голея — это цифровой фильтр, который может быть применен к дискретному набору данных [5]. Он позволяет извлекать полезную информацию из зашумленных данных, при правильном подборе параметров не изменяя общий вид сигнала, в отличие от многих других методов сглаживания. Такой результат достигается в процессе, известном как свертка. Фильтр осуществляет полиномиальную аппроксимацию отдельных кадров входного сигнала по критерию минимума средней квадратической ошибки [9]. Данный фильтр похож на метод фильтрации при помощи скользящего среднего. Однако в методе скользящего среднего все данные усредняются с одним и тем же коэффициентом, в то время как в фильтре Савицкого—Голея применяется взвешенное усреднение.

Быстрое преобразование Фурье. Быстрое преобразование Фурье позволяет совершить переход в частотную область [6]. Получение спектральной характеристики является важным инструментом при анализе сигналов и позволяет узнать распределение мощности сигнала в зависимости от его частоты.

Алгоритмы анализа

Сигналы ЭЭГ, которые прошли предобработку, становятся пригодными для анализа. Под анализом здесь понимают применение набора методов, который направлен на решение поставленной задачи. В этот набор могут входить методы классификации и кластеризации данных, построения прогнозирующих моделей, выделения необходимых признаков и т. д.

Индикатор наличия ритмов головного мозга. Важной задачей при определении состояния головного мозга является оценка наличия того или иного ритма мозга. Ритмы головного мозга — набор особых колебаний электрической активности мозга, каждое из которых имеют свой диапазон частот и предназначение.

Частота альфа-ритма находится в диапазоне 8...14 Гц, амплитуда волн — 30...70 мкВ. Этот ритм отчетливо заметен при закрытии глаз испытуемого и выражен в затылочной области. Бета-ритм имеет частоту 13...30 Гц, амплитуду волн — 5...30 мкВ и возникает при активном состоянии испытуемого. Он выражен в лобных областях. Частота гамма-ритма варьирует от 30 до 120...180 Гц, амплитуда волн — 2...10 мкВ. Гамма-ритм наблюдается при решении задач, требующих сосредоточенного внимания. В ряде публикаций освещается наличие разнообразных нарушений гамма-активности у больных шизофренией. Дельта-ритм находится в пределах 0,5...4 Гц, амплитуда волн — 50...500 мкВ. Он возникает при глубоком естественном или наркотическом сне, а также при коме. Тета-ритм имеет частоту 4...8 Гц, амплитуду волн — 10...30 мкВ. Этот частотный диапазон связан с глубоким отдыхом головного мозга.

Индикатор наличия ритма — некое значение, с помощью оценки которого можно сделать вывод о наличии или отсутствии того или иного ритма [8]. На настоящее время в предлагаемом автором программном комплексе реализовано определение присутствия у испытуемого трех ритмов — альфа, тета и дельта. Для получения значения индикатора необходимо выделить нужный диапазон частот

с помощью фильтрации ЭЭГ-сигналов, а затем рассматривать не сами сигналы, а их частотные характеристики. Применение фильтра Савицкого—Голея к полученной характеристике приводит к ее сглаживанию. Разделение полученной кривой на три равные части и вычисление площадей под этими отрезками позволяет получить значение-индикатор наличия того или иного ритма в мозговой активности. Таким образом, индикатор вычисляется следующей формулой:

$$R = 1 - \frac{S_0 + S_2}{2S_1}, \quad (1)$$

где S_0, S_1 и S_2 — площади левой, средней и правой областей соответственно. Полученное значение индикатора $R \leqslant 1$ позволяет оценить наличие того или иного ритма. Чем значение ближе к единице, тем с большей вероятностью можно утверждать, что ритм присутствует. Значение в окрестности нуля и меньше свидетельствует о том, что ритма, скорее всего, нет.

Градиентное усиление (LGBM). Градиентное усиление — это метод машинного обучения, используемый, в частности, в задачах регрессии и классификации [8]. Он позволяет получить модель прогнозирования в виде ансамбля слабых моделей прогнозирования, которые обычно представляют собой деревья решений. Когда дерево решений является «слабым учеником», результирующий алгоритм называется деревьями с градиентным усилением.

Модель деревьев с градиентным усилением строится поэтапно, она обобщает другие методы, позволяя оптимизировать произвольную дифференцируемую функцию потерь. Как и другие методы усиления, усиление градиента объединяет «слабых учеников» в одного «сильного ученика» итеративным образом.

Интерференционная нейросетевая модель. Интерференционная модель нейронной сети (НС) принципиально отличается от большинства классических моделей искусственных НС [9]. Структура НС этой модели максимально приближена к структуре биологических НС в головном мозге человека. Нейрон в этой модели представляет собой самоорганизующийся объект, а его обучение происходит за счет перемещения рецепторов под действием нейромедиатора, который выделяется синапсами (как в биологическом нейроне). Сигнал подается последовательно, распределено по времени, при этом количество данных обучения получается значительно меньше по сравнению с классическими моделями (что позволяет экономить память), так как необходимо хранить только координаты рецепторов в конечный момент времени и длины их траекторий. Для успешной классификации данных достаточно одного нейрона на один класс. Еще одной важной особенностью является то обстоятельство, что НС этой модели можно легко дообучать без необходимости начинать обучение с нуля.

Интерференционная модель нейронной сети зарекомендовала себя в решении задач распознавания изображений, однако в силу своей универсальности может быть применена к задаче классификации данных ЭЭГ [10—12].

Помимо перечисленных выше методов программной платформой поддерживаются и другие известные алгоритмы, такие как линейный дискриминантный анализ, случайный лес, k-ближайших соседей, метод опорных векторов (SVM).

Особенности программной реализации

Программный комплекс чтения, обработки и анализа данных ЭЭГ имеет свои особенности, актуальные при работе с потоком данных от электроэнцефалографа. Во-первых, для обработки сигналов используется многопоточность. Это позволяет выполнять операции над сигналами параллельно, что в свою очередь сокращает общее время, затрачиваемое на обработку сигналов. Во-вторых, отрисовка графиков осуществляется на графическом процессоре, что позволяет сократить задержки при выводе результатов.

Графический интерфейс пользователя позволяет выполнять настройку окружения при проведении исследований данных ЭЭГ. Это означает, что платформой может воспользоваться даже человек, не имеющий навыков программирования, задачей которого является проверка различных гипотез относительно обработки данных ЭЭГ в рамках решения поставленной задачи. Окружением в данном случае называется набор параметров и настроек алгоритмов, реализованных в программной платформе. На рис. 2 представлен графический интерфейс программной платформы.

Рассматриваемая платформа предоставляет открытый для пользователя программный интерфейс взаимодействия. Это позволяет исполь-

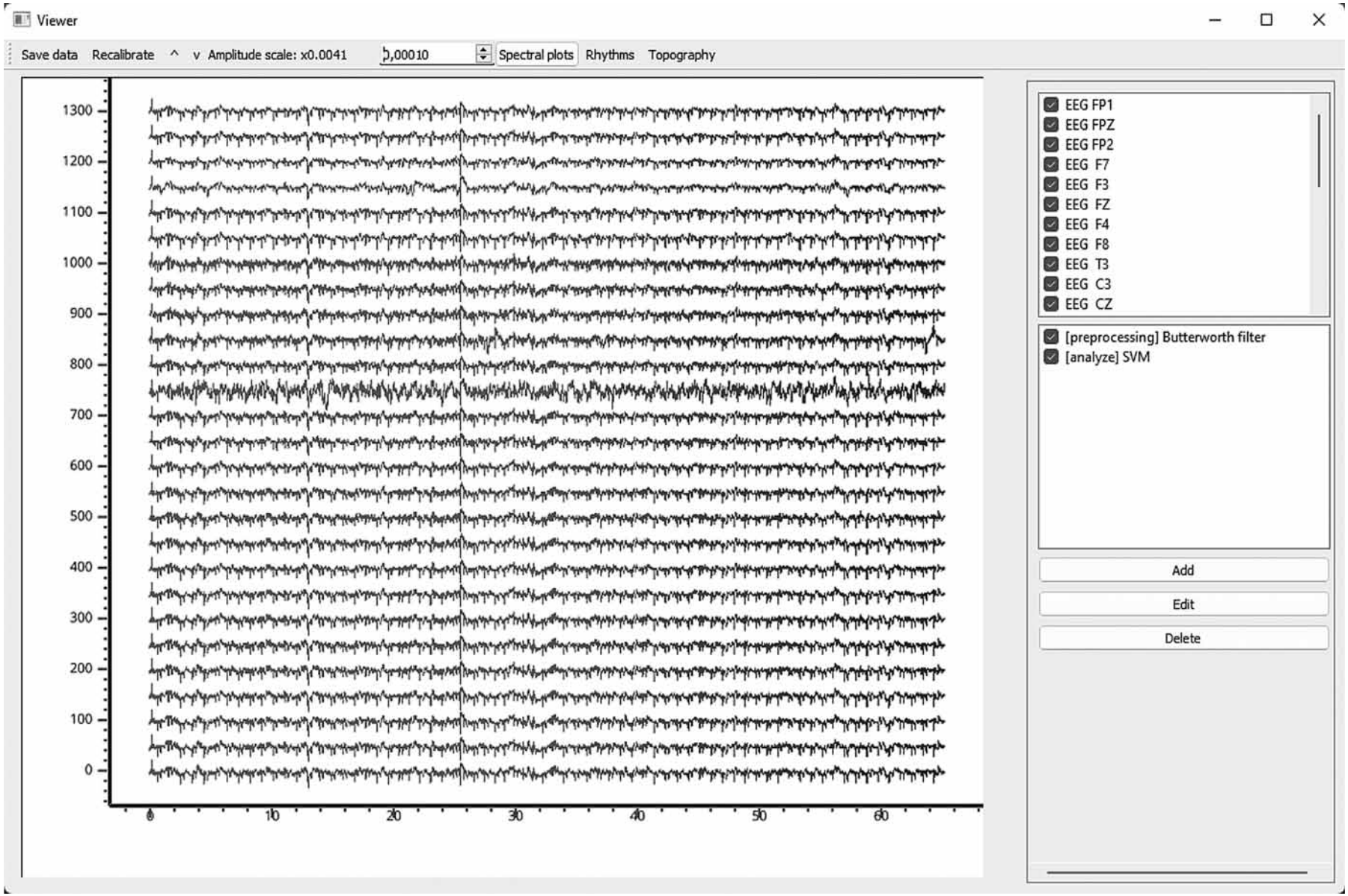


Рис. 2. Графический интерфейс программной платформы

зовать платформу как библиотеку для построения собственных приложений для работы с данными ЭЭГ. Это обстоятельство особенно полезно на этапе, например, прототипирования нового устройства, выполняющего роль управляющего блока в некоторой системе, использующей данные с электроэнцефалографа. Разработчику достаточно импортировать настройки окружения, реализовать связь с источником данных и объектом управления.

Заключение

Разработанная автором программная платформа для чтения, обработки и анализа данных ЭЭГ обладает широким набором функциональных возможностей. Поддержка различных алгоритмов предобработки и анализа данных делает ее удобным инструментом при исследовании данных ЭЭГ и разработке прототипов устройств, использующих эти данные.

Предлагаемая программная платформа решает важную задачу — повышение доступности разработки устройств, управляемых с помощью активности головного мозга. В частности, данная платформа уже используется в разработке человеко-машинных интерфейсов для управления механическими объектами. Эти интерфейсы находят применение в медицине, в частности, при создании специальных устройств для людей с ограниченными возможностями. С применением технологий программной платформы могут быть разработаны различные протезы или инвалидные коляски с управлением, использующим активность головного мозга.

Список литературы

1. **Teplan M.** Fundamentals of EEG measurement // Meas Sci. 2002. Vol. 2, No. 2. P. 1—11.
2. **Britton J. W., Frey L. C., Hopp J. L.** et al. Electroencephalography (EEG): An Introductory Text and Atlas of Normal and Abnormal Findings in Adults, Children, and Infants // American Epilepsy Society. 2016. URL: https://pubmed.ncbi.nlm.nih.gov/27748095/ (дата обращения 04.04.2023).
3. **Sarah N., Ayman A., Mostafa-Sami M.** Brain computer interfacing: Applications and challenges // Egyptian Informatics Journal. 2015. Vol. 16, No. 2. P. 213—230. DOI: 10.1016/j.eij.2015.06.002.
4. **Hussin S., Hamid Z., Birasamy G.** Design of Butterworth Band-Pass Filter // Politeknik & Kolej Komuniti Journal of Engineering and Technology. 2016. Vol. 1. URL: https://myjms.mohe.gov.my/index.php/PMJET/article/view/1169 (дата обращения 04.04.2023).
5. **Luo J., Ying K., Bai J.** Savitzky—Golay smoothing and differentiation filter for even number data // Signal Processing. 2005. Vol. 85, Issue 7. P. 1429-1434. DOI: 10.1016/j.sigpro.2005.02.002.
6. **Duhamel P., Vetterli M.** Fast fourier transforms: A tutorial review and a state of the art // Signal Processing. 1990. Vol. 19, Issue 4. P. 259—299. DOI: 10.1016/0165-1684(90)90158-U.
7. **Babbysh N.** Computing brain rhythm indicators of EEG signal // 2021 5th Scientific School Dynamics of Complex Networks and their Applications (DCNA). Калининград, 2021. С. 32—35. DOI: 10.1109/DCNA53427.2021.9587284.
8. **Jerome H.** Greedy function approximation: A gradient boosting machine // The Annals of Statistics, Institute of Mathematical Statistics. 2001. P. 1189—1232. DOI: 10.1214/aos/1013203451.
9. **Бабич Н. А.** Параметрический синтез интерференционной модели нейронной сети // Вестник современных исследований. 2019. Т. 1, № 13 (28). С. 52—56.
10. **Бабич Н. А., Останин М. Л**. Распознавание объектов на изображениях высокого разрешения с помощью интерференционной нейронной сети // Молодежь. Техника. Космос: труды XI Общероссийской молодежной науч.-техн. Конф. Т1. БГТУ «Военмех». СПб, 2019. С. 229—231.
11. **Бабич Н. А.** О применении интерференционной нейронной сети для динамического анализа данных в реальном времени // Автоматизация в промышленности. 2020. № 4. С. 19—21.
12. **Interference** C++ library. Открытый репозиторий GitHub. URL: https://github.com/nickware44/interference (дата обращения 12.03.2023).

Software Platform for Reading, Processing and Analyzing EEG Data

N. A. Babbysh, Postgraduate Student, nickware@mail.ru,
Institute for Problems in Mechanical Engineering of the Russian Academy of Sciences,
Saint-Petersburg, 199178, Russian Federation

Corresponding author:

Nikolay A. Babbysh, Postgraduate Student,
Institute for Problems in Mechanical Engineering of the Russian Academy of Sciences,
Saint-Petersburg, 199178, Russian Federation
E-mail: nickware@mail.ru

Received on March 14, 2023
Accepted on April 04, 2023

Electroencephalogram (EEG) data can be used in many different areas. For example, for diagnosing brain diseases, in brain computer interfaces, for conducting various studies, and much more. To apply EEG data, a large set of different algorithms for preprocessing and analyzing these data is needed. This paper describes a software platform containing a set of tools for automated processing of EEG signals and their analysis, including machine learning methods. The platform has a flexible architecture and consists of modules, which allows it to be used for various purposes. Data can be obtained both from files and directly from the electroencephalograph device in real time. The graphical interface provides a convenient way to configure the modules of the software. The software interface of client applications (API) makes it possible to use this platform to create prototypes of devices that use EEG data for their work.

Keywords**:** *signal analysis, EEG analysis, machine learning, brain rhythm indicators, signal filtering, brain rhythms, software platform*

For citation:

Babbysh*:* **N. A.** Software Platform for Reading, Processing and Analyzing EEG Data, *Programmnaya Ingeneria*, 2023, vol. 14, no. 5, pp. 254—260. DOI: 10.17587/prin.14.254-260.

References

1. **Teplan M.** Fundamentals of EEG measurement, *Meas Sci.* 2002, vol. 2, no. 2, pp. 1—11.

2. **Britton J. W., Frey L. C., Hopp J. L.** et al. Electroencephalography (EEG): An Introductory Text and Atlas of Normal and Abnormal Findings in Adults, Children, and Infants, *American Epilepsy Society*, 2016, available at: https://pubmed.ncbi.nlm.nih.gov/27748095/ (date of access 04.04.2023).

3. **Sarah N., Ayman A., Mostafa-Sami M.** Brain computer interfacing: Applications and challenges, *Egyptian Informatics Journal*, 2015, vol. 16, no. 2, pp. 213-230. DOI: 10.1016/j.eij.2015.06.002.

4. **Hussin S., Hamid Z., Birasamy G.** Design of Butterworth Band-Pass Filter, *Politeknik & Kolej Komuniti Journal of Engineering and Technology*, 2016, vol. 1, available at: https://myjms.mohe.gov.my/index.php/PMJET/article/view/1169 (date of access 04.04.2023).

5. **Luo J., Ying K., Bai J.** Savitzky—Golay smoothing and differentiation filter for even number data, *Signal Processing*, 2005, vol. 85, issue 7, pp. 1429—1434. DOI: 10.1016/j.sigpro.2005.02.002.

6. **Duhamel P., Vetterli M.** Fast fourier transforms: A tutorial review and a state of the art, *Signal Processing*, 1990, vol. 19, issue 4, pp. 259—299. DOI: 10.1016/0165-1684(90)90158-U.

7. **Babbysh N.** Computing brain rhythm indicators of EEG signal, *2021 5th Scientific School Dynamics of Complex Networks and their Applications (DCNA), Kaliningrad, Russian Federation,* 2021, pp. 32—35.

8. **Jerome H.** Greedy function approximation: A gradient boosting machine, *The Annals of Statistics, Institute of Mathematical Statistics,* 2001, pp. 1189—1232.

9. **Babbysh N.** Parametric synthesis of the interferential neural network, *Bulletin of modern research*, 2019, vol. 1, no. 13 (28), pp. 52—56 (in Russian).

10. **Babbysh N., Ostanin M.** Object recognition in high-resolution images using an interferential neural network, *Molodej. Technika. Kosmos: Proceedings of the XI All-Russian Youth Scientific Technical Conference,* Saint Petersburg, BSTU "Voenmeh", 2019, pp. 229—231 (in Russian).

11. **Babbysh N.** On the application of an interference neural network for dynamic data analysis in real time, *Automation in Industry*, 2020, vol. 4, pp. 19—21 (in Russian).

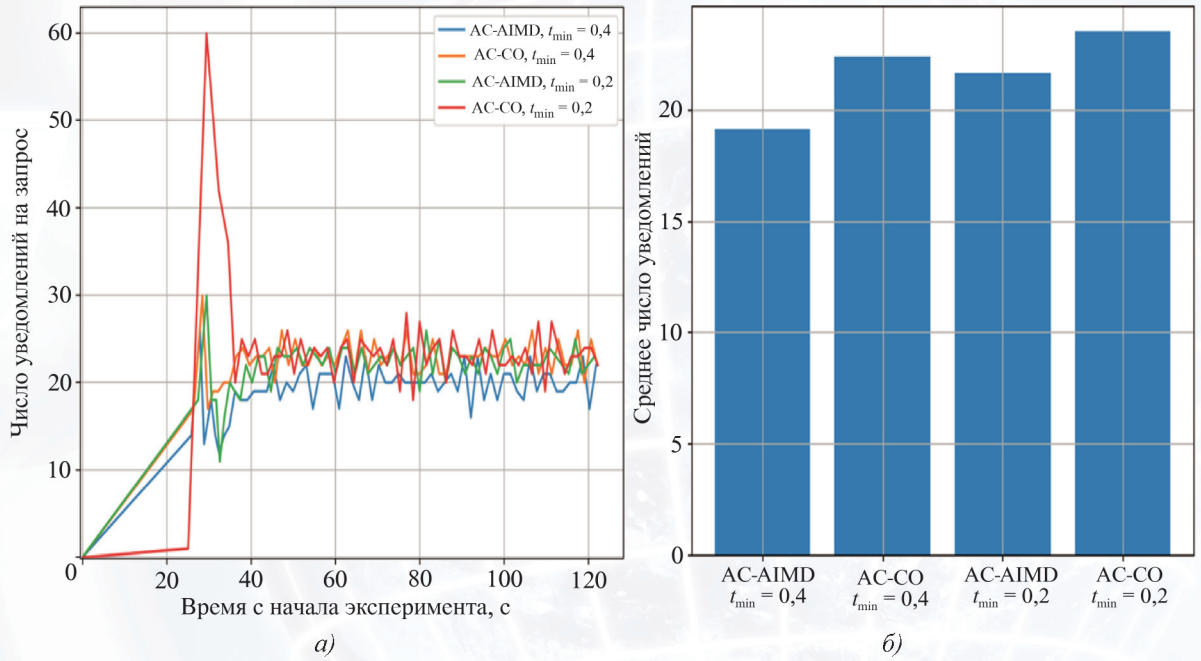
12. **Interference** C++ library. GitHub open repository, available at: https://github.com/nickware44/interference (date of access 12.03.2023).

ООО "Издательство "Новые технологии". 107076, Москва, ул. Матросская Тишина, д. 23, стр. 2
Технический редактор *Е. В. Конова*. Корректор *А. В. Чугунова*.

Сдано в набор 05.04.2023 г. Подписано в печать 28.04.2023 г. Формат 60×88 1/8. Заказ PI523
Цена свободная.

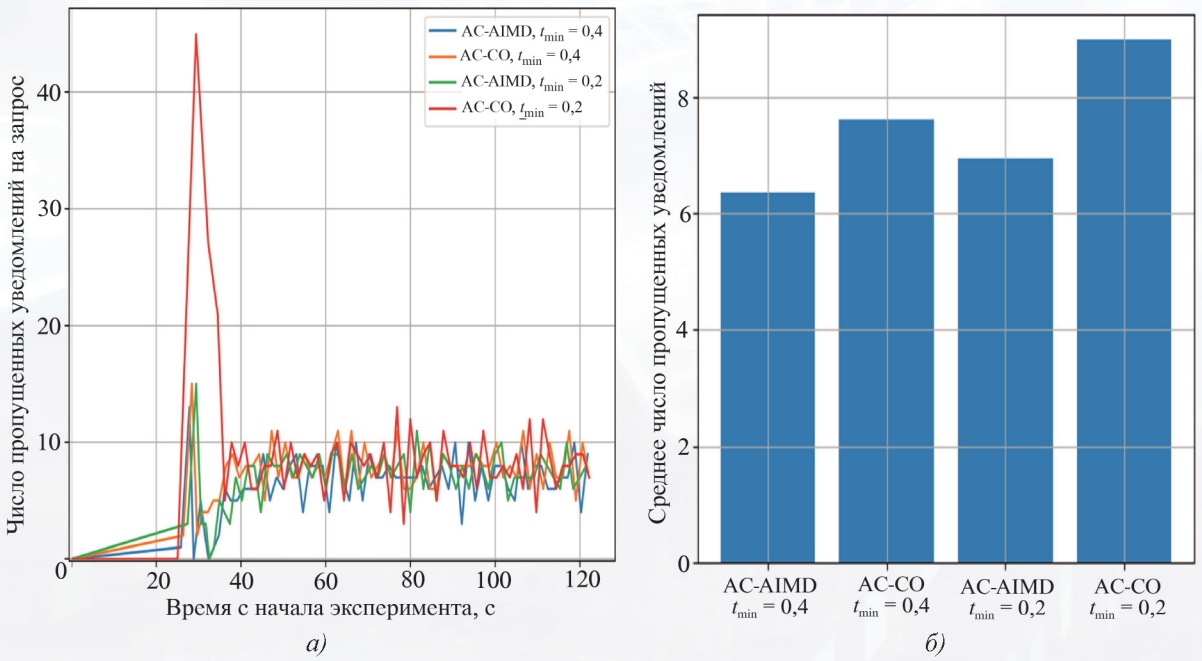
Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
119071, г. Москва, Ленинский пр-т, д. 19, стр. 1. Сайт: **www.aov.ru**

Рисунки к статье **Д. Ж. Корзуна, О. Ю. Богоявленской, К. А. Кулакова**
«ПРИМЕНЕНИЕ АЛГОРИТМА СЛУЧАЙНОЙ ОТСРОЧКИ ПРИ АКТИВНОМ УПРАВЛЕНИИ ОБМЕНОМ ИНФОРМАЦИИ В ИНТЕРНЕТ-СРЕДЕ»



а) *б)*

Рис. 4. Число уведомлений, получаемых ПА за один запрос:
а – поведение числа уведомлений; *б* – среднее число уведомлений на запрос за весь период



а) *б)*

Рис. 5. Число потерь – пропущенных уведомлений за один запрос:
а – поведение числа уведомлений;
б – среднее число пропущенных уведомлений на запрос за весь период

Рисунки к статье П. Н. Советова

«АЛГОРИТМЫ УЛУЧШЕНИЯ АВТОМАТИЧЕСКИ СИНТЕЗИРОВАННОГО НАБОРА КОМАНД РАСШИРЯЕМОГО ПРОЦЕССОРА»

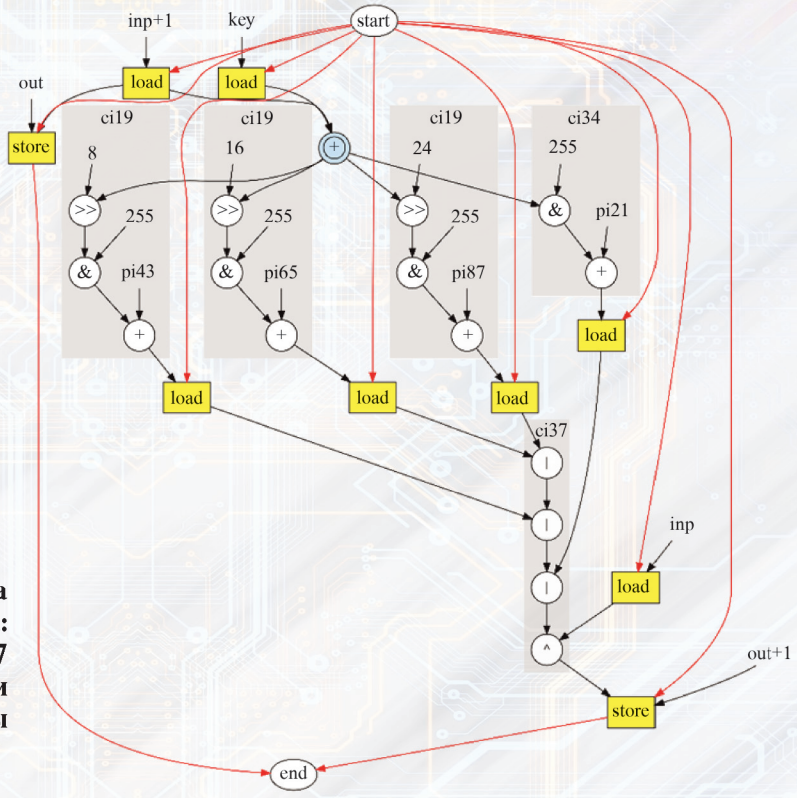


Рис. 1. Результаты работы алгоритма MaxMISO для раунда шифра Магма: кластеры ci19, ci34, ci37 являются кандидатами в синтезированные команды

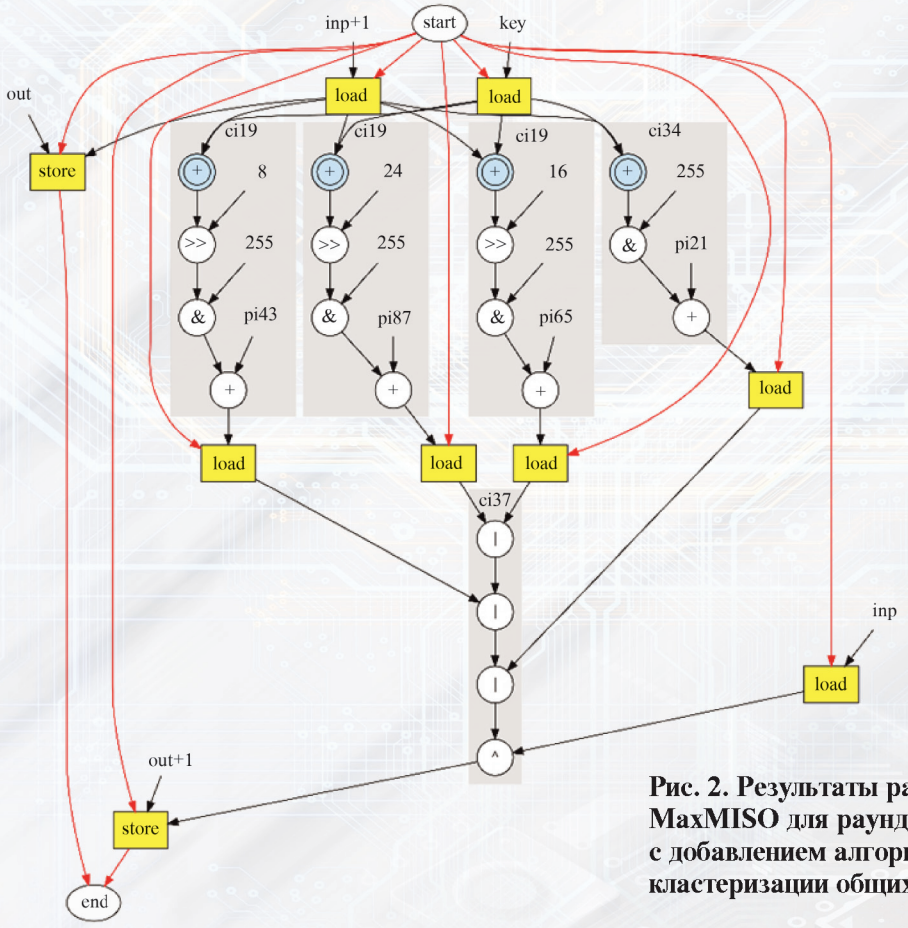


Рис. 2. Результаты работы алгоритма MaxMISO для раунда шифра Магма с добавлением алгоритма кластеризации общих операций

ISSN 2220-3397. Программная инженерия. 2023. Том 14. № 5. 205–260. Подписной индекс по Объединенному каталогу «Пресса России» – 22765