Санкт-Петербургский государственный университет

Материалы
весенней научно-практической
конференции по вопросам информатики,
математики, механики и астрономии

МАТ-МЕХ. НАУКА 2025

28 апреля – 3 мая 2025 г.
Санкт-Петербург

Санкт-Петербург
2025

УДК 004; 51; 531/534; 52
ББК 16; 22.1; 22.2; 22.6
М34

Печатается по рекомендации
кафедры системного программирования
Санкт-Петербургского государственного университета

Материалы весенней научно-практической конференции по вопросам информатики, математики, механики и астрономии «Мат-мех. Наука 2025». 28 апреля – 3 мая 2025 г. Санкт-Петербург — СПб.: Издательство ВВМ, 2025. — 58 с.
ISBN 978-5-9651-1632-4

Тематика сборника затрагивает широкий круг актуальных проблем теоретической и прикладной математики, информатики, механики и астрономии. Цели конференции: представление результатов и организация научного общения преподавателей, ведущих учёных и молодых исследователей, апробация результатов студентов и аспирантов, установление и укрепление научных связей между исследовательскими группами.

Для студентов и аспирантов естественно-научных специальностей.

Р е ц е н з е н т:
к.ф.-м.н., доцент СПбГУ Д. В. Луцив

Параллельные алгоритмы, вэйвлетная обработка числовых потоков

Макаров Антон Александрович
д.ф.-м.н., профессор, заведующий кафедрой параллельных алгоритмов

О контроле ошибки смешанных методов для решения эллиптических уравнений 4-го порядка с точным представлением границы области

Корнеев В.Г., СПбГУ, Санкт-Петербург vad.korneev2011@yandex.ru

Аннотация

A reliable and computable a posteriori error bound is derived for the mixed Ciarlet–Raviart method for the equation $\Delta\Delta u + \kappa u = f(x)$, $x \in \Omega \subset R^2$, with the first boundary condition and a piecewise constant $\kappa \geq 0$. Several authors derived residual type a posteriori error bounds at the assumptions that $\kappa \equiv 0$ and the domain is polygonal, none of which is used in this work. In case of a piecewise smooth boundary, we consider the mixed method with the triangular Lagrange finite elements of the 3d order, which, in general, are curvilinear along the boundary. The curvilinear finite elements provide an approximation to the boundary, matching the finite elements in accuracy. Our bounds belong to the class of a posteriori functional majorants and are evaluated with help of functions from the testing C^1 space. By the reasons of accuracy and simplicity, this space is generated by the finite elements with the domains, coinciding with the domains of the Lagrange elements, and singular rational coordinate functions.

Прикладная кибернетика и искусственный интеллект

Кузнецов Николай Владимирович
член-корреспондент РАН, д.ф.-м.н., профессор, заведующий кафедрой прикладной кибернетики

Алгоритмы генерации тестовых данных

Тишенинов М.А., СПбГУ, Санкт-Петербург st096244@student.sbpu.ru,
Благов М.В., СПбГУ, Санкт-Петербург st024103@student.spbu.ru

Аннотация

В данной статье приводится адаптация алгоритма DSOMA для генерации данных для тестирования SQL-запросов, поддерживающий все виды соединений, а также агрегационые запросы и оконные функции. В работе также приводится сравнительный анализ с другими алгоритмами, решающими такую задачу.

Введение

Формирование тестовых данных — одна из ключевых задач, возникающих в процессе тестирования программного обеспечения, и она сохраняет свою актуальность на протяжении многих лет [1, 2]. Это особенно важно в контексте тестирования взаимодействий с базами данных [3].

При тестировании запросов к базам данных разработчики сталкиваются с рядом вызовов, поскольку любые значимые изменения в программном обеспечении, взаимодействующем с базами данных, обычно приводят к усложнению SQL-запросов. С ростом требований к функциональности и производительности систем, запросы становятся все более многоуровневыми, включают сложные фильтры, объединения таблиц, агрегации и группировки. Рост сложности запросов также влечёт за собой увеличение временных затрат на получение тестовых данных для их тестирования [2].

Одним из популярных подходов к получению тестовых данных является их генерация [4, 5], она позволяет быстро получить необходимый набор данных для тестирования программы и при этом не несёт рисков раскрытия конфиденциальной информации, являющихся существенным ограничением для использования реальных данных при тестировании.

Постановка задачи

SQL-запрос

В процессе исполнения SQL-запроса любая СУБД формирует план его исполнения, который можно рассматривать как программу P. Входными данными такой программы будут являться данные, удовлетворяющие ограничениям, которые заданы типами данных и связями между таблицами в базе данных, обозначим всё множество входных данных за D. У SQL-запроса имеется

набор сценариев, по которому может пойти его исполнение, обозначим этот набор за S. В результате работы программы над некоторыми исходными данными $T \subset D$ получаем выходные данные, обозначим их $P(T)$.

Тестовые данные

Для тестирования SQL-запроса необходим некоторый набор данных, который с одной стороны позволяет проверить большое число сценариев из S, но одновременно не является слишком громоздким. Обозначим набор тестовых данных как T, причём $T \subseteq D$, а также $|T| < +\infty$.

Для оценки качества тестовых данных будем набор функций M. Формально их можно определить как отображения $\mu \in M : T \to \mathbb{R}^+$. Значением функции $\mu \in M$ на наборе тестовых данных T будет являться количество шагов в плане исполнения сценария $s \in S$, которое осталось пройти после получения нулевого результата. При нулевом значении функции результат работы программы на определённом сценарии будет не пуст, а значит в наборе данных существует подмножество, которое позволит протестировать исходный сценарий. Таким образом с каждым рассматриваемым сценарием $s \in S$ связана функция μ_s.

Генерация тестовых данных

Известно множество сценариев S, множество функций M, связанных со сценариями и ограничения которым должны удовлетворять входные данные, то есть известно множество D.

Сформулируем задачу генерации тестовых данных для SQL-запроса в виде задачи оптимизации:

$$F(T) = \lambda|T| + \sum_{s \in S} \mu_s(T) \to \min_{T \in 2^D} \tag{1}$$

где 2^D — множество всех возможных подмножеств множества D, λ – параметр, отвечающий за значимость размера набора тестовых данных.

Задача генерации тестовых данных сведена к задаче минимизации и поэтому можем применять методы решения оптимизационных задач, в том числе и генетические алгоритмы.

Покрытие тестовых сценариев

Покрытие тестовых сценариев будем использовать в качестве метрики эффективности работы алгоритмов генерации тестовых данных. Оно будет измеряться в процентах, а вычислятся как умноженное на 100 отношение числа сценариев $s \in S$, для которых результат исполнения на наборе тестовых данных будет непустым к общему числу сценариев.

Получение всего множества сценариев исполнения S для SQL-запроса производится с помощью процедуры SQLFpc[6], которая подразумевает разбиение SQL-запроса на предикаты, обращение всех полученных предикатов и компоновку новых SQL-запросов из предикатов SQL-запроса и обращённых предикатов.

Алгоритм генерации тестовых данных

Принцип работы DSOMA

DSOMA [7] принадлежит к классу генетических алгоритмов, их суть заключается в том, что минимум целевой функции приближается с помощью последовательных итераций над набором данных, имитирующих естественный отбор.

Алгоритм в качестве входных данных принимает набор из M векторов из $\mathbb{N}^n$, которое обычно называют начальной популяцией, обозначим её за X_0. Также для работы алгоритма необходима некоторая функция $F : \mathbb{N}^{n \times M} \to \mathbb{R}^+$, с помощью которой может быть определено отношение порядка между элементами множества входных данных.

Итерационный процесс начинается с сортировки популяции X_k, для этого необходимо вычислить значения функции F на каждом её элементе и затем отсортировать эти элементы по убыванию значения F. Первый элемент в отсортированной популяции назовём лидером.

Для всех остальных элементов популяции, кроме лидера, проведём следующую процедуру, описанную в Алг. 1.

Алгоритм 1. Создание новых элементов популяции.

```
1  FOR i = 2, 3, ..., M DO:
2    FOR j = 1, 2, ..., n DO:
3      J[j] = abs(X_k[1, j] - X_k[i, j]);
4    J_max = mode(J);
5    IF J_max >= J_min:
6      s = ceil(J_max / J_min);
```

```
7     ELSE:
8       s = 1;
9     FOR j = 2, 3, ..., M DO:
10     FOR l = 1, 2, ..., J_min DO:
11      IF X_k[i, j] < X_k[1, j]:
12        G[l, j] = X_k[i, j] + s * l;
13      ELSE IF X[i, j] > X[1, j]:
14        G[l, j] = X_k[i, j] - s * l;
15      ELSE:
16        G[l, j] = 0;
```

После выполнения процедуры для всех элементов популяции получим набор множеств G_i, состоящих из векторов, которые "двигаются" в направлении лидера. Объединим X_k и все G_i, после чего вычислим для значения функции F и отсортируем по возрастанию значения функции F получившееся множество, после этого удалим из него все элементы кроме первых M, тем самым получив новую популяцию X_{k+1}.

Адаптация DSOMA для генерации тестовых данных

Можно заметить, что DSOMA не подходит для решения задачи генерации тестовых данных, так как множеством в котором ведётся поиск решения в задаче является множество всех подмножеств множества D или для краткости 2^D в то время как DSOMA ведёт поиск решения в пространстве $\mathbb{N}^{n\times M}$.Поэтому для его применения надо свести рассматриваемую задачу к задаче, которую может решить алгоритм. Для этого необходимо было сделаны следующие модификации.

Во-первых, необходимо ограничить множество входных данных D, введём для этого параметр $a \in \mathbb{N} : |D_a| = a$. Это позволит установить биекцию между пространством индексов записей в таблицах $\mathbb{N}^a$ и пространством D_a записей в l таблицах, необходимых для рассматриваемого запроса, так как элементами D_a в случае SQL-запроса является конечное число записей в таблицах, которые можно проиндексировать. Записи в таблицах индексируются независимо для каждой таблицы таким образом, чтобы при чтении по порядку индексов они были отсортированы в лексикографическом порядке. Набором тестовых данных в таком случае будем называть матрицу T, которая состоит из l векторов $\mathbb{N}^n$, где $n \leq \frac{a}{l}$ содержащих индексы записей для каждой таблицы, используемой в SQL-запросе, таким образом $T = (t_{ij})_{i=1,j=1}^{n,l}$, где $t_{ij} \in \mathbb{N}$.

Во-вторых, необходимо определить функцию F, с помощью которой бу-

дем определять отношение порядка на множестве входных данных. Для этого будем использовать целевую функцию $F(T) = \mu_s(T)$, где $s \in S$, подобная функция уже встречалась в формулировке (1).

Процедура генерации начинается с парсинга рассматриваемого SQL-запроса, это позволяет выделить все константы, которые используются в этом запросе и аттрибуты с которыми они взаимодействуют. После этого начинается генерация данных, однако используя только что полученные отношения констант в запросе и аттрибутов при генерации записей в таблицах производится подстановка этих констант в сгенерированные элементы, что гарантирует наличие элементов с таким значением, а также наличие элементов с другими значениями. В каждой таблице, задействованной в SQL-запросе генерируется равное число записей.

Таким образом можем найти T_s^* с помощью модифицированного DSOMA, для каждого доступного сценария из S и после этого объединить эти наборы данных в единый набор T^*, который будет учитывать все возможные сценарии. Количество данных в каждом из T_s^* регулируется параметром n и поэтому в ходе работы алгоритма не производится оптимизация по размеру T^*.

Эксперименты

Параметры

Эксперименты проводились на наборе из 40 разнообразных SQL-запросах, взятых из системы управления взаимодействиями с клиентами EspoCRM [8]. Запросы содержать различные виды соединения таблиц, агрегационные запросы и оконные функции.

Основными параметрами алгоритма генерации тестовых данных, помимо SQL-запроса и схемы необходимых для него данных, являются M — размер популяции, a — размер множества входных данных, J — количество новых векторов, получаемых на каждой итерации алгоритма для каждого элемента популяции. K — количество новых векторов, добавляемое на каждой итерации алгоритма для сохранения разнообразия, а также n — количество записей из каждой таблицы D_a, индексы которых находятся в элементах T.

Результаты

Об эффективности работы можно судить по совокупности двух показателей — времени работы алгоритма и проценту покрытия сценариев. Для на-

бора SQL-запросов на которых проводились эксперименты это множество известно заранее.

В результате проведения ряда экспериментов с различными значениями параметров были получены следующие результаты об эффективности работы алгоритма в сравнении с другими алгоритмами, решающими ту же задачу.

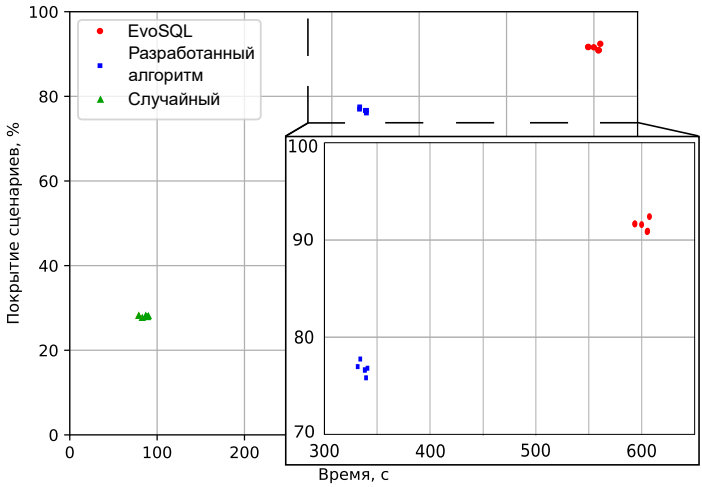


Рис. 1: Результаты тестовых запусков.

На Рис. 1 изображены результаты тестовых запусков разработанного алгоритма (квадраты), случайного подбора тестовых данных (треугольники), а также алгоритма EvoSQL [9] (круги).

В сравнении со случайным выбором представленный в статье алгоритм показывает намного более высокий процент покрытия сценариев. В сравнении с EvoSQL, представленный в статье алгоритм исполняется на 44% быстрее, хотя и с меньшим, но при этом сравнимым процентом покрытия сценариев.

Заключение

В рамках данной работы на основе DSOMA был разработан алгоритм генерации данных для тестирования SQL-запросов, а также было проведено

сравнение разработанного алгоритма с существующими аналогами, по итогам которого было выяснено, что разработанный алгоритм позволяет добиваться хорошего покрытия сценариев на 44% быстрее, чем ближайший аналог.

Список литературы

[1] Chan M., Shing C., Testing Database Applications with SQL Semantics // International Symposium on Cooperative Database Systems for Advanced Applications, 1999

[2] C. Yan, S. Nath, S. Lu, Generating Test Databases for Database-Backed Applications // 2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE), Melbourne, Australia, 2023, pp. 2048-2059

[3] Haller K., The test data challenge for database-driven applications // Proceedings of the Third International Workshop on Testing Database Systems (DBTest), Article 6, 1–6.

[4] Baudry B. et al. Generative AI to Generate Test Data Generators // IEEE Software, vol. 41, no. 6, pp. 55-64

[5] Patki N., Wedge R., Veeramachaneni K. The Synthetic Data Vault // 2016 IEEE International Conference on Data Science and Advanced Analytics (DSAA), 2016, pp. 399-410

[6] Tuya J., Suárez-Cabal M., María José, de la Riva C., Full predicate coverage for testing SQL database queries // Software Testing, Verification Reliability, Volume 20, Issue 3, 2010, pp. 237-288

[7] Davendra D., Zelinka I., Pluhacek M., Senkerik R. DSOMA—Discrete Self Organising Migrating Algorithm // Self-Organizing Migrating Algorithm. Studies in Computational Intelligence, vol 626. Springer, pp 51-63.

[8] EspoCRM. Исходный код: [Электронный ресурс] // URL: https://github.com/espocrm/espocrm (Дата обращения: 06.03.2025)

[9] Castelein J., Aniche M., Soltani M., Panichella A., van Deursen A. Search-based test data generation for SQL queries // Proceedings of the 40th International Conference on Software Engineering, Association for Computing Machinery, pp 1220–1230.

Глубокое обучение для оценки оператора Купмана в идеализированной динамике атмосферы

Кирпичев И.Е., СПбГУ, Санкт-Петербург st088147@student.spbu.ru,
Мокаев Р.Н., СПбГУ, Санкт-Петербург r.mokaev@spbu.ru

Аннотация

Атмосферные процессы описываются нелинейными системами, что существенно затрудняет их прямой анализ с помощью численных методов. Глубокое обучение позволяет разрабатывать модели, которые по точности сопоставимы с традиционными физическими моделями для среднесрочных прогнозов. В данной работе исследуется подход применения глубокого обучения для оценки оператора Купмана, который обеспечивает линейное представление нелинейных динамик и способствует повышению прозрачности модели основанной на данных. Для этого создаётся набор данных, который описывает динамику развития холодных и тёплых пузырей в атмосфере. Далее реализуется частичный автоэнкодер на основе сверточной нейронной сети, который обучается на этих данных. Результаты работы модели сравниваются с исходными данными.

Введение

Анализ и прогнозирование динамических процессов, таких как изменение скорости ветра, плотности и температуры, возникающих в атмосфере, играют важную роль в современном обществе. Традиционные физические и численные модели, хотя и обеспечивают приемлемую точность, требуют значительных вычислительных ресурсов. Кроме того, сложность и нелинейность атмосферной динамики затрудняют её прямой анализ, что делает необходимым поиск подходов, позволяющих одновременно улучшить интерпретируемость моделей и сохранить высокую точность прогнозов.

Одним из таких подходов является применение оператора Купмана [2], [3], который преобразует нелинейную динамическую систему в линейную бесконечномерную систему. Несмотря на теоретическую привлекательность оператора Купмана, его прямое применение для задач атмосферной динамики остаётся затруднительным вследствие бесконечной размерности и необходимости эффективной аппроксимации оператора в конечномерных пространствах. Современные методы глубокого обучения предлагают инстру-

менты для приближённой реализации оператора Купмана. Подходы на основе нейронных сетей (например, автоэнкодеров и сверточных нейронных сетей) позволяют найти оптимальное представление данных в скрытом пространстве, где оператор Купмана будет действовать линейно, сохраняя при этом богатство исходной информации и повышая интерпретируемость полученных моделей. В частности, сверточные автоэнкодеры показали себя перспективными благодаря способности эффективно работать с многомерными пространственными данными и успешно справляться с высокими уровнями нелинейности и шума.

Генерация обучающего набора данных

Физическая модель

Для решения задачи будем использовать двумерные нестационарные уравнения Эйлера, описывающие движение сжимаемой жидкости:

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u^1}{\partial x^1} + \frac{\partial \rho u^3}{\partial x^3} = 0, \tag{1}$$

$$\frac{\partial \rho u^1}{\partial t} + \frac{\partial (\rho u^1 u^1 + p)}{\partial x^1} + \frac{\partial (\rho u^1 u^3)}{\partial x^3} = 0, \tag{2}$$

$$\frac{\partial \rho u^3}{\partial t} + \frac{\partial (\rho u^1 u^3)}{\partial x^1} + \frac{\partial (\rho u^3 u^3 + p)}{\partial x^3} + \rho g = 0, \tag{3}$$

$$\frac{\partial \rho \theta}{\partial t} + \frac{\partial \rho u^1 \theta}{\partial x^1} + \frac{\partial \rho u^3 \theta}{\partial x^3} = 0, \tag{4}$$

$$p_0 \left(\frac{R_d \rho \theta}{p_0} \right)^{\gamma} = p, \tag{5}$$

где ρ – плотность жидкости, p – давление, u^1 и u^3 – горизонтальная и вертикальная скорости соответственно, θ – потенциальная температура, t – время, x^1 и x^3 – горизонтальная и вертикальная пространственные координаты, $g \approx 9.81\,\text{м/с}^2$ – ускорение свободного падения, R_d – удельная газовая постоянная для сухого воздуха, γ – показатель адиабаты, $p_0 = 100000\,\text{Па}$ – опорное давление.

Для численного решения применяется разрывный метод Галёркина 4-го порядка [5] на сетке 20×20 элементов с 5 узлами на элемент (всего 10,000 точек). Временная дискретизация выполняется полунеявным методом Розенброка 2-го порядка [6] с шагом $\Delta t = 5$.

Линейные системы на каждом шаге решаются итерационным методом FGMRES с предобуславливанием. Матрица Якоби аппроксимируется методом конечных разностей, а для ускорения сходимости используется LU-разложение. Сравнение решений с предобуславливанием и без него показало пренебрежимо малые различия (l_2-норма разности $\sim 1.1^\circ C$, относительная ошибка $\sim 0.037\%$ на последнем шаге генерации).

Генерация данных

На основе модели WxFactory [4] создан набор данных с тепловыми пузырями [7] в двумерной атмосфере. Параметры генерации:

- Сетка 1000×1000 м
- 1-2 горячих ($\theta = 30.3 - 30.6^\circ C$) и 0-2 холодных ($\theta = 29.8 - 29.95^\circ C$) пузырей
- Гауссово распределение температуры: $\theta(r) = A \cdot \exp[-(r-a)^2/s^2]$ при $r > a$; A при $r \leq a$
- Радиусы $a = 10 - 80$ м, координаты центров $x_0 = 200 - 800$ м, $z_0 = 50 - 750$ м

Результаты генерации

Всего было сгенерировано 960 уникальных сценариев эволюции атмосферы (720 обучающих и 240 валидационных). Каждый сценарий содержит до 215 временных шагов. При генерации некоторые временные шаги сталкивались с неустойчивостью, в таких случаях в набор не добавлялись последние 10 шагов.

Архитектура модели и оператор Купмана

Оператор Купмана

Рассмотрим $\Omega \subset \mathbb{R}^2$, введем:

$$x_k(x^1, x^3) = \big(\rho_k(x^1, x^3),\, u_k^1(x^1, x^3),\, u_k^3(x^1, x^3),\, \theta_k(x^1, x^3)\big) : \Omega \to \mathbb{R}^4,$$

где ρ_k, u_k^1, u_k^3, θ_k — плотность, скорости ветра в двух направлениях и температура соответственно. Рассмотрим $M = L^2(\Omega, \mathbb{R}^4)$, M - множество состояний. Динамика системы определяется отображением $f \in C^\infty : M \to M$, $x_{k+1} = f(x_k)$, образуя дискретную динамическую систему (M, f).

Оператор Купмана $K : F \to F$, где $F = \{g : M \to \mathbb{C}\}$, задаётся как

$$[Kg](x_k) = g(f(x_k))$$

.

Частичный автоэнкодер на основе сверточной нейронной сети

Работа [8] предоставляет базовую методику оценки оператора Купмана, однако ее подход сталкивается с серьезными трудностями при применении к данным высокой размерности [1]. В данной работе используется модифицированная структура автоэнкодера, предложенная в работе [10]. Энкодер преобразует x_k в скрытое представление $z_k = g(x_k)$, где g — нелинейное отображение. В скрытом пространстве применяется оператор K_m: $z_{k+1} = K_m z_k$. Декодер восстанавливает состояние: $\tilde{x}_{k+1} = g^{-1}(z_{k+1})$.

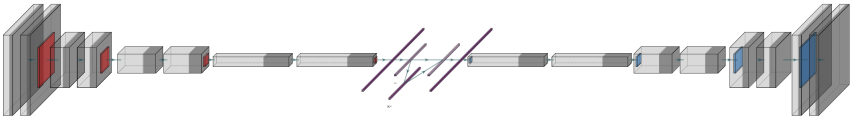


Рис. 1: Архитектура частичного автоэнкодера на основе сверточной нейронной сети. Энкодер (слева): остаточные блоки и Max Pooling. Скрытое пространство (центр): K_m для продвижения состояния. Декодер (справа): остаточные блоки и транспонированная свёртка.

Архитектура модели представлена на рисунке 1. Энкодер состоит из блоков снижения разрешения (DownBlock), которые включают в себя последовательности двух остаточных блоков (Residual Blocks) [9], после которых идет операция Max Pooling. Декодер включает блоки повышения разрешения (UpBlock) с транспонированной свёрткой и остаточными блоками. После энкодера данные преобразуются в вектор, проецируются в скрытое пространство, где применяется K_m, и затем декодируются.

Метрики оценки и функции потерь

Для обучения используется среднеквадратическая ошибка:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2,$$

где y_i — истинное значение, $\hat{y}_i$ — предсказанное.

В работе [8], для эффективного обучения автоэнкодера, оценивающего нелинейные преобразования вместе с оператором Купмана, вводятся три функции потерь (восстановления, предсказания и линейности). Для улучшения обобщающей способности модели и её регуляризации в работе [1] предлагается два дополнительных ограничения (функции потерь шума и замены). Общая функция потерь:

$$\begin{aligned}\mathcal{L} = &\underbrace{\|x_k - g^{-1}(g(x_k))\|_{\text{MSE}}}_{\text{восстановление}} + \underbrace{\|x_{k+1} - g^{-1}(K_m g(x_k))\|_{\text{MSE}}}_{\text{предсказание}} \\ &+ \underbrace{\|g(x_{k+1}) - K_m g(x_k)\|_{\text{MSE}}}_{\text{линейность}} + \underbrace{\|g(\tilde{x}_{k+1}) - K_m g(x_k)\|_{\text{MSE}}}_{\text{шум}} \\ &+ \underbrace{\|g(\tilde{x}_{k+1}) - g(x_{k+1})\|_{\text{MSE}}}_{\text{замена}}\end{aligned} \tag{6}$$

Результаты обучения

Модель обучалась на наборе из 151545 векторов состояний, валидационный набор содержал 50344 вектора. Использовалась ранняя остановка (Early Stopping) для контроля обучения, оптимизация проводилась алгоритмом Adam со скоростью обучения 1×10^{-6}.

Псевдосходимость достигнута на 81 эпохе, с минимальными значениями потерь 0.351 (тренировочный набор) и 0.399 (валидационный набор).

На рисунках 2 и 3 показаны реальное развитие пузыря из валидационного набора и одношаговый прогноз его состояния с использованием начальной температуры соответственно. Полученные результаты свидетельствуют о значительной способности модели к обобщению. Несмотря на то, что моделью использовалась только одна переменная, её прогнозы хорошо согласуются с реальной динамикой развития температурного поля. Кроме того, даже с ограниченными знаниями о динамике модель может выполнять одношаговые предсказания, используя только начальное состояние.

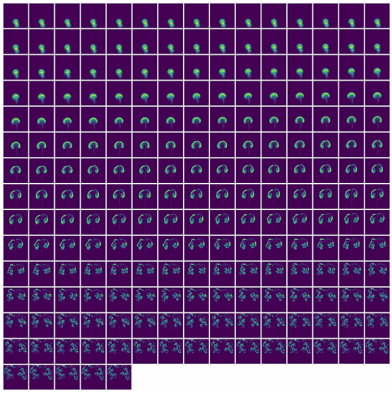


Рис. 2: Оригинальное развитие пузыря из валидационного набора данных.

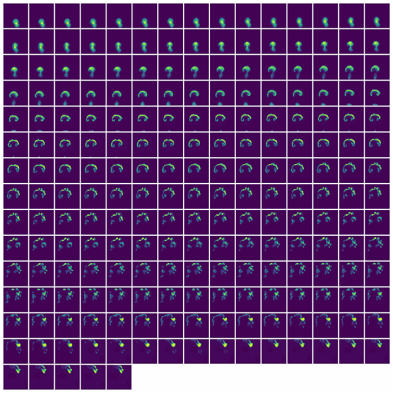


Рис. 3: Однократное предсказание с использованием модели при тех же начальных условиях.

Заключение

В данной работе рассмотрен подход к линейной аппроксимации сложных нелинейных процессов в динамике атмосферы с использованием оператора Купмана. Актуальность выбранной тематики обусловлена сложностью анализа и прогнозирования атмосферных явлений, что имеет важное значение для метеорологии, климатологии и смежных областей.

Для этого был сформирован синтетический набор данных, моделирующий эволюцию температуры в идеализированной атмосфере. Была реализована модель, способная эффективно находить пространство, где оператор Купмана действует линейно. После обучения модели на сгенерированных данных было проведено сравнение прогноза модели с данными из валидационной выборки. Результаты показали, что даже при использовании лишь одного канала (температуры) архитектура успешно улавливает ключевые закономерности динамики. Предсказанные одноступенчатые приращения состояния хорошо согласуются с оригинальными данными, что свидетельствует о способности модели воспроизводить краткосрочную динамику атмосферы с помощью линейного оператора.

Список литературы

[1] D. Millard, A. Carr, and S. Gaudreault. Deep Learning for Koopman Operator Estimation in Idealized Atmospheric Dynamics. *arXiv preprint*,

2024. arXiv:2409.06522.

[2] M. Budisic, R. Mohr, and I. Mezic. Applied Koopmanism. *Chaos*, 22(4):047510, 2012. DOI:10.1063/1.4772195.

[3] M. J. Colbrook, I. Mezić, and A. Stepanenko. Limits and Powers of Koopman Learning. *arXiv preprint*, 2024. arXiv:2407.06312.

[4] S. Gaudreault, M. Charron, V. Dallerit, and M. Tokman. High-Order Numerical Solutions to the Shallow-Water Equations on the Rotated Cubed-Sphere Grid. *arXiv preprint*, 2022. arXiv:2101.05617.

[5] J. S. Hesthaven and T. Warburton. *Nodal Discontinuous Galerkin Methods: Algorithms, Analysis, and Applications*. Springer Science & Business Media, 2007. DOI:10.1007/978-0-387-72067-8.

[6] V. Dallerit, T. Buvoli, M. Tokman, and S. Gaudreault. Second-Order Rosenbrock-Exponential (Rosexp) Methods for Partitioned Differential Equations. *Numerical Algorithms*, 96(3):1143–1161, 2024. DOI:10.1007/s11075-023-01698-4.

[7] A. Robert. Bubble Convection Experiments with a Semi-Implicit Formulation of the Euler Equations. *Journal of the Atmospheric Sciences*, 50(13):1865–1873, 1993. DOI:10.1175/1520-0469(1993)050<1865:BCEWAS>2.0.CO;2.

[8] B. Lusch, J. N. Kutz, and S. L. Brunton. Deep Learning for Universal Linear Embeddings of Nonlinear Dynamics. *Nature Communications*, 9:4950, 2018. DOI:10.1038/s41467-018-07210-0.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. DOI:10.1109/CVPR.2016.90.

[10] Y. Xiao, Z. Tang, X. Xu, X. Zhang, and Y. Shi. A Deep Koopman Operator-Based Modelling Approach for Long-Term Prediction of Dynamics with Pixel-Level Measurements. *IET Cyber-Systems and Robotics*, 5(1):e12149, 2023. DOI:10.1049/cit2.12149.

Управление состояниями нелинейных динамических систем с использованием резервуарных вычислений

Васильев М.А., СПбГУ, Санкт-Петербург mvasilev801@gmail.com,
Мокаев Т.Н., СПбГУ, Санкт-Петербург t.mokaev@spbu.ru

Аннотация

В данной работе исследуется метод управления нелинейными динамическими системами с использованием резервуарных вычислений. Этот подход позволяет переводить систему в заданное целевое состояние без необходимости точного знания её уравнений движения. Основная цель — воспроизвести результаты исследования Haluszczynski и Räth, а также изучить возможности улучшения метода. Эксперименты проводились на системах Лоренца и Рёсслера, демонстрируя эффективность предложенного подхода. В работе обсуждаются преимущества и недостатки метода, а также предлагаются направления его совершенствования.

Введение

Управление нелинейными динамическими системами представляет собой важную задачу в различных областях науки и техники, включая физику, биологию и инженерию. Такие системы, часто проявляющие хаотическое поведение, сложно поддаются управлению традиционными методами, которые требуют точных математических моделей или больших объёмов данных. Развитие методов машинного обучения, в частности резервуарных вычислений, открывает новые перспективы для управления такими системами. RC позволяет строить управляющие стратегии на основе данных о поведении системы, не требуя её явного математического описания.

Цель данной работы — изучить метод управления, предложенный Haluszczynski и Räth , подтвердить его эффективность на примерах систем Лоренца и Рёсслера, а также исследовать пути улучшения. Этот подход основан на использовании эхо-сетей (Echo State Networks, ESN), которые являются разновидностью RC, для предсказания и коррекции динамики системы.

Теоретические основы и методология

Динамические системы

Динамическая система может быть представлена как пара $(\varphi^t{}_{t\in J}, (M, \rho))$, где M — метрическое пространство, J — множество значений времени, а φ^t — семейство отображений, удовлетворяющих условиям тождественности, композиции и непрерывности. Для систем, описываемых дифференциальными уравнениями вида $\dot{x} = f(x)$, управление достигается введением сигнала $u(t)$, изменяющего динамику: $\dot{x} = f(x) + Bu(t)$. Цель управления — перевести систему из текущего состояния в заданное целевое состояние.

Резервуарные вычисления

Резервуарные вычисления — это класс рекуррентных нейронных сетей, состоящих из фиксированного резервуара и обучаемого выходного слоя. В данной работе используется модель эхо-сети (ESN), которая обладает свойством эхо-состояния, обеспечивающим зависимость текущего состояния сети от истории входных данных. Резервуар случайным образом инициализируется с заданным спектральным радиусом (обычно в диапазоне 0.8–1.2), а обучение проводится только для выходного слоя, что снижает вычислительную сложность.

Методы управления

Метод, предложенный Haluszczynski и Räth [1], включает следующие этапы:

1. Симуляция системы в целевом состоянии X.
2. Обучение RC на данных состояния X.
3. Изменение параметров системы для перехода в состояние Y.
4. Использование RC для предсказания поведения в состоянии X и вычисления управляющей силы $F(t) = K(u(t) - v(t))$, где $u(t)$ — текущее состояние, $v(t)$ — предсказанное состояние.
5. Применение $F(t)$ для возвращения системы в целевое состояние X.

Этот подход проверялся на системах Лоренца и Рёсслера, где изменение параметров вызывало переход системы в новое состояние, а RC успешно возвращала её в исходное. В качестве метрики использовались наибольший показатель Ляпунова и корреляционная размерность.

Наибольший показатель Ляпунова (Largest Lyapunov Exponent, LLE) — это величина, которая измеряет, насколько быстро расходятся близкие траектории системы в фазовом пространстве. Она характеризует чувствительность системы к начальным условиям:

Корреляционная размерность (Correlation Dimension, CD) — это мера сложности аттрактора системы, которая связана с её фрактальной размерностью. Она показывает, сколько степеней свободы фактически задействовано в динамике системы.

Синяя траектория - целевая, красная - система без управления, зеленая - система с управлением.

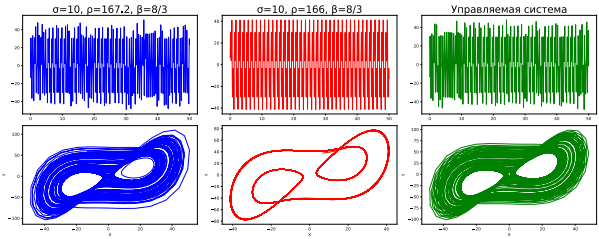


Рис. 1: Управление системой Лоренца с помощью ESN

	Исходная	Измененная	Управляемая
λ (LLE)	2.7878	0.0668	2.7440
μ (CD)	1.8131	1.1955	1.7339

	Исходная	Измененная	Управляемая
λ (LLE)	0.1405	0.2035	0.1382
μ (CD)	1.8150	1.9427	1.7516

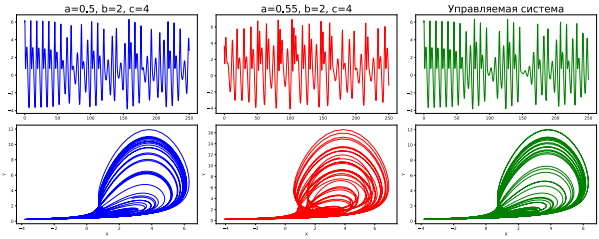


Рис. 2: Управление системой Рёсслера с помощью ESN

Сравнение с другими моделями

Многослойный перцептрон (MLP)

Многослойный перцептрон (MLP) — это базовая архитектура искусственных нейронных сетей, состоящая из входного слоя, одного или нескольких скрытых слоев и выходного слоя. Обучение проводится методом обратного распространения ошибки. В работе MLP применялся со скользящим окном для обработки временных рядов, что позволяло учитывать последовательные данные. Однако из-за отсутствия встроенной памяти эта модель показала ограниченную эффективность в задачах управления динамическими системами по сравнению с более продвинутыми архитектурами.

LSTM (Long Short-Term Memory)

LSTM — это разновидность рекуррентных нейронных сетей, специально разработанная для работы с последовательными данными. Благодаря ячейкам памяти и вентильным механизмам (входной, выходной и забывания), LSTM способна сохранять информацию на длительные временные интервалы. В экспериментах эта модель улучшала точность предсказаний динамики систем Лоренца и Рёсслера по сравнению с MLP, что делает её подходящей для задач управления динамическими системами с долговременными зависимостями.

ReservoirLSTM

ReservoirLSTM представляет собой гибридную архитектуру, сочетающую принципы резервуарных вычислений и LSTM. В этой модели внутренние веса вентилей заменены на резервуары, а обучение проводится только

для выходных весов. Такое сочетание позволяет использовать вычислительные возможности резервуарных вычислений для обработки краткосрочных временных зависимостей и способности LSTM для моделирования долгосрочных закономерностей. В работе эта модель показала улучшенные результаты в управлении динамическими системами.

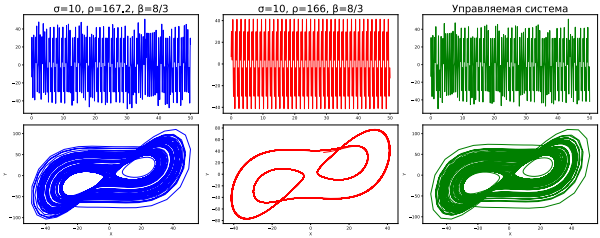


Рис. 3: Управление системой Лоренца с помощью ReservoirLSTM

	Исходная	Измененная	Управляемая
λ (LLE)	2.7878	0.0668	2.8053
μ (CD)	1.8131	1.1955	1.8076

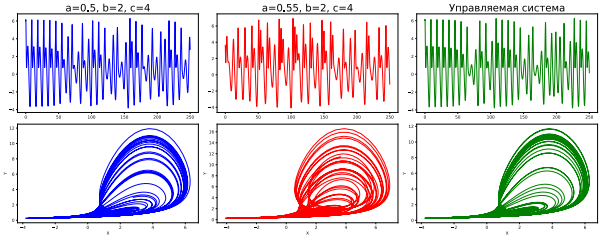


Рис. 4: Управление системой Рёсслера с помощью ReservoirLSTM

	Исходная	Измененная	Управляемая
λ (LLE)	0.1405	0.2035	0.1484
μ (CD)	1.8150	1.9427	1.8301

ESN-LSTM

ESN-LSTM — это ещё одна гибридная модель, интегрирующая резервуарный слой и слой LSTM. Архитектура состоит из резервуарного слоя, слоя LSTM и выходного слоя. Обновлённое состояние резервуара обрабатывается LSTM-блоком, состояние резервуара и LSTM конкатенируются и через линейный слой получаем предсказание модели.

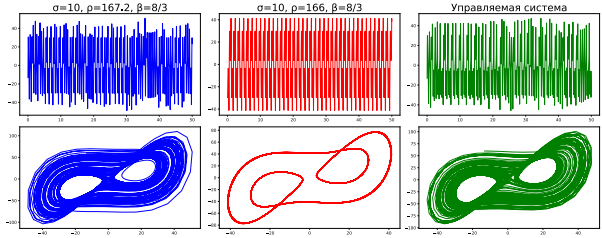


Рис. 5: Управление системой Лоренца с помощью ESN-LSTM

	Исходная	Измененная	Управляемая
λ (LLE)	2.7878	0.0668	2.8053
μ (CD)	1.8131	1.1955	1.8076

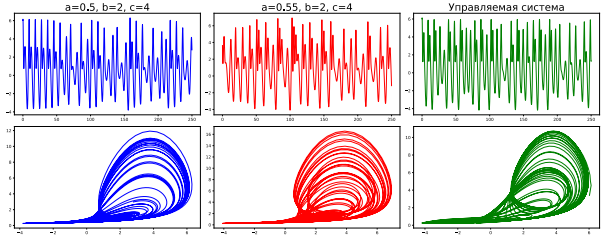


Рис. 6: Управление системой Рёсслера с помощью ESN-LSTM

	Исходная	Измененная	Управляемая
λ (LLE)	0.1405	0.2035	0.1784
μ (CD)	1.8150	1.9427	1.7825

Работа алгоритма

Результаты экспериментов показали, что схема управления на основе ESN успешно возвращает систему к целевому состоянию. Сравнение показывает, что простые MLP и даже одиночные LSTM не справляются в полной мере с задачей управления сложной динамикой без обширной настройки. Резервуарные подходы обладают универсальностью и демонстрируют высокую эффективность в прогнозе и управлении динамическими системами. Именно фиксированный резервуар и обучение только выходного слоя делают такие модели быстрыми и устойчивыми к переобучению, обеспечивая при этом превосходную аппроксимацию сложных отображений фазового пространства.

Заключение

Метод управления на основе ESN позволил восстановить численные атрибуты хаоса (Ляпуновский показатель и корреляционную размерность) практически до исходных значений. Гибридные нейросетевые модели (ReservoirLSTM, ESN-LSTM) оказались более эффективными, чем стандартные MLP или LSTM (с одиночной памятью). Это указывает на перспективность использования резервуарных архитектур в задачах управления хаосом и динамикой. Перспективы дальнейших исследований включают оптимизацию архитектуры резервуарных вычислений, внедрение моделей NG-RC и разработку гибридных подходов с глубокими нейронными сетями, что может повысить точность и расширить область применения метода в реальных задачах управления сложными системами

Список литературы

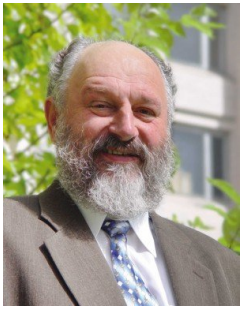
[1] Haluszczynski, A., Räth, C. Controlling nonlinear dynamical systems into arbitrary states using machine learning. Sci Rep 11, 12991 (2021).

[2] Ott, E., Grebogi, C., Yorke, J. A. Controlling chaos. Phys. Rev. Lett. 64, 1196 (1990).

[3] Pyragas, K. Continuous control of chaos by self-controlling feedback. Phys. Lett. A 170, 421-428 (1992).

[4] Grigoryeva, L., Ortega, J.-P. Echo state networks are universal (2018).

Системное программирование и программная инженерия



Терехов Андрей Николаевич
д.ф.-м.н., профессор, заведующий кафедрой системного программирования, президент ООО «Ланит-Терком»

Литвинов Юрий Викторович, к.т.н., старший преподаватель кафедры системного программирования

Луцив Дмитрий Вадимович, к.ф.-м.н., доцент кафедры системного программирования

Сартасов Станислав Юрьевич, к.т.н., Chief Technical Officer, TheDenominator, Inc.

Повышение качества дедупликации с помощью Frequency Based Chunking

Дмитриевцев А.С., СПбГУ, Санкт-Петербург a.dmitrievtsev@spbu.ru,
Гориховский В.И., СПбГУ, Санкт-Петербург gorihovskyvyacheslav@gmail.com

Аннотация

В данной работе предлагается параллельная версия алгоритма улучшения качества дедупликации Frequency Based Chunking. Использование вычисления частот в нескольких потоках позволяет намного улучшить скорость записи в файловой системе, поддерживающей такой подход. Также в работе рассматривается влияние коллизий, возникающих при одновременном анализе чанков.

Введение

Дедупликация данных – подход к сжатию данных, направленный на устранение повторяющихся копий на одном или группе носителей с целью снижения накладных расходов на хранение и передачу информации, широко используется в системах хранения резервных копий.

Content-Defined Chunking (CDC) – метод разделения на блоки для первичной дедупликации, в котором данные используются для определения границ будущих чанков. Для улучшения дедупликации существует подход Frequency Based Chunking[1], который основывается на анализе частот подчанков в данном CDC-разбиении и производит его измельчение. Такой подход позволяет значительно увеличить коэффициент дедупликации, однако на порядки замедляет скорость записи.

В представленной работе предлагается метод оптимизации FBC, основанный на параллельном вычислении частот. Этот метод интегрирован в ChunkFS[2] – модельную файловую систему, предназначенную для комплексного анализа методов дедупликации. Основной целью работы является исследование возможности применения FBC в системах резервного копирования, в особенности TATLIN.BACKUP от компании YADRO.

Алгоритм FBC

Этапы работы Frequency Based Chunking представлены на 1. Скользящее окно проходит по чанкам, полученным после работы CDC-алгоритма. При

нахождении частого подчанка, он добавляется в словарь. На основе частотного анализа проводится дальнейшее дробление: некоторые исходные CDC-чанки разделяются на несколько новых чанков меньшего размера, которые были найдены на предыдущем этапе. Таким образом, вследствие измельчения чанков, мы находим большее число дубликатов, что приводит к повышению качества дедупликации.

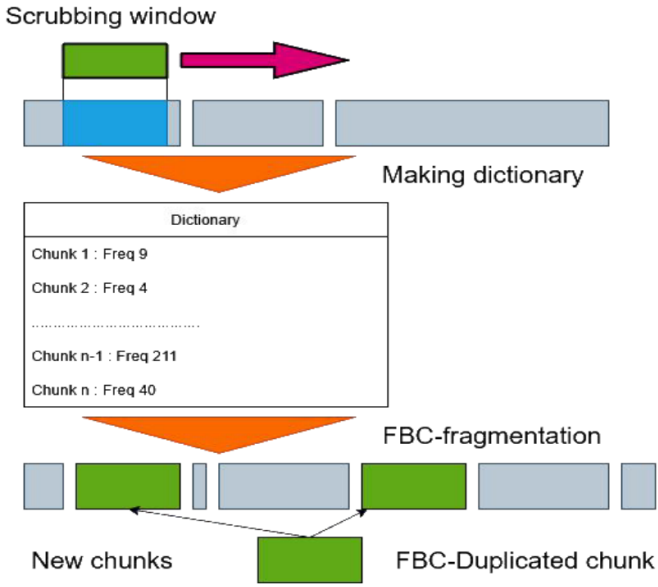


Рис. 1: Этапы алгоритма Frequency Based Chunking

Специфика интеграции

Ранее была реализована и интегрирована в ChunkFS базовая[3] версия алгоритма. Проведенный комплексный анализ эффективности FBC, выявил ряд особенностей алгоритма:

- измельчение чанков приводит к вплоть до стократному снижению скорости записи
- системы хранения резервных копий предполагают редкие, но крупные периоды записи данных

- запись может производиться независимо от чтения уже размещенных данных
- FBC – скраббирующий алгоритм, но при этом требуется существенное улучшение производительности записи
- чтение может легко выполняться параллельно, так как не является деструктивной операцией

Одной из наиболее трудоемких операций в алгоритме FBC является вычисление частот, поэтому наиболее естественным улучшением производительности является их параллельное вычисление, не предполагавшееся в оригинальной работе.

Параллельный подход к FBC

Алгоритм проходит чанки скользящим окном, периодически добавляя в словарь новые записи о наиболее частых подчанках из данного разбиения. Данные действия можно выполнять параллельно, обеспечив конкурентный доступ к словарю частых чанков, однако подобный подход к вычислению частот предвещает недостаток – при параллельной записи могут возникать коллизии. Источником коллизий является одновременное возникновение нового подчанка в разных потоках, затрудняющее корректное определение частоты этого подчанка. Из-за этого может не произойти разбиение самих чанков. Однако выдвигается предположение, что коллизии несущественно влияют на качество дедупликации. Если подчанк действительно частый, он встретится в разных потоках неоднократно и будет добавлен в словарь. Чанки, уже находящиеся в FBC словаре, повторно в него не добавляются. Поэтому не будут участвовать в подразбиении только относительно редкие подчанки.

Для решения задачи необходимо обеспечить эффективный конкурентный доступ к словарю FBC-чанков. Был проведен анализ эффективности различных реализаций: Mutex<HashMap>, RwLock<HashMap>, DashMap

Наибольшую эффективность показал вариант с DashMap, поэтому выбор был сделан именно в пользу данной реализации.

Экспериментальное исследование

Было проведено экспериментальное исследование, направленное на оценку производительности решения и влияния коллизий на качество дедупликации.

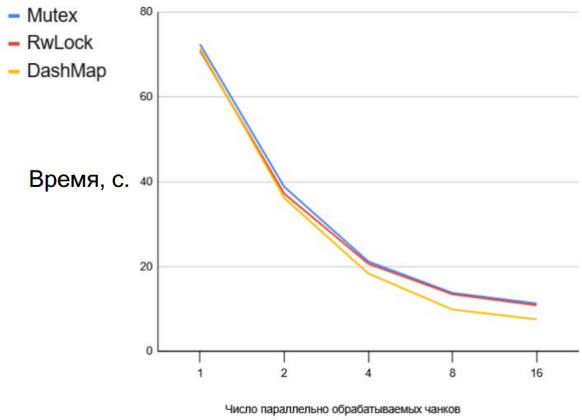


Рис. 2: Скорость вычисления частот для разных вариантов синхронизации, на фрагменте 20% от Enron

Постановка эксперимента

Исследовательский вопрос 1 - Как влияет изменение числа потоков на качество дедупликации?

Исследовательский вопрос 2 - Насколько эффективно параллельное вычисление частот чанков?

Для исследования были выбраны два датасета: Enron emails (1,4 Гб), две версии ядра Linux в совокупности (3.4.6 и 3.4.7, 900 Мб). Enron emails содержит малое количество дубликатов крупного размера и предназначен для определения улучшения качества дедупликации в сравнении с базовыми CDC алгоритмами. Ядра Linux в свою очередь содержат большое количество дубликатов и лучше выделят коллизии.

В качестве метрик были выбраны: время записи целого датасета в секундах и коэффициент дедупликации (считается как начальный размер данных, деленный на размер данных после обработки). Исследовались запуски на 1, 2, 4, 8 и 16 потоках, в которых весь датасет загружался в словарь и вычислялось время, затраченное на составление словаря, и объем такого словаря. Такие запуски проводились по 10 раз, из которых выбирались худшие, средние и лучшие результаты.

Исследования проводились в системе со следующими характеристиками: ОС Linux, дистрибутив Manjaro 24.1.1, процессор Intel® Core™ i7-12650H,

объем оперативной памяти 16 Гиб.

Результаты экспериментального исследования

На 4 приведены результаты влияния коллизий на качество дедупликации, для датасета Enron emails слева и ядер Linux справа. С ростом числа потоков наблюдается снижение коэффициента дедупликации за счет коллизий. При этом большее снижение заметно на датасете Enron emails. Это можно объяснить малым количеством дубликатов в этом датасете и, соответственно, меньшей частотой встречи подчанков. В датасете с ядрами Linux более вероятная последующая повторная встреча подчанка может нивелировать коллизию. Отсюда можно сделать вывод, что коллизии не оказывают существенного влияния на качество дедупликации: около 1-5% при записи новых чанков. Что значительно меньше выигрыша от использования FBC: около 30-40%[3].

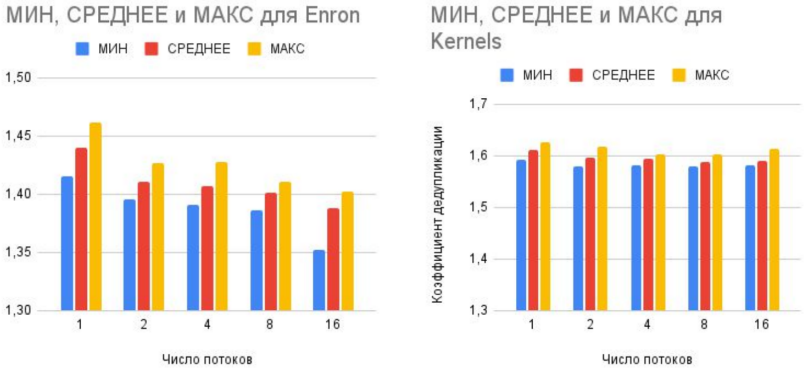


Рис. 3: Коэффициент дедупликации при параллельном исполнении

Результаты оценки времени составления словаря представлены на 4. Разница между лучшими и худшими показателями незначительна (не превышает 2%), поэтому на графике приведены только средние значения. Параллельная реализация показала существенный прирост в скорости составления словаря частых чанков, вплоть до 10 раз на 16 потоках. Однако, возможно, прирост эффективности между использованием 8 и 16 потоков является уже не настолько существенным, чтобы тратить столько ресурсов, которые можно было бы использовать на оптимизацию других частей ChunkFS. Например, на оптимизацию скорости записи чанков в хранилище. Более того, использование 16 потоков приводит к снижению коэффициента дедупликации ввиду

возникновения большего количества коллизий.

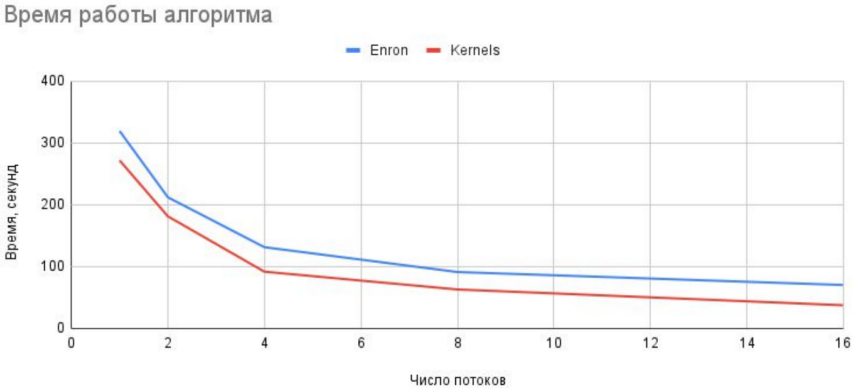


Рис. 4: Время на составление словаря для используемых датасетов

Для систем хранения резервных копий мы можем рекомендовать использовать параллельную реализацию на 8 потоках. Более того, для снижения количества коллизий можно предложить проводить запись первой версии последовательно. Это может уменьшить количество коллизий и снизить ухудшение качества дедупликации от использования параллелизма. Такой подход не ухудшит производительность регулярного использования системы, а только скорость ее инициализации.

Заключение

В работе предложен подход к распараллеливанию алгоритма FBC. Параллельное вычисление частот позволило достичь скорости алгоритма, сравнимой с базовыми CDC алгоритмами, при практически сохранении улучшения качества дедупликации. Также на основе экспериментального исследования предлагается эвристическое решение для использования в системах хранения резервных копий. Оно позволяет снизить количество коллизий при вычислении частот за счет последовательной записи первой версии и параллельной записи последующих.

Список литературы

[1] Lu, Guanlin & Jin, Yu & Du, David. (2010). Frequency Based Chunking for Data De-Duplication. 287 - 296. 10.1109/MASCOTS.2010.37. https://ieeexplore.ieee.org/document/5581583

[2] Piletskii O., Gorikhovskii V. (2025). ChunkFS: A Tool for Data Deduplication Methods Comparison. 221 - 227. Proceedings of FRUCT'37.

[3] Дмитриевцев А. С., Гориховский В. И. Повышение качества дедупликации с помощью Frequency Based Chunking. CFP: сборник материалов конференции, Санкт-Петербург, 23–24 апреля 2025 года. Санкт-Петербургский политехнический университет Петра Великого. – Санкт-Петербург: ПОЛИТЕХ-ПРЕСС, 2025. – С. 307-309.

Автоматизация процессов подготовки и проведения защит ВКР

Мясникова М.Г., СПбГУ, Санкт-Петербург mariamya243@gmail.com

Аннотация

Данная работа посвящена разработке веб-приложения для автоматизации процессов подготовки и проведения защит выпускных квалификационных работ. В статье изложены требования к функциональности системы, описаны её архитектура и реализация.

Введение

Процедура защиты выпускных квалификационных работ (ВКР) проводится на заседании Государственной экзаменационной комиссии (ГЭК), где члены комиссии при помощи критериев оценивают работы на основании их содержания, оформления, качества доклада студента, а также отзывов научного руководителя и рецензента [1].

Однако организация процесса защиты ВКР сопряжена с трудностями, характерными для большинства вузов: использование бумажных оценочных листов или электронных таблиц затрудняет работу комиссии. Кроме того, применяемые на заседаниях критерии оценивания часто обладают сложной структурой.

Таким образом, процессы подготовки и проведения защит ВКР нуждаются в автоматизации. Предлагается разработать веб-приложение, которое позволит упростить процесс подготовки к защите при помощи генерации необходимых документов и предоставит членам комиссии информацию о заседании с возможностью оценить работу по критериям и автоматически подсчитать итоговую оценку. Данное решение планируется внедрить в процесс защит ВКР и учебных практик на кафедре системного программирования СПбГУ, однако оно также может быть использовано и другими университетами.

Требования

Требования к веб-приложению были сформированы на основе списка, предоставленного научным руководителем, и уточнялись в процессе разработки.

В приложении должны быть предусмотрены две роли пользователей: администратор и член комиссии (Рис. 1).

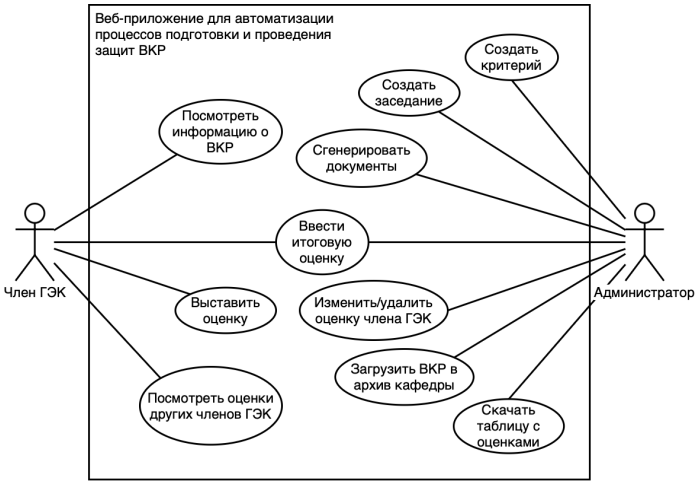


Рис. 1: Диаграмма случаев использования веб-приложения

Администратор может создавать заседания вручную, указывая основную информацию и задавая списки защищающихся студентов и принимающих участие членов комиссии, или автоматически генерировать из .xlsx-таблиц с расписаниями различного формата. Для каждого заседания должны быть указаны критерии оценивания, выбранные из ранее созданного администратором списка. Каждый критерий состоит из названия, комментария, шкалы оценивания и дополнительных правил. В результате администратор должен получить ссылку на заседание, которую сможет распространить членам ГЭК.

После создания заседания администратор может сгенерировать документы для проведения защиты ВКР, а именно: ведомость (одна для заседания, содержит списки защищающихся студентов и членов комиссии) и оценочные листы (по одному для каждого члена ГЭК, содержит информацию о члене комиссии и список защищающихся студентов).

В ходе защиты каждый член комиссии выставляет оценки по критериям текущему защищающемуся с возможностью оставить комментарии. Члены комиссии могут менять свою оценку вплоть до окончания заседания и видеть оценки, выставленные другими участниками. Администратор, в свою очередь, имеет право отредактировать или удалить любую оценку.

После защиты администратору и всем членам комиссии должны быть видны усреднённые оценки по каждому отдельному критерию, общая средняя оценка и итоговая оценка за работу для обсуждения. На этапе голосования оценки остаются доступными для редактирования. Посчитанная оценка име-

ет рекомендательное значение для комиссии, итоговая оценка может быть изменена администратором или любым членом ГЭК в результате обсуждения.

По результатам заседания работы, успешно прошедшие защиту, загружаются в архив кафедры по команде администратора, а также генерируется .xlsx-таблица со всеми оценками, выставленными членами комиссии, средними оценками и итоговой оценкой и её версия для студентов (без оценок членов комиссии).

Используемые технологии

ASP.NET Core [2] — кроссплатформенный и высокопроизводительный фреймворк для разработки веб-приложений. В проекте использован подход Minimal API, который упрощает создание API за счёт минимизации шаблонного кода (без явных контроллеров), сохраняя совместимость с экосистемой .NET.

Entity Framework Core [3] — кроссплатформенная ORM-система для .NET, позволяющая работать с реляционными базами данных через объекты. В проекте интегрирована с **PostgreSQL** [4] — объектно-реляционной СУБД, поддерживающей транзакции, масштабируемость и сложные запросы. Entity Framework Core обеспечивает управление схемой через миграции и выполнение LINQ-запросов, что ускоряет взаимодействие с данными.

React [5] — библиотека для создания интерактивных пользовательских интерфейсов, основанная на компонентном подходе. Для сборки фронтенда используется **Vite** [6] — инструмент, обеспечивающий быструю компиляцию и обновление интерфейса при изменениях в коде, что значительно ускоряет процесс разработки. Для работы с API используется **Axios** [7] — популярная библиотека для отправки HTTP-запросов, которая облегчает обработку асинхронных операций и работу с запросами и ответами от сервера.

Архитектура системы и структура базы данных

Перед началом реализации была спроектирована архитектура будущего веб-приложения (Рис. 2).

Приложение включает следующие компоненты:

- **Client** — представляет собой интерфейс пользователя и использует Axios Service для взаимодействия с сервером.
- **API** — отвечает за обработку поступающих HTTP-запросов и взаимодействие с компонентом данных, которое осуществляется при помощи

сервисов. Пакет Integrations содержит инструменты для работы с внешними системами (при помощи API сайта кафедры СП СПбГУ[1]) и документами. Поскольку средства интеграции структурно изолированы, адаптация веб-приложения под инфраструктуру другого университета может быть осуществлена с незначительными модификациями архитектуры при помощи замены соответствующих модулей.

- **Data** — отвечает за организацию и управление данными в приложении и включает несколько ключевых частей, обеспечивающих корректную работу с базой данных: сущности, конфигурации и репозитории.

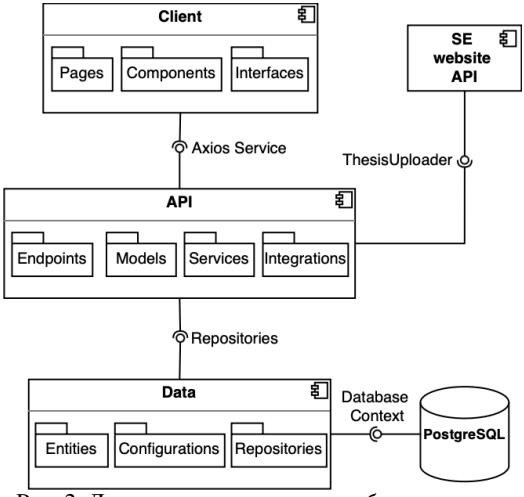


Рис. 2: Диаграмма компонентов веб-приложения

Также была определена структура базы данных (Рис. 3). Основные сущности:

- **User** — пользователь с указанием роли (Role).
- **Meeting** — заседание комиссии, имеет связь один ко многим с сущностью StudentWork (работа студента).
- **Criteria** — критерий оценивания, имеет связи многие ко многим с таблицей заседаний и один ко многим с таблицей правил (Rule).

[1]Кафедра системного программирования СПбГУ: `https://se.math.spbu.ru` (дата обращения: 2024-11-19)

- **MemberMark** — оценка члена комиссии, состоящая из нескольких CriteriaMark — оценок по критериям, каждая из которых содержит набор выбранных правил (SelectedRule). Для хранения усреднённой оценки работы по критерию определена сущность AverageCriteriaMark.

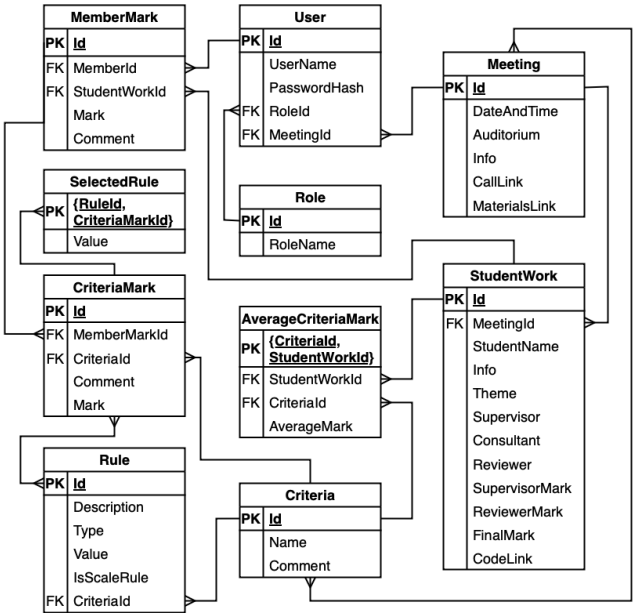


Рис. 3: ER-диаграмма базы данных

Особенности реализации

Веб-приложение позволяет создавать критерии, состоящие из правил двух видов — правила шкалы оценивания и дополнительные правила (штрафы и бонусы). Вторые, в свою очередь, обладают тремя типами: фиксированное число (представляется в виде checkbox'а), диапазон (отображается в виде ползунка) и произвольное число (позволяет ввести число).

Для обеспечения авторизации и аутентификации пользователей в системе используются JWT-токены (JSON Web Tokens). В токене зашифрованы данные о пользователе, включая его имя, роль и срок действия. Благодаря хранению роли пользователя в токене сервис может разграничивать права доступа между членами комиссии и администратором.

Поскольку в системе одновременно работают несколько пользователей, возникает необходимость в актуальном отображении изменений данных. Для этой цели была использована библиотека **SignalR**, упрощающая добавление функций реального времени в веб-приложения [8].

Завершающий этап защиты — скачивание таблиц с оценками и публикация успешно защищённых работ на сайте кафедры СП. Для этого был использован API для загрузки ВКР[2].

Для генерации вышеупомянутых таблиц с оценками, документов и автоматического создания заседаний из .xlsx-файла используется библиотека **NPOI** [9], предоставляющая высокоуровневый API для работы с файлами Excel и Word.

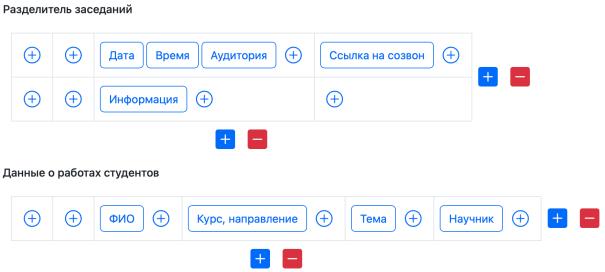


Рис. 4: Окно создания заседаний из файла

Необходимо было реализовать возможность загружать расписания различных форматов, поэтому веб-приложение предоставляет пользователю право динамически указывать структуру таблиц (Рис. 4). Таблица условно делится на два элемента — «разделитель заседаний», отделяющий их друг от друга и содержащий информацию о дате, времени, месте проведения и т.д., и «данные о работах студентов», определяющий их структуру. Приложение позволяет пользователю выбирать размер каждого элемента и указывать содержание ячеек, используя опции из выпадающего меню.

Тестирование, апробация, развёртывание

В ходе работы проводилось модульное, интеграционное и сквозное тестирование веб-приложения, процент покрытия кода тестами составил 74%.

Для проведения апробации веб-приложение запускалось локально на внешнем IP-адресе. В ходе использования инструмента во время зимних за-

[2]Официальный пример скрипта загрузки ВКР: `https://se.math.spbu.ru/files/upload.py` (дата обращения: 2025-03-11)

щит и пересдач учебных практик была получена обратная связь и учтены замечания. Пользователи отметили удобство интерфейса и интуитивность взаимодействия с системой.

На данный момент веб-приложение развёрнуто на удалённом сервере. Для обеспечения защищённого соединения были настроены **Nginx** и автоматическое получение SSL-сертификата с помощью **Certbot**.

Заключение

В рамках работы были сформулированы требования к веб-приложению, проведен обзор используемых технологий, спроектирована архитектура приложения и определена структура базы данных. Веб-приложение разработано, проведены его тестирование и апробация. Приложение развёрнуто и готово к внедрению в процесс защит ВКР на кафедре системного программирования СПбГУ.

Список литературы

[1] Приказ № 14453/1 от 13.11.2023 «О методическом обеспечении государственной итоговой аттестации в 2024 году (СВ.5162.*)» https://edu.spbu.ru/files/2023/20231113_14453_1.pdf (дата обращения: 2024-10-02)

[2] ASP.NET Core https://dotnet.microsoft.com/ru-ru/apps/aspnet (дата обращения: 2024-10-02)

[3] Entity Framework Core https://learn.microsoft.com/en-us/ef/core/ (дата обращения: 2024-11-18)

[4] PostgreSQL https://www.postgresql.org (дата обращения: 2024-11-18)

[5] React https://ru.react.dev (дата обращения: 2024-11-18)

[6] Vite https://vite.dev (дата обращения: 2024-11-18)

[7] Axios https://axios-http.com (дата обращения: 2024-11-18)

[8] Overview of ASP.NET Core SignalR https://learn.microsoft.com/en-gb/aspnet/core/signalr (дата обращения: 2025-03-11)

[9] NPOI https://github.com/nissl-lab/npoi (дата обращения: 2024-08-18)

Оптимизация системы управления очередью задач в Tokio runtime

Калашников М.М., СПБГУ, Санкт-Петербург kalashnikov.matvey.m@gmail.com,
Гориховский В.И., доцент кафедры системного программирования СПбГУ, к.ф.-м.н.,
Санкт-Петербург v.gorikhovskii@spbu.ru

Аннотация

Система поддержки периода исполнения Tokio для языка Rust предназначена для эффективного выполнения асинхронных задач в многопоточных приложениях. Однако при высокой нагрузке она сталкивается с ограничением масштабируемости, связанным с конкуренцией при распределении задач между потоками.

В этой работе рассматривается модификация Tokio, включающая дополнительную очередь задач, что позволяет снизить конкуренцию за общие ресурсы планировщика и сократить задержки при распределении задач между потоками.

Введение

Многопоточное программирование стало неотъемлемой частью разработки высокопроизводительных систем, позволяя выполнять множество операций одновременно, эффективно используя вычислительные ресурсы процессора. Это особенно важно для серверных приложений и систем хранения данных, которые подвергаются высокой нагрузке. Однако при работе с разделяемыми ресурсами, такими как память, файлы или сетевые соединения, возникает необходимость синхронизации доступа. Использование для этих целей блокирующей синхронизации, хотя и упрощает реализацию, может приводить к простою процессора и снижению производительности.

Для решения этой проблемы применяются асинхронные системы поддержки периода исполнения (runtime), которые позволяют эффективно управлять выполнением задач в многопоточных приложениях, минимизируя задержки, связанные с блокирующей синхронизацией. Они распределяют задачи между потоками и обеспечивают кооперативное переключение между ними, уменьшая накладные расходы на создание новых потоков и синхронизацию доступа к общим ресурсам. В языке Rust наиболее часто используемой системой поддержки периода исполнения является Tokio[1].

[1]Tokio: Asynchronous runtime for Rust — https://tokio.rs/tokio.

Основой эффективной работы асинхронной системы поддержки периода исполнения является механизм распределения задач между потоками-работниками. Планировщик в Tokio реализует множество оптимизаций, одной из которых является использование двух типов очередей для хранения задач. Помимо общей глобальной очереди, каждый поток-работник имеет свою локальную очередь для хранения задач[2]. Такая архитектура позволяет снизить конкуренцию за доступ к глобальной очереди и повысить степень параллелизма.

Тем не менее, в определенных сценариях исполнения многопоточных программ под управлением Tokio наблюдается снижение производительности при обработке большого количества коротких задач. Эта проблема была выявлена сотрудниками компании ООО «Ядро Центр Программных Разработок». При этом конкуренция за доступ к глобальной очереди задач становится узким местом, снижая производительность многопоточного приложения. Эта работа направлена на проектирование и внедрение улучшенного алгоритма управления очередями задач в Tokio, направленного на повышение производительности системы поддержки периода исполнения.

Устройство Tokio

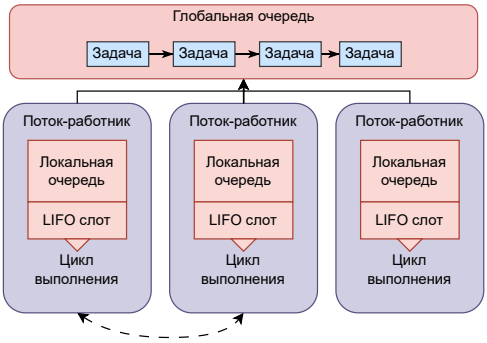


Рис. 1: Архитектура многопоточной системы поддержки периода исполнения

Tokio использует фиксированный пул потоков-работников. Количество потоков в пуле обычно определяется количеством ядер процессора, но может быть настроено пользователем. Планировщик задач Tokio организован

[2]Making the Tokio scheduler 10x faster — `https://tokio.rs/blog/2019-10-scheduler`.

следующим образом:

- каждый поток-работник имеет собственную локальную очередь задач;
- каждый поток-работник может получить блокирующий доступ к глобальной очереди;
- для балансировки нагрузки между потоками используется механизм кражи задач (work stealing).

Потоки-работники сначала обрабатывают задачи из своей локальной очереди (структура LIFO). При ее опустошении поток обращается к глобальной очереди, либо пытается украсть задачи у соседей. Задачи, создаваемые самим потоком-работником в процессе исполнения, сначала помещаются в его локальную очередь, а при ее переполнении вытесняются в глобальную.

Использование блокирующего доступа к глобальной очереди снижает производительность Tokio в сценариях с высокой нагрузкой, особенно при большом количестве короткоживущих задач. В таких условиях потоки пытаются одновременно получить доступ к глобальной очереди, что приводит к увеличению времени ожидания.

Архитектура решения

Для решения описанной проблемы была предложена модификация системы очередей Tokio, основанная на использовании дополнительной очереди задач. Такой подход позволяет снизить количество обращений к глобальной очереди и уменьшить расходы на синхронизацию. Исходный код модификации Tokio доступен в репозитории[3].

Дополнительная очередь выполняет роль буфера между потоками-работниками и глобальной очередью. Работа с дополнительной очередью состоит из следующих этапов.

1. **Вытеснение задач из локальной очереди**. Когда локальная очередь потока-работника переполняется, половина задач из нее переносится в глобальную очередь.
2. **Перенос задач в дополнительную очередь**. При достижении определенного порога задач в глобальной очереди группа задач из начала очереди переносится в дополнительную очередь.

[3] https://github.com/foreverjun/tokio/pull/2

3. **Приоритет дополнительной очереди**. При поиске следующей задачи для выполнения поток-работник сначала обращается к дополнительной очереди. Это позволяет снизить нагрузку на глобальную очередь.

4. **Обращение к глобальной очереди**. Если дополнительная очередь пуста, поток-работник обращается к глобальной очереди для получения задач.

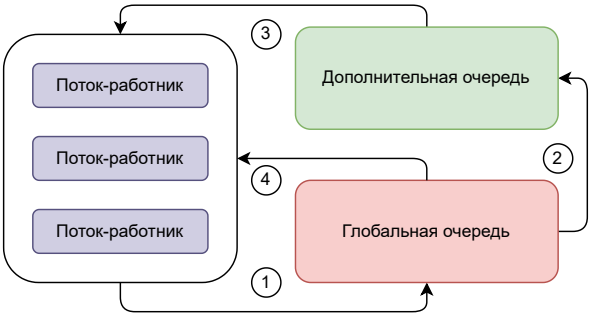


Рис. 2: Схема работы системы с дополнительной очередью

Выбор и реализация дополнительной очереди

Для реализации дополнительной очереди были рассмотрены структуры данных, использующие атомарные операции. Такой подход эффективнее при высокой конкуренции и снижает расходы на синхронизацию.

- **SegQueue** — очередь, состоящая из блоков, каждый из которых представляет собой массив из 31 элемента[4]. Использует `CAS` операции. Используется реализация из библиотеки `crossbeam`, активно применяющейся в Rust.

- **FAAArrayQueue** — неблокирующая очередь, состоящая из блоков фиксированного размера, где доступ к ячейкам синхронизируется с помощью атомарной инструкции `FAA` [1]. Были реализованы две версии с различными механизмами управления памятью: одна с использованием Epoch-Based Reclamation (EBR)[5], другая с использованием Hazard

[4]SegQueue. An unbounded multi-producer multi-consumer queue — `https://docs.rs/crossbeam/latest/crossbeam/queue/struct.SegQueue.html`.

[5]Crossbeam: Epoch-Based Reclamation — `https://docs.rs/crossbeam-epoch`.

Pointers[6].

- **BatchQueue** — упрощенная реализация неблокирующей очереди, поддерживающая пакетную обработку задач [2]. Основана на модифицированной версии классической очереди Michael–Scott [3].

Эксперимент

Сравнение очередей

Производительность очередей сравнивалась в сценарии «производитель–потребитель». Элементы передавались в очереди пакетами фиксированного размера (64 элемента). Пропускная способность измерялась для разных соотношений числа производителей и потребителей (производителей:потребителей). Эксперимент основан на подходе, описанном в статье [4]. Исходный код очередей и сценария тестирования доступен в репозитории[7].

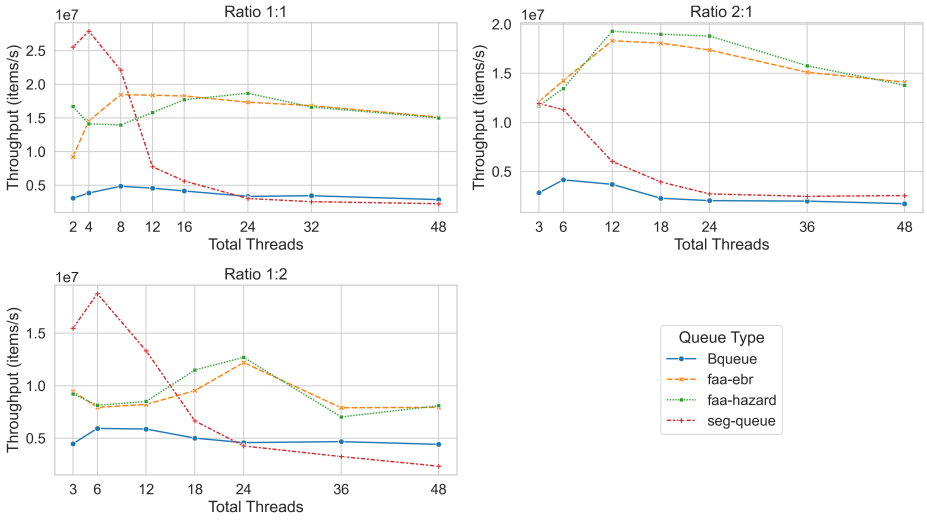


Рис. 3: Сравнение пропускной способности реализованных очередей и `SeqQueue`. Больше — лучше

[6]Haphazard: Hazard Pointers for Rust — `https://docs.rs/haphazard/0.1.8/haphazard/index.html`.

[7]`https://github.com/foreverjun/queue-bench`

По итогам сравнения наилучшую производительность при большом количестве потоков показала FAAArrayQueue. Заметим, что использование различных способов освобождения памяти (EBR и Hazard Pointers) практически не влияет на пропускную способность. Для интеграции в Tokio в качестве дополнительной очереди была выбрана FAAArrayQueue с реализацией на основе EBR, так как библиотека crossbeam широко используется в сообществе Rust, благодаря своей активной поддержке, длительной истории разработки и высокой надежности.

Сравнение модифицированного и оригинального Tokio

Для оценки производительности модификации использовался тестовый сценарий, основанный на реализации Игоря Ерина[8]. Его цель — имитировать ситуацию, при которой глобальная очередь заполняется большим количеством коротких процессорозависимых задач. Нагрузка создается задачами-производителями, каждая из которых генерирует по 7000 пустых задач. Эти задачи добавляются в локальные очереди потоков-работников и при переполнении вытесняются в глобальную очередь, моделируя высокую конкуренцию за доступ к ней. Такой подход позволяет нагружать как локальные, так и глобальную очереди.

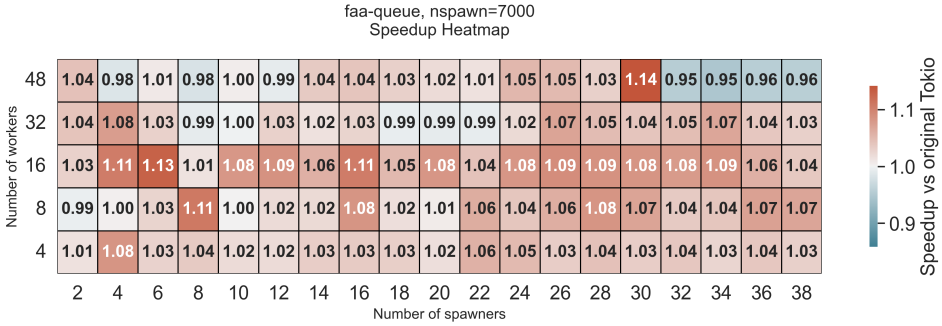


Рис. 4: Пропускная способность Tokio с `FAAArrayQueue`, относительно оригинала

На тепловой карте представлена относительная пропускная способность модифицированного Tokio по сравнению с оригинальной реализацией. Модифицированная версия показывает устойчивое увеличение пропускной способности практически для всех конфигураций, особенно для 16 потоков-работников, где достигается прирост до 13%. Даже в сценариях с высоким

[8] `https://github.com/foreverjun/tokiobench/tree/additional_q_bench`

числом потребителей модифицированная версия не уступает оригинальной в производительности.

Тестирование выполнялось на виртуальной машине Yandex Cloud с процессором Intel Xeon (Ice Lake) с 24 физическими и 48 виртуальными ядрами. Для стабильности приоритет выполнения был повышен до максимального (nice –20).

Заключение

В работе была спроектирована и реализована модификация системы поддержки периода исполнения Tokio с использованием дополнительной очереди задач. Проведен сравнительный анализ неблокирующих очередей. Лучшая реализация интегрирована в Tokio, в результате чего удалось добиться увеличения пропускной способности системы до 13% для целевых сценариев использования.

Список литературы

[1] P. Ramalhete, A. Correia. FAAArrayQueue — MPMC lock-free queue. Concurrency Freaks Blog. https://concurrencyfreaks.blogspot.com/2016/11/faaarrayqueue-mpmc-lock-free-queue-part.html.

[2] Gal Milman-Sela, Alex Kogan, Yossi Lev, Victor Luchangco, and Erez Petrank. 2022. BQ: A Lock-Free Queue with Batching. ACM Trans. Parallel Comput. 9, 1, Article 5 (March 2022), 49 pages. https://doi.org/10.1145/3512757

[3] Maged M. Michael and Michael L. Scott. 1996. Simple, fast, and practical non-blocking and blocking concurrent queue algorithms. In Proceedings of the fifteenth annual ACM symposium on Principles of distributed computing (PODC '96). Association for Computing Machinery, New York, NY, USA, 267–275. https://doi.org/10.1145/248052.248106

[4] Raed Romanov and Nikita Koval. 2023. The State-of-the-Art LCRQ Concurrent Queue Algorithm Does NOT Require CAS2. In Proceedings of the 28th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming (PPoPP '23). Association for Computing Machinery, New York, NY, USA, 14–26. https://doi.org/10.1145/3572848.3577485

Содержание

Организаторы конференции «Мат-мех. Наука 2025» благодарят спонсоров и партнёров конференции за неоценимую ресурсную и организационную помощь в проведении конференции и при подготовке сборника материалов.

В 2025 году нас поддержали:

Санкт-Петербургский государственный университет

Группа компаний «ЯДРО»

Научное издание

Материалы весенней научно-практической конференции по вопросам информатики, математики, механики и астрономии «Мат-мех. Наука 2025»

28 апреля – 3 мая 2025 г.
Санкт-Петербург

Компьютерная верстка: Д.В. Луцив

Издательство «ВВМ»
196247, г. Санкт-Петербург,
ул. Бассейная
д. 5, лит. А. кв. 52
E-mail: vvmpub@yandex.ru

Подписано в печать 04.07.2025. Формат 60 × 84 $^{1}/_{16}$.
Бумага офсетная. Гарнитура Times. Печать цифровая.
Усл. печ. л. 3,38. Тираж 100 экз. Заказ № 2516.

Отпечатано в Издательстве ВВМ.
196247, г. Санкт-Петербург, ул. Бассейная д. 5, лит. А. кв. 52.