УДК 004.75 : 004.415

МАТЕМАТИЧЕСКАЯ МОДЕЛЬ ПОТОКОВОГО МАСШТАБИРОВАНИЯ МИКРОСЕРВИСНЫХ ПРИЛОЖЕНИЙ В KUBERNETES

© 2026 М. Маццара, М.О. Филиппов

АНО ВО Университет Иннополис, г. Иннополис, Россия

Статья поступила в редакцию 23.01.2026

Рассматривается задача автоматического масштабирования микросервисных приложений, развернутых в контейнерных кластерах Kubernetes, в условиях нестационарной нагрузки и ограниченных вычислительных ресурсов. Предлагается потоковый метод автоматического масштабирования Flow-Autoscaler, основанный на математической формализации масштабирования как задачи минимальной стоимости потока на ориентированном графе. В отличие от подходов на основе машинного обучения, предложенный метод не требует этапа обучения и обеспечивает детерминированное и воспроизводимое поведение при заданных параметрах системы. Разработан алгоритм построения потоковой модели на основе телеметрических данных и реализован программный комплекс для автоматического применения решений масштабирования. Проведено экспериментальное сравнение с системами автоматического масштабирования на основе графовых нейронных сетей, использующими явный учет межсервисных зависимостей. Показано, что методы машинного обучения обеспечивают более высокое качество соблюдения соглашений об уровне обслуживания, тогда как Flow-Autoscaler позволяет существенно снизить затраты вычислительных ресурсов при умеренном снижении качества обслуживания.

Ключевые слова: микросервисные системы, Kubernetes, автоматическое масштабирование, математическое моделирование, оптимизация на графах, задача минимальной стоимости потока, качество обслуживания.

DOI: 10.37313/1990-5378-2026-28-2(2)-333-340
EDN: ECLLHI

ВВЕДЕНИЕ

Микросервисная архитектура является одним из основных подходов к построению масштабируемых распределенных информационных систем и широко применяется при разработке современных облачных приложений [1, 2]. Использование контейнерных технологий и оркестраторов, таких как Kubernetes, позволяет автоматизировать развертывание и масштабирование приложений, однако эффективность встроенных механизмов автоматического масштабирования остается ограниченной [3]. Стандартные средства горизонтального и вертикального масштабирования, как правило, опираются на локальные реактивные показатели и пороговые правила, не учитывающие структуру межсервисных взаимодействий и распространение нагрузки внутри приложения [6, 7].

В условиях нестационарного характера запросов и сложных зависимостей между сервисами такие подходы приводят к запаздывающим решениям, колебаниям числа экземпляров сервисов и избыточному резервированию ресурсов, что неоднократно отмечалось в обзорах и экспериментальных исследованиях [4, 5]. Это стимулировало развитие методов автоматического масштабирования, основанных на машинном обучении, в том числе на графовых нейронных сетях, позволяющих явно учитывать топологию микросервисного приложения и динамику нагрузки [8-11]. Подобные методы демонстрируют высокие показатели качества обслуживания, однако требуют этапов обучения и переобучения моделей, повышенных требований к наблюдаемости системы и значительных вычислительных затрат [11-13, 15].

Альтернативой обучаемым моделям являются методы, основанные на строгой математической постановке задачи управления ресурсами. В рамках данного направления распределение вычислительных ресурсов может быть формализовано как задача оптимизации на графах, что позволяет явно учитывать ограничения ресурсов и стоимость принимаемых решений [16, 17]. Классические методы теории потоков обладают хорошо изученными вычислительными свойствами и обеспечивают детерминированность и воспроизводимость решений, что делает их привлекательными для использования в задачах управления распределенными системами.

В данной работе предлагается потоковый подход к автоматическому масштабированию микросервисных приложений в контейнерных кластерах Kubernetes, в котором задача распределения

Мануэль Маццара, доктор технических наук, декан факультета компьютерных и инженерных наук. E-mail: m.mazzara@innopolis.university
Филиппов Максим Олегович, аспирант. E-mail: m.filippov@innopolis.university

ресурсов формализуется как задача минимальной стоимости потока на ориентированном графе. Целью настоящей работы является разработка и исследование алгоритма потокового масштабирования Flow-Autoscaler для микросервисных систем в кластерах Kubernetes. В статье предложена математическая модель масштабирования, описан алгоритм построения и решения потоковой задачи, а также представлена программная реализация разработанного метода. Проведено экспериментальное сравнение с системами автоматического масштабирования на основе графовых нейронных сетей, позволяющее проанализировать компромисс между качеством обслуживания и затратами вычислительных ресурсов [8-10, 13].

Практическая значимость рассматриваемой задачи обусловлена необходимостью выбора масштабируемых и воспроизводимых методов управления ресурсами в реальных вычислительных кластерах, где объем доступных телеметрических данных, вычислительные ресурсы и требования к стабильности поведения системы могут быть ограничены [18]. В таких условиях использование обучаемых моделей автоматического масштабирования не всегда является оправданным, и возникает потребность в методах, обеспечивающих прозрачную интерпретацию принимаемых решений и предсказуемость их последствий. Сравнение потокового оптимизационного подхода с методами автоматического масштабирования на основе графовых нейронных сетей позволяет не только оценить выигрыш в качестве обслуживания, но и проанализировать издержки, связанные с усложнением модели и ростом вычислительных затрат, что имеет принципиальное значение при проектировании и эксплуатации микросервисных систем.

МЕТОДИКА ИССЛЕДОВАНИЙ

В рамках исследования микросервисное приложение рассматривается как распределенная система взаимосвязанных компонентов, функционирующих в контейнерном кластере Kubernetes и обменивающихся потоками запросов [1, 2, 18]. Структура приложения описывается ориентированным графом

$$G = (V, E), \quad V = \{s_1, s_2, \dots, s_N\}, \quad E \subseteq V \times V,$$

вершины которого соответствуют отдельным сервисам, а дуги отражают направление передачи нагрузки между ними. Такое представление позволяет формализовать распространение запросов и задержек по цепочкам межсервисных взаимодействий и учитывать влияние локальных решений масштабирования на глобальное состояние системы [4, 5].

Каждому сервису сопоставляются текущие характеристики нагрузки и использования ресурсов, получаемые из системы мониторинга, включая интенсивность запросов, потребление процессорного времени и памяти, а также показатели задержки, используемые для контроля соблюдения соглашений об уровне обслуживания [6, 7]. Масштабирование сервисов осуществляется за счет изменения числа экземпляров сервисов и параметров выделяемых ресурсов контейнеров в пределах допустимых ограничений, определяемых конфигурацией кластера и политиками оркестратора.

Задача автоматического масштабирования формулируется как задача оптимального распределения вычислительных ресурсов между сервисами при заданной нагрузке. Для этого используется потоковая модель, в которой вычислительные ресурсы трактуются как ограниченный поток, распределяемый между сервисами с учетом их потребностей и стоимости использования ресурсов. Узлы кластера рассматриваются как источники ресурсов с ограниченной емкостью, а сервисы — как потребители, предъявляющие требования к пропускной способности и качеству обслуживания, что соответствует классическим постановкам задач оптимизации потоков в сетях [16, 17].

Математически задача масштабирования сводится к задаче минимальной стоимости потока на ориентированном графе

$$\min_f \sum_{(i,j)\in E} c_{ij} f_{ij},$$

где f_{ij} — поток ресурсов, а c_{ij} — стоимость их использования, в которой требуется обеспечить удовлетворение суммарного спроса сервисов на ресурсы при минимальных затратах.

$$0 \le f_{ij} \le C_{ij}, \quad \sum_j f_{ij} - \sum_j f_{ji} = b_i,$$

где C_{ij} – доступные ресурсы узлов кластера,

b_i – потребность сервиса в ресурсах.

В модели учитываются ограничения на пропускную способность, соответствующие доступным вычислительным ресурсам узлов, а также весовые коэффициенты, отражающие относительную стоимость выделения ресурсов. Такое представление позволяет одновременно учитывать как технические ограничения системы, так и экономические аспекты масштабирования [16].

Решение задачи минимальной стоимости потока осуществляется с использованием классических численных методов оптимизации на графах, обладающих полиномиальной вычислительной

сложностью. Найденное оптимальное распределение потока интерпретируется в терминах числа экземпляров сервисов и параметров вертикального масштабирования, которые затем преобразуются в команды управления Kubernetes. Благодаря детерминированному характеру используемой оптимизационной постановки алгоритм обеспечивает воспроизводимость решений при одинаковых входных данных и параметрах модели.

Разработанный метод реализован в виде программного комплекса, интегрированного с системой мониторинга Prometheus и API Kubernetes [6, 7]. Архитектура реализации обеспечивает автоматический сбор телеметрических данных, построение потоковой модели и применение вычисленных решений масштабирования в рамках стандартного цикла управления кластером. Исходный код и конфигурации экспериментов доступны в открытом репозитории [19], что обеспечивает воспроизводимость полученных результатов и возможность использования предложенного подхода в дальнейших исследованиях.

Для обеспечения устойчивости масштабирования в условиях кратковременных всплесков нагрузки и шумов телеметрических данных в алгоритме используется агрегирование метрик по скользящему временному окну. Это позволяет сгладить краткосрочные колебания и формировать решения масштабирования на основе устойчивых оценок интенсивности запросов и потребления ресурсов [11, 13]. Такой подход снижает вероятность избыточных колебаний числа экземпляров сервисов и обеспечивает более стабильную работу системы при нестационарном характере нагрузки.

Параметры потоковой модели, включая потребности сервисов в ресурсах и ограничения узлов кластера, пересчитываются на каждом шаге управления на основе актуальных данных мониторинга. Полученные решения масштабирования применяются с учетом дискретной природы ресурсов и ограничений оркестратора Kubernetes, что обеспечивает корректную интеграцию оптимизационного алгоритма в стандартный цикл управления кластером. Таким образом, предложенная методика сочетает строгую математическую постановку задачи масштабирования с практической реализуемостью и применимостью в реальных микросервисных системах.

РЕЗУЛЬТАТЫ И ОБСУЖДЕНИЕ

Экспериментальные исследования проводились на тестовом Kubernetes-стенде с воспроизводимыми сценариями нестационарной нагрузки, включающими как плавные изменения интенсивности запросов, так и кратковременные всплески [3, 6]. Для оценки эффективности предложенного потокового метода автоматического масштабирования Flow-Autoscaler выполнено сравнение с системами автоматического масштабирования на основе графовых нейронных сетей, ориентированными на явный учет межсервисных зависимостей и динамики нагрузки [8-10, 13]. В качестве таких реализаций использовались открытые программные комплексы GAT-Burst, TGAT и HGT, функционирующие в идентичных условиях эксперимента [20-22].

Для формирования параметров потоковой модели и анализа качества обслуживания использовались агрегированные показатели мониторинга, собираемые системой Prometheus. В их число входят относительная загрузка процессора и оперативной памяти контейнеров, интенсивность входящих запросов и 95-й процентиль задержки обработки запросов.

Важным аспектом экспериментальной методики является способ измерения задержки обработки запросов, используемой для оценки соблюдения соглашений об уровне обслуживания (SLO). В рамках настоящего исследования задержка определяется на уровне микросервисов как время между моментом поступления запроса в сервис и моментом формирования ответа, включая все внутренние вычисления и обращения к зависимым сервисам.

Измерение задержки осуществляется с использованием стандартного механизма гистограмм в системе мониторинга Prometheus. Для каждого сервиса фиксируется распределение времени обработки запросов в виде гистограммы ***requests_duration_in_seconds***, агрегируемой по меткам сервиса. На основе этой гистограммы вычисляется 95-й процентиль задержки с использованием функции:

$$L_{0.95}(t) = \text{histogram_quantile}\left(0.95,\ \sum_i \text{rate}\left(b_i(t, \Delta t)\right)\right),$$

где $b_i(t, \Delta t)$ — число наблюдений в *i*-м интервале гистограммы за скользящее временное окно Δt, а суммирование выполняется по всем интервалам гистограммы.

В экспериментах использовалось скользящее окно длительностью 1 минута, что соответствует стандартным практикам мониторинга в Kubernetes и обеспечивает баланс между чувствительностью к кратковременным всплескам нагрузки и устойчивостью оценки [6, 7]. Для анализа долгосрочной динамики значения 95-го перцентиля дополнительно усреднялись по интервалам принятия решений системы автоматического масштабирования, равным 5 минутам.

Следует отметить, что использование процентильных показателей, в отличие от средних значений, позволяет учитывать хвостовые задержки, критичные для пользовательского опыта и соблюдения SLO в микросервисных системах. При этом выбранный уровень 95-го процентиля отражает компромисс между устойчивостью оценки и чувствительностью к экстремальным значениям задержки, что соответствует общепринятой практике в исследованиях автоматического масштабирования [5, 11, 13].

Указанные показатели применяются как для оценки текущего состояния микросервисов и вычислительных узлов, так и для анализа соблюдения соглашений об уровне обслуживания, что соответствует общепринятым практикам мониторинга и управления микросервисными системами [6, 7]. Перечень используемых показателей и их назначение приведены в табл. 1.

Таблица 1. Показатели мониторинга, используемые для построения потоковой модели и принятия решений масштабирования

Показатель	**Prometheus-показатель (агрегация)**	**Физический смысл**	**Назначение в экспериментах**
CPU utilization	rate(container_cpu_usage_seconds_total) / (container_spec_cpu_quota / container_spec_cpu_period)	Относительная загрузка процессора контейнеров, %	Оценка вычислительной нагрузки сервисов и ограничений узлов при масштабировании
Memory utilization	container_memory_usage_bytes / container_spec_memory_limit_bytes	Относительное использование оперативной памяти контейнеров, %	Контроль потребления памяти и предотвращение превышения лимитов контейнеров
Requests per second (RPS)	rate(requests_total_by_service[1m])	Интенсивность входящих запросов, запросов/с	Формирование спроса сервисов на ресурсы в потоковой модели
p95 latency	histogram_quantile(0.95, requests_duration_in_seconds_by_service_bucket)	95-й процентиль задержки обработки запросов	Контроль соблюдения соглашений об уровне обслуживания (SLO)

Сравнение методов автоматического масштабирования проводилось по показателям доли нарушений соглашений об уровне обслуживания (SLO) по задержке, средней величине 95-го процентиля задержки, суммарным затратам вычислительных ресурсов и числу событий масштабирования за рассматриваемый период. Использование указанных показателей является стандартным при оценке эффективности автоматического масштабирования и отражает как качество обслуживания, так и экономичность использования ресурсов [5, 11, 13]. Численные значения указанных показателей, усредненные по результатам длительного эксперимента, приведены в табл. 2.

Таблица 2. Сравнение показателей эффективности различных методов масштабирования

Метод масштабирования	**Доля нарушений SLO, %**	**p95-задержка, мс**	**Нормализованные затраты ресурсов**	**Число масштабирований**
Flow-Autoscaler	3,9	185	**0,65**	**42**
GAT-Burst	**2,5**	**160**	0,78	57
TGAT	2,8	168	0,88	61
HGT	2,9	172	0,92	64

Дополнительно выполнено длительное тестирование рассматриваемых методов на горизонте 30 дней при воспроизводимых профилях нестационарной нагрузки. Динамика доли нарушений SLO за весь период наблюдения приведена на рис. 1. Видно, что система автоматического масштабирования GAT-Burst демонстрирует наименьшие значения доли нарушений SLO на протяжении

всего эксперимента, что указывает на более точную адаптацию к изменяющимся условиям нагрузки и эффективный учет межсервисных зависимостей, характерный для графовых нейронных сетей [8–10]. При этом преимущество по сравнению с Flow-Autoscaler остается умеренным и наиболее отчетливо проявляется в дни с выраженными всплесками нагрузки. Методы TGAT и HGT показывают близкие траектории и в рассматриваемых условиях не обеспечивают существенного выигрыша по сравнению с GAT-Burst.

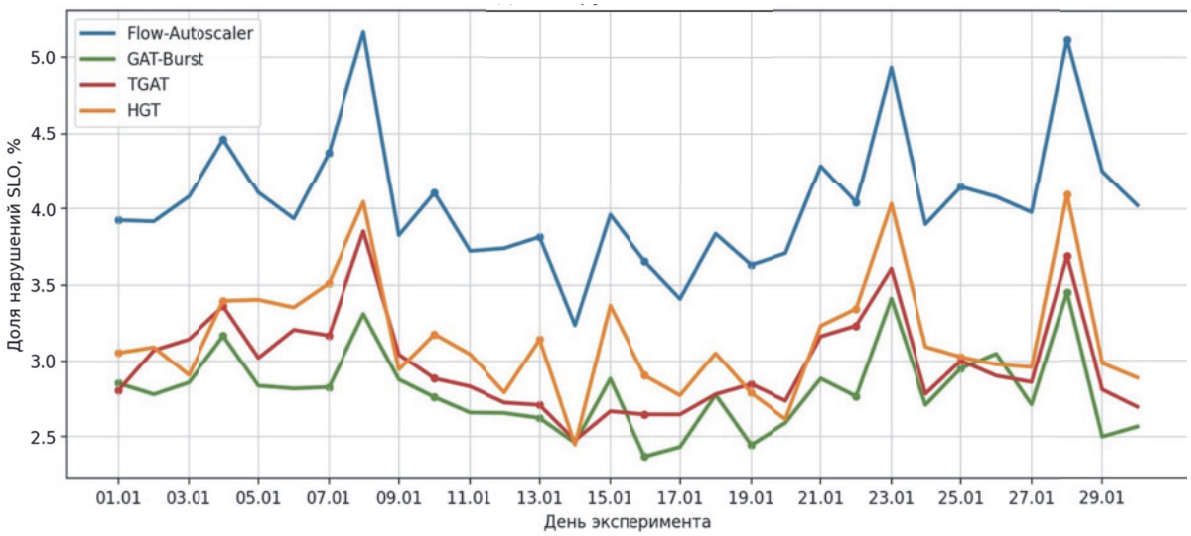

Рис. 1. Доля нарушений SLO за 30 дней.

Динамика нормализованных затрат вычислительных ресурсов за 30 дней представлена на рис. 2. Flow-Autoscaler обеспечивает минимальные затраты ресурсов на всем интервале наблюдения, что подтверждает экономическую направленность предложенной оптимизационной постановки и отсутствие необходимости в избыточном резервировании ресурсов, характерном для обучаемых методов [14, 15]. В то же время GAT-Burst демонстрирует более высокие затраты по сравнению с Flow-Autoscaler, что может рассматриваться как плата за более строгое соблюдение соглашений об уровне обслуживания. Системы автоматического масштабирования TGAT и HGT характеризуются наибольшими затратами ресурсов, что согласуется с их смещением в сторону более ресурсоемких стратегий масштабирования.

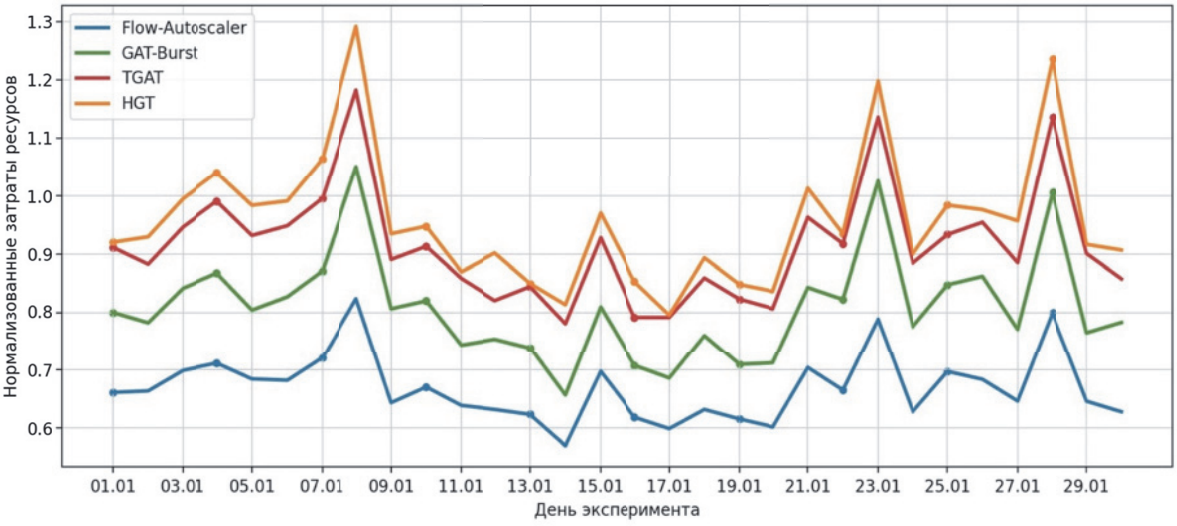

Рис. 2. Нормализованные затраты ресурсов за 30 дней

На рис. 3 приведена динамика 95-го процентиля задержки обработки запросов за 30 дней. Результаты показывают, что GAT-Burst обеспечивает наименьшие значения p95-задержки, в том числе в периоды кратковременных всплесков нагрузки. Flow-Autoscaler демонстрирует несколько более высокие значения задержки, однако разница по сравнению с лидирующим методом остается умеренной и не приводит к резкому ухудшению качества обслуживания. Таким образом, длительное наблюдение подтверждает устойчивость выявленного компромисса между качеством обслуживания и экономичностью масштабирования [5, 11].

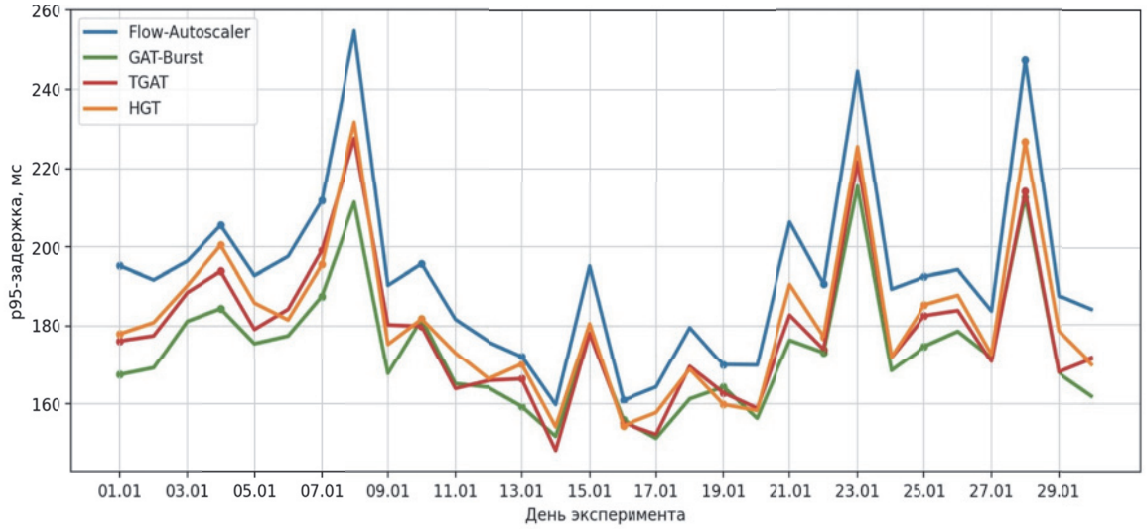

Рис. 3. p95-задержка за 30 дней

Интегральная оценка соотношения «качество–стоимость», вычисленная по средним значениям показателей за 30 дней, представлена на рис. 4. Диаграмма наглядно демонстрирует наличие компромисса между минимизацией задержек и затратами вычислительных ресурсов [5, 13, 15]. GAT-Burst соответствует области более высокого качества обслуживания при повышенной стоимости, тогда как Flow-Autoscaler располагается в области минимальных затрат ресурсов при умеренном снижении качества обслуживания. Методы TGAT и HGT в рассматриваемых условиях оказываются смещены в область более высоких затрат без пропорционального выигрыша по качеству.

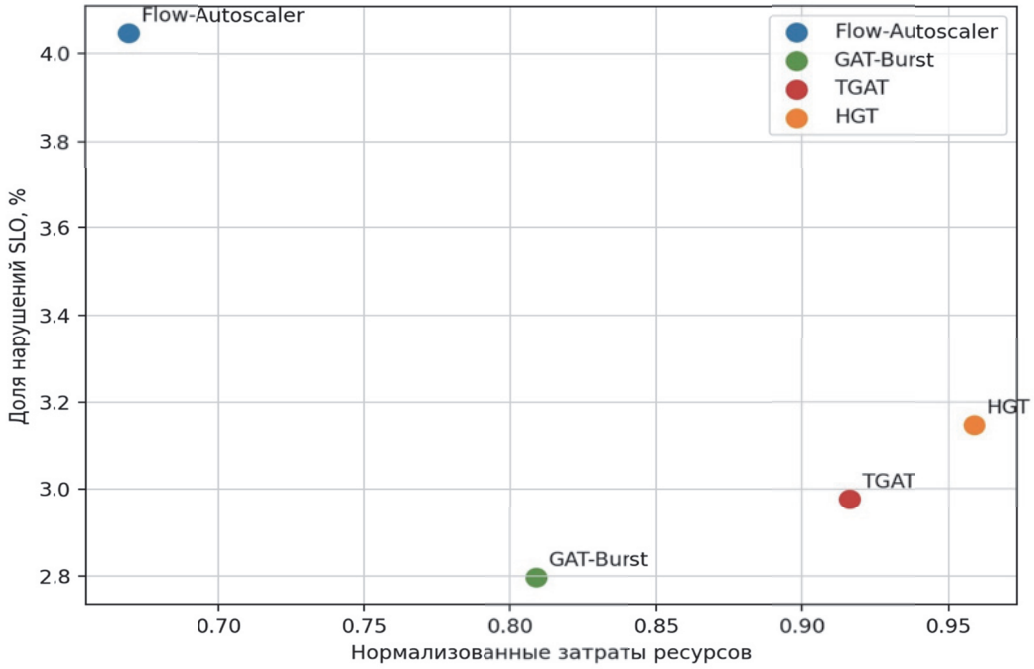

Рис. 4. Соотношение «качество–стоимость» (30 дней)

В совокупности результаты табл. 1, табл. 2 и рис. 1–4 подтверждают, что потоковый оптимизационный подход представляет собой практически применимое и воспроизводимое решение для экономически эффективного масштабирования микросервисных систем. Flow-Autoscaler может рассматриваться как базовое решение для управления ресурсами в условиях ограниченного бюджета и повышенных требований к стабильности поведения системы, тогда как использование GAT-Burst целесообразно в сценариях, где приоритетом является более строгое соблюдение соглашений об уровне обслуживания при допустимом росте затрат ресурсов и частоты управляющих воздействий [19-22].

ЗАКЛЮЧЕНИЕ

В работе предложен потоковый подход к автоматическому масштабированию микросервисных приложений в контейнерных кластерах Kubernetes, основанный на математической формализации задачи распределения вычислительных ресурсов в виде задачи минимальной стоимости потока. Разработанный метод автоматического масштабирования Flow-Autoscaler не использует обучаемые модели и обеспечивает детерминированное и воспроизводимое поведение при заданных параметрах системы, что особенно важно в условиях ограниченных вычислительных ресурсов и требований к предсказуемости работы распределенных приложений.

Проведенные экспериментальные исследования, включающие длительное тестирование на горизонте 30 дней при воспроизводимых профилях нестационарной нагрузки, показали, что автоскейлеры на основе графовых нейронных сетей, в частности GAT-Burst, обеспечивают более высокие показатели соблюдения соглашений об уровне обслуживания, прежде всего по метрике задержки обработки запросов. Вместе с тем предложенный потоковый метод демонстрирует существенно меньшие затраты вычислительных ресурсов и более стабильное поведение, выражающееся в снижении числа управляющих воздействий и отсутствии резких колебаний конфигурации системы при кратковременных всплесках нагрузки.

Анализ динамики показателей мониторинга на протяжении всего периода эксперимента показал устойчивость выявленного компромисса между качеством обслуживания и экономичностью масштабирования. Flow-Autoscaler обеспечивает умеренное увеличение задержек по сравнению с лидирующим нейросетевым методом, оставаясь при этом в допустимых пределах и позволяя значительно сократить суммарное потребление ресурсов. Это делает предложенный подход особенно привлекательным для практических сценариев эксплуатации микросервисных систем, в которых приоритетом являются контролируемые затраты, воспроизводимость результатов и простота сопровождения.

Полученные результаты подтверждают целесообразность применения классических методов оптимизации на графах для задач управления ресурсами микросервисных систем и демонстрируют, что такие методы могут успешно конкурировать с более сложными обучаемыми моделями в реальных условиях эксплуатации. Предложенный Flow-Autoscaler может использоваться как самостоятельное средство автоматического масштабирования, а также как базовое оптимизационное решение и эталон при сравнительном анализе и разработке более сложных методов автоматического масштабирования, ориентированных на учет межсервисных зависимостей и динамики нагрузки.

СПИСОК ЛИТЕРАТУРЫ

1. *Dragoni N., Lanese I., Larsen S. T., et al.* Microservices: Yesterday, Today, and Tomorrow // Present and Ulterior Software Engineering. Berlin: Springer, 2017. P. 195–216.
2. *Newman S.* Building Microservices. 2nd ed. Sebastopol: O'Reilly Media, 2021. 600 p.
3. *Burns B., Grant B., Oppenheimer D., Brewer E., Wilkes J.* Borg, Omega, and Kubernetes // Communications of the ACM. 2016. Vol. 59, No. 5. P. 50–57. DOI: 10.1145/2890784.
4. *Zhang Q., Chen M., Li L., et al.* A Survey of Microservice Architecture: From Design Principles to Deployment Challenges // *IEEE Access*. 2021. Vol. 9. P. 108033–108050. DOI: 10.1109/ACCESS.2021.3100867.
5. *Filippov M., Mazzara M.* Autoscaling Containerized Microservices: A Survey // International Conference on Advanced Information Networking and Applications: Proceedings of the International Conference on Advanced Information Networking and Applications (AINA), Vol. 7 / ed. by L. Barolli. Cham: Springer Nature Switzerland, 2025. P. 326–334.
6. *Kubernetes Authors.* Horizontal Pod Autoscaler. URL: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/ (дата обращения: 22.01.2026).
7. *Kubernetes Authors.* Vertical Pod Autoscaler. URL: https://github.com/kubernetes/autoscaler/tree/master/vertical-pod-autoscaler (дата обращения: 22.01.2026).
8. *Veličković P., Cucurull G., Casanova A., et al.* Graph Attention Networks // International Conference on Learning Representations (ICLR). 2018. DOI: 10.48550/arXiv.1710.10903.
9. *Xu D., Ruan C., Korpeoglu E., et al.* Inductive Representation Learning on Temporal Graphs // International Conference on Learning Representations (ICLR). 2020. DOI: 10.48550/arXiv.2002.07962.
10. *Hu Z., Dong Y., Wang K., Sun Y.* Heterogeneous Graph Transformer // Proceedings of the Web Conference (WWW). 2020. P. 2704–2710. DOI: 10.1145/3366423.3380027.
11. *Gheibi O., Weyns D., Quin F.* Applying Machine Learning in Self-Adaptive Systems: A Systematic Literature Review // ACM Transactions on Autonomous and Adaptive Systems (TAAS). 2021. Vol. 15. No. 3. P. 1–37. DOI: 10.1145/3469440.
12. *Aslanpour M. S., Dashti S. E.* SLA-aware Resource Allocation for Application Service Providers in the Cloud // Proceedings of 2016 Second International Conference on Web Research (ICWR). Piscataway: IEEE, 2016. P. 31–42. DOI: 10.1109/ICWR.2016.7498443.

13. *Khaleq A. A., Ra I.* Intelligent Autoscaling of Microservices in the Cloud for Real-Time Applications // IEEE Access. 2021. Vol. 9. P. 35464–35476. DOI: 10.1109/ACCESS.2021.3061890.
14. *Duc T. L., Nguyen C., Östberg P. O.* Workload Prediction for Proactive Resource Allocation in Large-Scale Cloud-Edge Applications // Electronics. 2025. Vol. 14. No. 16. P. 3333. DOI: 10.3390/electronics14163333.
15. *Zhang Y. et al.* Predictive Auto Scaling and Cost Optimization Using Machine Learning in AWS Cloud Environments // Proceedings of the 2nd International Symposium on Integrated Circuit Design and Integrated Systems. 2025. P. 161–167. DOI: 10.1145/3772326.3774726.
16. *Goldberg A. V., Tarjan R. E.* Finding Minimum-Cost Circulations by Successive Approximation // Mathematics of Operations Research. 1987. Vol. 15, No. 3. P. 430–466. DOI: 10.1287/moor.15.3.430.
17. *Ahuja R. K., Magnanti T. L., Orlin J. B.* Network Flows: Theory, Algorithms, and Applications. Englewood Cliffs: Prentice Hall, 1993. 846 p.
18. *Jamshidi P., Casale G., Cheung S., et al.* Microservices: The Journey So Far and Challenges Ahead // IEEE Software. 2018. Vol. 35, No. 3. P. 24–35. DOI: 10.1109/MS.2018.2141039.
19. *Flow-Autoscaler.* Open-source implementation. URL: https://github.com/MaxOF/flow-autoscaler (дата обращения: 22.01.2026).
20. *GAT-Burst Autoscaler.* Open-source implementation. URL: https://github.com/MaxOF/gat-burst-autoscaler (дата обращения: 22.01.2026).
21. *TGAT Autoscaler.* Open-source implementation. URL: https://github.com/MaxOF/tgat-autoscaler (дата обращения: 22.01.2026).
22. *HGT Autoscaler.* Open-source implementation. URL: https://github.com/MaxOF/hgt-autoscaler (дата обращения: 22.01.2026).

A MATHEMATICAL MODEL OF FLOW-BASED AUTOSCALING FOR MICROSERVICE APPLICATIONS IN KUBERNETES

© 2026 M. Mazzara, M. O. Filippov

University Innopolism, Innopolis, Russia

This paper addresses the problem of automatic autoscaling of microservice applications deployed in Kubernetes container clusters under non-stationary workloads and limited computational resources. A flow-based autoscaler, referred to as Flow-Autoscaler, is proposed. The method is based on a mathematical formulation of autoscaling as a minimum-cost flow problem on a directed graph. In contrast to machine learning–based approaches, the proposed method does not require a training phase and provides deterministic and reproducible behavior for given system parameters. An algorithm for constructing the flow model from telemetry data is developed, and a software system for automatic application of scaling decisions is implemented. An experimental comparison with autoscalers based on graph neural networks that explicitly account for inter-service dependencies is conducted. The results show that machine learning–based methods achieve higher service-level objective compliance, whereas Flow-Autoscaler significantly reduces computational resource costs with a moderate decrease in service quality.

Keywords: microservice systems, Kubernetes, autoscaling, mathematical modeling, graph optimization, minimum-cost flow problem, quality of service.

DOI: 10.37313/1990-5378-2026-28-2(2)-333-340
EDN: ECLLHI

Manuel Mazzara, PhD Computer Science, Dean of the Faculty of Computer Science and Engineering.
E-mail: m.mazzara@innopolis.university
Maxim Olegovich Filippov, Graduate Student. E-mail: m.filippov@innopolis.university