УДК 004.75:004.415.53 DOI 10.21777/2500-2112-2026-1-113-121

МЕТОДЫ АДАПТИВНОЙ ТРАССИРОВКИ РАСПРЕДЕЛЁННЫХ ПРИЛОЖЕНИЙ НА ОСНОВЕ САМООБУЧАЮЩЕГОСЯ СЕМПЛИРОВАНИЯ

Зубков Михаил Витальевич[1],
e-mail: zubkovmihail12345@gmail.com

Макеев Павел Сергеевич[1],
e-mail: kevin123321@gmail.com
[1]*«МИРЭА – Российский технологический университет», Институт информационных технологий, г. Москва, Россия*

Современные микросервисные приложения порождают значительный объём телеметрических данных, необходимых для обеспечения наблюдаемости и быстрой диагностики инцидентов. Однако полная трассировка всех запросов становится экономически нецелесообразной из-за накладных расходов на сбор, хранение и обработку информации. В статье рассматриваются методы адаптивного семплирования распределённой трассировки, позволяющие динамически регулировать объём собираемых данных в зависимости от текущего состояния системы и характеристик запросов. Предложена архитектура системы трассировки, включающая компонент самообучающегося выбора трассируемых запросов на основе анализа метрик и признаков запросов. Представлен алгоритм, сочетающий системные и поведенческие критерии оценки значимости запросов. Экспериментальное моделирование показало, что предложенный подход позволяет охватить до 90 % аномальных случаев при сокращении объёма телеметрии более чем в шесть раз по сравнению с полным сбором. Результаты демонстрируют перспективность адаптивного семплирования для повышения эффективности наблюдаемости микросервисных систем без увеличения инфраструктурной нагрузки.

Ключевые слова: микросервисы, наблюдаемость, распределённая трассировка, семплирование, телеметрия, самообучающийся алгоритм, адаптивный мониторинг, машинное обучение

ADAPTIVE TRACING METHODS FOR DISTRIBUTED APPLICATIONS BASED ON SELF-LEARNING SAMPLING

Zubkov M.V.[1],
e-mail: zubkovmihail12345@gmail.com

Makeev P.S.[1],
e-mail: kevin123321@gmail.com
[1]*"MIREA – Russian Technological University", Institute of Information Technology, Moscow, Russia*

Modern microservice applications generate a significant amount of telemetry data necessary to ensure the observability and rapid diagnosis of incidents. However, full tracing of all requests becomes economically impractical due to the overhead of collecting, storing, and processing information. The article discusses methods of adaptive sampling of distributed tracing, which allow dynamically adjusting the amount of data collected depending on the current state of the system and query characteristics. The architecture of the tracing system is proposed, which includes a component of self-learning selection of traceable queries based on the analysis of metrics and query features. An algorithm is presented that combines system and behavioral criteria for evaluating the significance of queries. Experimental modeling has shown that the proposed approach can cover up to 90 % of abnormal cases while reducing the volume of telemetry by more than six times compared to the full collection. The results demonstrate the promise of adaptive sampling for increasing the efficiency of observability in microservice systems without increasing infrastructure load.

Keywords: microservices, observability, distributed tracing, sampling, telemetry, self-learning algorithm, adaptive monitoring, machine learning

Введение

Современные программные системы всё чаще строятся по микросервисной архитектуре, которая повышает гибкость и масштабируемость, но усложняет мониторинг и понимание работы системы [1]. В распределённых приложениях, состоящих из множества взаимодействующих сервисов, традиционные методы наблюдения испытывают затруднения с локализацией проблем и узких мест. Для обеспечения надёжности таких систем необходима наблюдаемость – способность внешними измерениями полно и точно судить о внутренних состояниях сервисов. Она достигается сбором трёх основных видов телеметрии: метрик, логов и трейсов. Метрики отражают количественные показатели, логи содержат события работы, а трассировка позволяет проследить путь запроса через множество сервисов.

Особое значение приобретает распределённая трассировка – методика профилирования и диагностики, при которой каждому запросу присваивается уникальный идентификатор трассы, распространяемый между сервисами. В процессе выполнения запрос помечается на каждом шаге, благодаря чему лог-сообщения и метрики группируются в единую цепочку – трейс. Такой подход открывает широкие возможности для анализа: разработчики и инженеры поддержки получают контекст вызовов и задержек, что существенно облегчает поиск причин сбоев и деградации производительности. Однако высокая детализация сопровождается резким ростом объёма данных. Современные микросервисные системы генерируют огромные потоки телеметрии – до десятков петабайт в сутки, что затрудняет их хранение и обработку. Полный сбор всех трассировок становится непомерно ресурсоёмким и дорогостоящим. Возникает противоречие между стремлением к полной наблюдаемости и необходимостью ограничивать накладные расходы.

Одним из ключевых направлений решения данной проблемы является применение семплирования – выборочной трассировки. Оно предполагает отбор части запросов для трассировки, позволяя получить репрезентативную картину системы без избыточного объёма данных. Грамотно выбранная выборка трейсов способна отразить поведение системы и выявить проблемы при существенно меньших затратах. В условиях растущей сложности микросервисов актуальной задачей становится разработка методов адаптивного семплирования, автоматически подстраивающих уровень трассировки под текущую ситуацию. В данной работе предлагается архитектура системы адаптивной трассировки на основе самообучающегося семплирования, снижающая объём телеметрии при сохранении её информативности.

Обзор подходов к трассировке и семплированию в распределённых системах

Современные системы трассировки развились от внутренних корпоративных решений до зрелых *open-source* и облачных инструментов. Платформа Google Dapper заложила принципы масштабируемой трассировки с минимальными накладными расходами. На её основе появились проекты Jaeger, Zipkin, OpenTelemetry, а также облачные решения AWS X-Ray, Dynatrace и другие. Трассировка реализуется через библиотечное инструментирование сервисов или с помощью прокси-агентов.

Стандарты OpenTracing и OpenTelemetry обеспечивают унифицированный формат сбора телеметрии и поддержку интеграции в масштабируемые микросервисные среды.

При реальной эксплуатации полный сбор трасс невозможен из-за огромного объёма данных. В связи с этим используются различные стратегии семплирования, основными из которых являются:

– **Равномерное (*head-based*)** – отбор фиксированной доли запросов, простой и быстрый, но уязвим к пропуску редких событий.

– **Смещённое (*biased*)** – выбор по метаданным запроса или другим критериям.

– **Хвостовое (*tail-based*)** – анализ запроса после завершения, подходит для фиксации ошибок и долгих откликов.

– **Комбинированное** – гибкое сочетание нескольких стратегий в зависимости от ситуации.

– **Интеллектуальное/адаптивное** – динамическое изменение стратегии на основе текущей нагрузки и состояния системы, с возможностью самообучения моделей.

В таблице 1 приведено сравнение стратегий семплирования, оговоренных выше.

Таблица 1 – Сравнение стратегий семплирования

Метод	Момент принятия решения	Критерий отбора	Преимущества	Ограничения
Head-based	До обработки запроса	Случайная вероятность	Простота, минимальные издержки	Не гарантирует попадание аномалий
Tail-based	После завершения запроса	Статус, задержка	Фиксирует ошибки, высокоинформативные трассы	Требует буферизации, позднее решение
Biased по атрибутам	До или после	Метки, тип операции	Гибкость настройки, фокус на критичном	Риски недоучёта новых аномалий
Rate-limiting	В реальном времени	Ограничение частоты	Стабильность объёма данных	Не учитывает полезность конкретных трасс
Адаптивное	В реальном времени	Метрики, признаки запроса	Высокая точность, самоподстройка	Сложность реализации, необходимость обучения

Этот обзор демонстрирует переход от фиксированных и эвристических стратегий к интеллектуальным и самообучающимся системам, где решение о трассировке зависит от динамического контекста и накопленного опыта. Подобные подходы позволяют точечно реагировать на изменение состояния системы, адаптируя объём собираемой телеметрии без участия человека. Однако внедрение адаптивных методов требует ясного понимания тех ограничений, с которыми сталкиваются традиционные стратегии при попытке обеспечить полную наблюдаемость.

Проблема избыточной телеметрии и накладных расходов

Рост наблюдаемости в микросервисных системах сопровождается увеличением объёма собираемых данных [2]. Каждый сервис генерирует метрики, логи и трейсы, которые в совокупности создают значительную нагрузку на инфраструктуру. Чем выше детализация, тем точнее диагностика, но тем выше накладные расходы.

Высокая детализация метрик приводит к экспоненциальному росту временных рядов, особенно при множестве меток. Это перегружает системы хранения и аналитики, снижая производительность. Методы оптимизации (агрегация, *downsampling*) могут привести к потере критически важной информации.

Каждый запрос порождает множество спанов, передаваемых по сети и записываемых в хранилища. Полная трассировка увеличивает задержку и нагрузку на CPU, особенно при высокой частоте вызовов. Частичное семплирование помогает сократить объём данных, но всегда сопровождается риском упустить важные события.

При больших объёмах телеметрии корреляция логов, метрик и трейсов становится трудоёмкой. Даже при наличии *trace-id* сигнал может теряться в шуме. Это снижает оперативность и эффективность диагностики.

Наращивание ресурсов не решает проблему – темпы роста телеметрии опережают инфраструктурные возможности. Необходимы интеллектуальные механизмы ограничения объёма данных, обеспечивающие баланс между полнотой наблюдения и его стоимостью. Далее представляется архитектура, реализующая этот подход на основе самообучающегося алгоритма.

Архитектура адаптивной трассировки распределённого приложения

Принципиальная схема предлагаемой системы адаптивной трассировки представлена на рисунке 1. Архитектура включает следующие основные компоненты:

1. Агенты трассировки в сервисах. Каждый микросервис приложения оснащается лёгким агентом сбора телеметрии. В роли агента может выступать библиотека трассировки (например, OpenTelemetry SDK), встроенная в код сервиса, либо *sidecar*-контейнер, перехватывающий вызовы. Агенты помечают входящие запросы уникальными идентификаторами трасс и порождают спаны для

важных операций внутри сервиса. Однако, в отличие от традиционных агентов, адаптивный агент умеет работать в двух режимах:

– полная трассировка: детализация включена – агент регистрирует все спаны, логи и метрики по запросу и отправляет их в систему сбора;

– ограниченная трассировка: агент ведёт только минимальный набор метрик, таких как суммарное время обработки и код ответа, без детального логирования внутреннего исполнения.

Переключение между режимами происходит по сигналу от централизованного контроллера. Таким образом, агент выступает исполняющей «ножницей» семплирования: по умолчанию все запросы метятся *trace-id*, но подробные данные собираются только если для данного запроса (трейса) принято решение о трассировке.

2. Диспетчер семплирования (контроллер). Центральный компонент, отвечающий за адаптивный выбор трасс. Диспетчер собирает агрегированную информацию о состоянии системы в реальном времени: глобальные метрики, а также заголовочные сведения о текущих запросах. Используя эти данные, контроллер применяет алгоритм самообучающегося семплирования и решает, какие из поступающих запросов стоит трассировать подробно [3]. Технически это реализуется так: при поступлении нового запроса диспетчер назначает *trace-id* и на основе алгоритма помечает запрос флагом Sampled=true/false. Данный флаг распространяется вместе с запросом через все микросервисы. Агенты в сервисах, получив запрос, читают этот флаг: если он *true*, они включают полный сбор спанов, если *false* – работают в ограниченном режиме. Таким образом, решение о семплировании принимается централизованно на уровне входа запроса в систему и доводится до всех участников распределённой транзакции. Диспетчер семплирования постоянно учится на поступающих данных: он обновляет внутренние модели и правила выбора по мере изменения нагрузки и появления новых типов запросов.

3. Хранилище телеметрии и аналитическая платформа. Все собранные агентами данные поступают в систему хранения и анализа. Она состоит из трёх частей, соответствующих типам телеметрии:

– TSDB для метрик: Временной ряд метрик получает агрегированные метрики со всех сервисов. Отсюда контроллер берёт информацию о текущей нагрузке: RPS, потребление ресурсов, ошибки.

– Логовое хранилище: Централизованный сбор логов получает поток лог-сообщений. При семплировании логов по неотобранным запросам будет меньше, по отобранным – полный *trace log*. Хранилище позволяет поисковым запросом извлекать трассы по *trace-id* для детального анализа инцидентов.

– Хранилище трасс (спанов): Специализированная база для распределённых трейсов. В неё агенты отправляют спаны тех запросов, которые помечены для трассировки. В итоге здесь сохраняются только выборочные трейсы. Каждый трасс включает все спаны с временными метками, что даёт полную картину прохождения конкретного запроса через систему.

Над хранилищами располагается слой аналитики: интерфейсы визуализации (дашборды Grafana, Kibana), система алертов и модуль автоматического анализа. Последний модуль особенно важен – он может включать в себя алгоритмы машинного обучения для обнаружения аномалий в телеметрии [4]. Например, анализировать временные ряды метрик на предмет резких отклонений или применять кластеризацию к трассам для выявления новых паттернов. Результаты анализа могут служить обратной связью для диспетчера семплирования. Таким образом формируется замкнутый цикл адаптации: система учится на исторических данных и регулирует будущее поведение сбора данных.

4. Компонент самообучения. В архитектуру интегрирован блок машинного обучения, который непрерывно совершенствует критерии семплинга [5]. Он может состоять из нескольких моделей:

– модель прогнозирования нагрузки – предсказывает, когда и где в системе может возрасти нагрузка или возникнуть сбой, чтобы упреждающе повысить семплирование на критических узлах;

– модель классификации/регрессии трасс – оценивает поступающие запросы по ряду признаков и даёт предсказание, насколько запрос «интересен» с точки зрения аномалий. Например, может обучаться различать нормальные и аномальные трассы на основе исторических данных;

– модель оптимизации бюджета – решает, как распределить доступный «бюджет» трассировок между различными сервисами или типами запросов, чтобы с максимальной пользой использовать выделенные ресурсы на телеметрию.

Обучение этих моделей происходит на основе накопленных данных, возможно, офлайн либо в виде периодического пересчёта параметров. Затем обновлённые модели используются диспетчером семплирования для принятия решений в режиме реального времени. Благодаря этому система самообучается – со временем всё точнее выделяя, какие запросы нужно трассировать, а какие можно опустить без ущерба для наблюдаемости. Подходы машинного обучения, такие как кластеризация и детектирование выбросов, уже демонстрировали эффективность для обнаружения аномалий в микросервисных средах [6]. В данной архитектуре ML помогает адаптировать правила семплинга динамически.

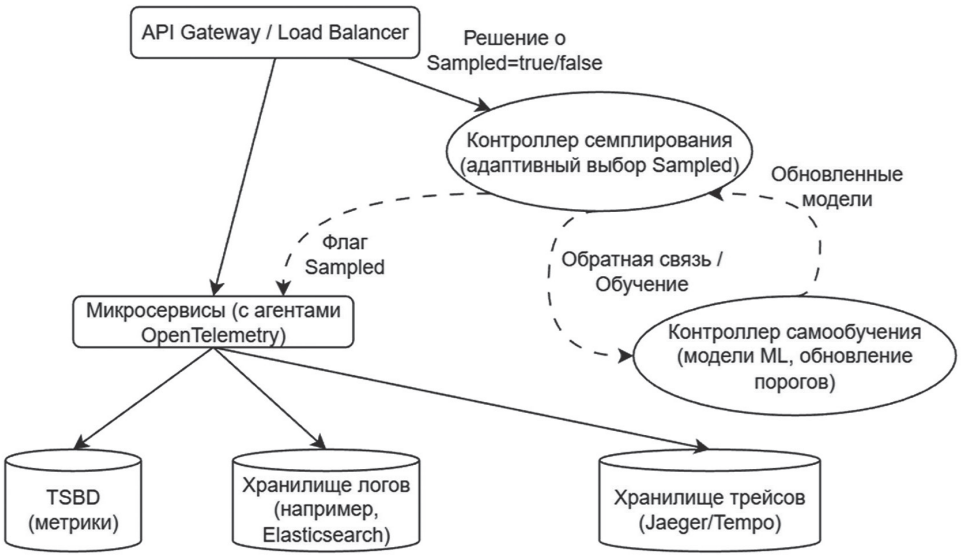

Рисунок 1 – Архитектура системы адаптивной трассировки распределённого приложения

Архитектура спроектирована модульно, что упрощает её расширение [7]. Можно добавлять новые критерии отбора или подключать дополнительные источники данных для улучшения принятия решений. В следующем разделе детально описан алгоритм, по которому диспетчер семплирования на базе этой архитектуры выбирает трассируемые запросы.

Алгоритм самообучающегося выбора трассируемых запросов

Адаптивное семплирование работает в реальном времени, определяя, нужно ли трассировать конкретный запрос. Решение принимается на основе анализа состояния системы и признаков самого запроса.

На входе:

– вектор состояния системы S (нагрузка, ошибки, задержки;

– вектор признаков запроса X (тип операции, пользователь, размер).

Каждый из них обрабатывается соответствующей моделью в выражении (1):

$$Score_{sys} = f_{sys}(S),\ Score_{req} = f_{req}(X). \quad (1)$$

Оценки агрегируются в итоговый балл важности трассировки в выражении (2):

$$Score_{total} = \alpha \cdot Score_{sys} + \beta \cdot Score_{req}. \quad (2)$$

Коэффициенты α и β подбираются эмпирически или автоматически, исходя из приоритетов (например, $\alpha > \beta$ при упоре на стабильность системы).

Далее сравнивается $\text{Score}_{\text{total}}$ с адаптивным порогом T. Если выражение (3):

$$Score_{total} \geq T, то\ Sampled = true. \quad (3)$$

Порог T регулируется динамически, чтобы соблюсти целевой объём телеметрии $P\%$ – при нехватке данных он снижается, при избытке – повышается.

Алгоритм дополняется механизмом обратной связи: если трасса была полезной (содержала сбой или аномалию), модель усиливает её признаки в обучении; если ложной – ослабляет. Модели обновляются периодически на основе накопленных данных. Это обеспечивает адаптацию под текущие паттерны нагрузки и поведения пользователей.

Выбор алгоритмов f_{sys}, f_{req} зависит от требований к скорости: в высоконагруженных системах предпочтительны лёгкие модели, работающие за O(1): решающие деревья, линейные регрессии. Основные вычисления выполняются в централизованном контроллере, упрощая логику агентов в микросервисах.

Экспериментальная часть: моделирование нагрузки

Для оценки эффективности методов была развернута микросервисная система “Online Shop” на базе Kubernetes, аналогичная Socks Shop, включающая 10 сервисов. В качестве стека наблюдаемости использовались OpenTelemetry и Elastic (EFK) [8]. Нагрузку моделировали с помощью wrk2 – инструмента нагрузочного тестирования HTTP-серверов.

Сценарии нагрузки:

– Номинальный режим – 50 запросов/с, система стабильна.

– Пик – всплеск до 500 запросов/с.

– Сбой – 5 % запросов к Order завершаются ошибкой.

– Замедление – у 1 % запросов к Catalog задержка 2 секунды.

Методы трассировки:

1. Полный сбор – трассируются все запросы (базовый уровень).
2. Статическое семплирование – 10 % запросов отбираются случайно.
3. Адаптивное семплирование – алгоритм из раздела 4, с целевым уровнем ~10–15 %.
4. Только ошибки – сохраняются только запросы с ошибкой.

Оцениваемые метрики:

– объём собранных трейсов;

– доля захваченных аномалий;

– накладная задержка на запрос;

– нагрузка на систему мониторинга.

Эксперименты длились по часу на каждый сценарий. Результаты сведены в таблице 2.

Таблица 2 – Сравнение эффективности методов трассировки (тестовый сценарий со сбоем и замедлением)

Метод трассировки	Доля трассируемых запросов	Доля захваченных аномалий	Относительный объём данных
Полный сбор (100 %)	100 %	100 %	100 % (базовый уровень)
Статическое семплирование 10 %	10 %	~50 %	~10 %
Только ошибки	~5 % (ошибки)	~80 % (ошибки и задержки)	~5 %
Адаптивное (предлагаемое)	~15 %	**~90 %**	~15 %

Как видно из данных таблицы, адаптивный подход почти полностью охватывает аномалии (~90 %), сокращая объём данных более чем в 6 раз по сравнению с полным сбором. Он превосходит статическое семплирование и метод «только ошибки» в точности и устойчивости. При этом накладные задержки минимальны – ~1–2 %, сравнимо с обычным семплированием. Метод показал оптимальный баланс между эффективностью трассировки и затратами.

Анализ результатов и сравнение с традиционными методами

Эксперимент подтвердил, что адаптивное семплирование эффективно снижает объём телеметрии без потери качества диагностики. В отличие от полного сбора, который перегружает систему, и

статического семплирования, случайно исключающего важные события, адаптивный метод гибко реагирует на ситуацию в системе. Результаты эксперимента приведены на рисунке 2.

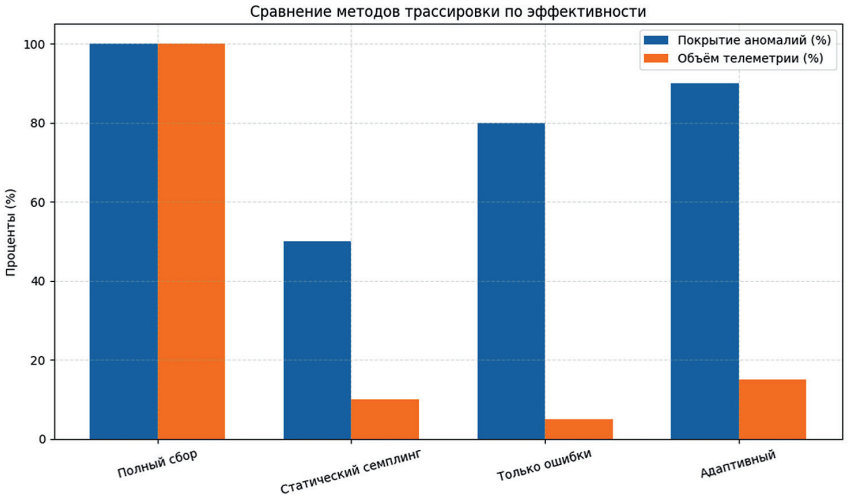

Рисунок 2 – Сравнение методов трассировки по покрытию аномалий и объёму данных

При возникновении ошибок или задержек доля трассировки автоматически увеличивается для соответствующих сервисов, как это произошло с Order (до 50 %) и Catalog. Такой селективный подход позволяет фокусироваться на аномалиях, не расходуя ресурсы на рутинные запросы.

Кроме количества, адаптивный метод выигрывает и по качеству: он сохраняет больше «информативных» трасс, повышая точность диагностики. В отличие от статического семплирования, где *precision* был примерно 0.1, в адаптивной выборке каждая вторая трасса была связана с проблемой, где *precision* больше 0.5.

По сравнению с подходом «только ошибки», адаптивный алгоритм охватывает и скрытые аномалии, выявляя паттерны, недоступные метрикам. Это позволяет обнаружить узкие места в логике сервисов ещё до появления явных сбоев.

При высоких нагрузках адаптивное семплирование временно расширяет охват до 20 %, чтобы убедиться в стабильности. После нормализации объём данных автоматически снижается. Такой механизм служит проактивной страховкой в стрессовых условиях, в отличие от статического подхода, фиксированного по охвату.

Адаптивный метод также снижает накладные расходы:

– уменьшает сетевой и дисковый трафик;

– снижает требования к хранилищам (до 6–7 раз меньше объёма);

– упрощает анализ инцидентов за счёт отбора только релевантных трасс.

Тем не менее качество работы зависит от моделей и порогов. Некорректные настройки могут привести либо к избыточному сбору, либо к пропуску критичных событий. Поэтому система требует валидации и постепенного внедрения, включая «теневой» режим.

Адаптивные методы становятся ключевым направлением развития мониторинга: они позволяют контролировать телеметрию в условиях масштабируемости и нагрузки, сохраняя её ценность для анализа. Представленный подход демонстрирует, как динамическое семплирование обеспечивает эффективный компромисс между полнотой наблюдаемости и её стоимостью.

Заключение

В данной работе предложен адаптивный подход к трассировке распределённых микросервисных систем, направленный на устранение проблемы избыточной телеметрии. Использование фиксированных стратегий семплирования ограничено: они либо порождают чрезмерный объём данных, либо

теряют критически важную информацию о сбоях. Разработанный алгоритм самообучающегося семплирования учитывает как текущее состояние системы, так и характеристики отдельных запросов, что позволяет формировать выборку трасс с максимальной диагностической ценностью при минимальных издержках.

Экспериментальное моделирование показало, что при сокращении объёма трассируемых запросов до 15 % от общего потока удаётся охватить до 90 % аномалий. Адаптивный механизм автоматически увеличивает охват при выявлении отклонений и снижает его в стабильных условиях, что позволяет эффективно распределять ресурсы наблюдаемости. В сравнении с традиционными методами предложенный подход показал более высокую точность и полезность собираемой информации, при этом минимально влияя на производительность системы.

С научной точки зрения новизна работы заключается в интеграции методов машинного обучения в процессы динамического семплирования. Это позволило реализовать механизм обратной связи, при котором система не только реагирует на текущие события, но и постепенно улучшает критерии отбора на основе накопленного опыта. Такой подход формирует основу для построения интеллектуальных, самообучающихся систем мониторинга.

Практическая значимость заключается в том, что архитектура адаптивной трассировки может быть внедрена поверх уже используемых решений, таких как OpenTelemetry и Jaeger. Это особенно важно в условиях масштабируемых и высоконагруженных систем, где нагрузка от телеметрии может составлять значительную долю всех ресурсов. Адаптивный подход позволяет удерживать эту нагрузку под контролем, не снижая качество диагностики и времени реакции на инциденты.

В дальнейшем перспективными направлениями являются: расширение подхода на гибридные данные (логи и трейсы), реализация децентрализованных стратегий принятия решений о трассировке, а также интеграция с системами автоматического масштабирования и отказоустойчивости. Всё это позволяет рассматривать адаптивную наблюдаемость как важный компонент самоуправляемых микросервисных платформ. Таким образом, предложенный подход обеспечивает технологическую основу для балансировки между полнотой мониторинга и его стоимостью – ключевую задачу современных распределённых информационных систем.

Список литературы

1. *Олейник В.А., Картбаев А.Ж.* Интеграция методов машинного обучения в систему мониторинга и анализа микросервисов // Universum: технические науки. – 2024. – № 12 (129). – С. 62–68.
2. *Мясников И.В.* Мониторинг компонентов ISTIO для обеспечения надежности и наблюдаемости SERVICE MESH // Вестник науки. – 2025. – № 5 (86). – С. 734–744.
3. *Huang H.* et al. TraStrainer: Adaptive Sampling for Distributed Traces with System Runtime State // Proc. ACM on Software Engineering. – 2024. – Vol. 1 (FSE). – P. 473–493. – DOI 10.1145/3643748.
4. *Худяков Д.А.* Разработка системы выявления аномалий на основе распределенной трассировки логов // Вестник НГУ. Серия: Информационные технологии. – 2023. – № 1. – С. 62–72.
5. *Baklanov I.* Adaptive Machine Learning Algorithms for Stream Data Processing // Professional Bulletin: Information Technology and Security. – 2024. – № 3. – С. 3–7.
6. *Шевцова Т.А.* Выявление аномалий в сложных данных с помощью кластеризации // Professional Bulletin: Information Technology and Security. – 2024. – № 4. – С. 39–45.
7. *Земсков М.А.* Использование модульных монолитов для разработки масштабируемых web-приложений // Universum: технические науки. – 2024. – № 10 (127). – С. 32–39.
8. *Максимов В.Ю.* Преодоление трудностей наблюдаемости в микросервисной архитектуре // Инновационная наука. – 2024. – № 2-1. – С. 30–35.

References

1. *Olejnik V.A., Kartbaev A.Zh.* Integraciya metodov mashinnogo obucheniya v sistemu monitoringa i analiza mikroservisov // Universum: tekhnicheskie nauki. – 2024. – № 12 (129). – S. 62–68.
2. *Myasnikov I.V.* Monitoring komponentov ISTIO dlya obespecheniya nadezhnosti i nablyudaemosti SERVICE MESH // Vestnik nauki. – 2025. – № 5 (86). – S. 734–744.

3. *Huang H.* et al. TraStrainer: Adaptive Sampling for Distributed Traces with System Runtime State // Proc. ACM on Software Engineering. – 2024. – Vol. 1 (FSE). – P. 473–493. – DOI 10.1145/3643748.
4. *Hudyakov D.A.* Razrabotka sistemy vyyavleniya anomalij na osnove raspredelennoj trassirovki logov // Vestnik NGU. Seriya: Informacionnye tekhnologii. – 2023. – № 1. – S. 62–72.
5. *Baklanov I.* Adaptive Machine Learning Algorithms for Stream Data Processing // Professional Bulletin: Information Technology and Security. – 2024. – № 3. – S. 3–7.
6. *Shevcova T.A.* Vyyavlenie anomalij v slozhnyh dannyh s pomoshch'yu klasterizacii // Professional Bulletin: Information Technology and Security. – 2024. – № 4. – S. 39–45.
7. *Zemskov M.A.* Ispol'zovanie modul'nyh monolitov dlya razrabotki masshtabiruemyh web-prilozhenij // Universum: tekhnicheskie nauki. – 2024. – № 10 (127). – S. 32–39.
8. *Maksimov V.Yu.* Preodolenie trudnostej nablyudaemosti v mikroservisnoj arhitekture // Innovacionnaya nauka. – 2024. – № 2-1. – S. 30–35.

Статья поступила в редакцию: 18.02.2026
Статья принята к публикации: 16.03.2026

Received: 18.02.2026
Accepted: 16.03.2026