Щербань Г.О.

Аналитический отчёт.
Аналитический отчёт по исследованиям в области оркестрации ИИ-нагрузок.

Часть I. Анализ статьи “Enhancing Kubernetes Automated Scheduling with Deep Learning and Reinforcement Techniques for Large-Scale Cloud Computing Optimization” (2024)

Авторы: Zheng Xu, Yulu Gong, Yanlin Zhou, Qiaozhi Bao, Wenpin Qian. ArXiv: 2403.07905.

https://arxiv.org/abs/2403.07905

1. Введение

В 2024 году возникла заметная тенденция к усиленной интеграции методов искусственного интеллекта в механизмы оркестрации контейнеризированных систем, в первую очередь Kubernetes.

Статья Xu et al представляет собой ключевой пример этой тенденции: авторы предлагают гибридный планировщик для Kubernetes, в котором глубокие нейронные сети используются для предсказания состояния кластера, а алгоритмы Reinforcement Learning - для динамической корректировки стратегий распределения задач.

2. Проблематика, на которую отвечает исследование

Kubernetes был спроектирован в эпоху классических микросервисов, где нагрузка была умеренной, а профили ресурсоёмкости — предсказуемыми. Рост AI-нагрузок (LLM-инференс, обработки на GPU, гетерогенные вычислители) создал новые вызовы.

2.1. Ограничения классического планировщика Kubernetes

1. использует статические правила (BinPack, LeastAllocated, BalancedAllocation)
2. плохо реагирует на динамичные нагрузки
3. отсутствует прогнозирование будущих состояний кластера
4. не оптимален для крупномасштабных AI-кластеров, где состояние узлов меняется ежесекундно.

2.2. Проблема “слепоты” к будущей нагрузке

Большие объёмы AI-трафика часто имеют bursty-характер: резко возрастают при батч-обработке, пиковой активности пользователей или во время генерации больших моделей.

2.3. Проблема высокой задержки принятия решений

K8s scheduler работает итерационно и не оптимизирует глобальную картину загрузки.

Исследование Xu et al. предлагает когнитивный планировщик, который “видит вперёд” и активно подстраивает политику.

3. Архитектура предложенного решения

Авторы делят систему на две ключевые части:

3.1. Deep Learning Predictor

Нейросеть прогнозирует:

1. CPU/GPU-загрузку по узлам,
2. динамику потребления памяти,
3. ожидаемое количество входящих задач,
4. состояние очередей.

Используются рекуррентные сети (LSTM), что логично, так как нагрузка — временной процесс.

Функции предсказателя

1. Прогноз состояния кластера через Δt (число шагов вперёд).
2. Выделение “опасных зон”, где ожидается перегруз.
3. Генерация признаков для RL-агента.

3.2. Reinforcement Learning Scheduler

Вторая часть - это RL-агент, который выбирает, как переупаковать или распределить задачи.

Используются методы:

1. Policy Gradient,
2. Deep Q-Learning.

Агент обучается через:

1. состояние кластера;
2. прогнозы нагрузки от LSTM-модели;
3. реальные метрики: CPU idle %, average pod latency, utilization ratio.

Reward-функция учитывает:

1. минимизацию задержек,
2. уменьшение числа пересозданий,
3. балансировку нагрузки,
4. снижение количества конфликтов при размещении подов.

4. Эксперименты и результаты

Исследование демонстрирует улучшение нескольких KPI:

4.1. Эффективность использования ресурсов

RL-планировщик:

1. снижает средний idle CPU на 10–17%;
2. повышает utilization больших GPU-узлов;
3. уменьшает фрагментацию ресурсов.

4.2. Снижение времени выполнения задач

Для AI-нагрузок, использующих очереди задач:

1. ускорение на 8–21%;
2. сокращение хвостовой латентности (p95/p99 задержек).

4.3. Снижение количества re-scheduling событий

По сравнению с Kubernetes Default Scheduler:

до −30% пересозданий подов.

4.4. Масштабируемость

Модель обучалась на кластерах с 1000+ узлами, демонстрируя способность RL-агента адаптироваться вместо деградации стандартных эвристик.

5. Научная значимость и вклад статьи

5.1. Переход от статических эвристик к когнитивной оркестрации

Это новый уровень планировщиков:

предсказательная аналитика,

адаптивное управление,

самокорректирующиеся политики.

5.2. Применимость к реальным AI-нагрузкам

Важно то, что авторы тестировали не только микросервисы, но и:

1. ML-batch задачи,
2. GPU-heavy контейнеры,
3. inference workloads.

5.3. Использование RL как универсального механизма оптимизации

Фреймворк может помочь оптимизировать:

1. энергоэффективность,
2. распределение GPU/TPU,
3. планирование LLM-инференса,
4. даже сетевые маршруты.

6. Тенденции, выявленные на основе статьи

На основе анализа можно выделить несколько ключевых трендов:

Тенденция 1. Переход оркестраторов к self-learning-архитектурам

Kubernetes постепенно движется от монолитного scheduler'а к системам, которые *учатся* на данных кластера.
Это приближает среды оркестрации к:

1. autonomous computing (IBM),
2. self-driving clusters (Google Borg Autopilot),
3. AI-first scheduling.

Тенденция 2. Встраивание DL-прогнозирования в контрольные петли

Оркестрация становится прямым потребителем данных:

1. метрики,
2. временные ряды,
3. логические зависимости задач.

Как следствие - рост популярности:

1. LSTM,
2. Temporal Convolution Networks,

3. графовых нейросетей (для DAG-графов задач).

Тенденция 3. Развитие RL-систем для управления вычислительной инфраструктурой

Kubernetes всё чаще рассматривается как среда, где RL может давать стабильный прирост эффективности:

1. Autoscaling → PPO, DDPG
2. Scheduling → DQN, Policy Gradient
3. Placement → Actor-Critic
4. Migration → Multi-agent RL

Тенденция 4. Комбинированные модели DL + RL становятся стандартом

Планировщики следующего поколения работают по принципу:

Predict ->Optimize ->Act ->Observe-> Retrain

То есть это замкнутые системы управления, аналогичные оптимизации трафика в Google или Яндексе.

7. Ограничения исследования

Статья честно указывает на ряд ограничений:

1. RL-агент сложно обучать на реальном кластере → долгие циклы обратной связи.
2. Сложность объяснимости решений планировщика.
3. Невозможность напрямую использовать модель без адаптации под конкретный workload.
4. Ресурсозатратность обучения (GPU + исторические метрики).
5. Потенциальная нестабильность при быстрых изменениях нагрузки.

8. Выводы по статье

Работа Xu et al. — это важный шаг в сторону **интеллектуальных оркестраторов**, где Kubernetes становится не просто средством запуска контейнеров, а полуавтономной системой принятия решений.

С точки зрения развития индустрии - это фундаментальный шаг к Kubernetes 2.0.

Часть II. Анализ статьи “KIS-S: A GPU-Aware Kubernetes Inference Simulator with RL-Based Auto-Scaling” (2025)

Авторы: Guilin Zhang, Wulan Guo, Ziqi Tan, Qiang Guan, Hailong Jiang. ArXiv: 2507.07932.

https://arxiv.org/abs/2507.07932

1. Введение

Во второй статье анализируется проблема автомасштабирования GPU-инференса в Kubernetes, которая стала ключевой с ростом LLM-сервисов, CV-моделей и других интенсивных AI-нагрузок.

Стандартные механизмы масштабирования Kubernetes (в первую очередь Horizontal Pod Autoscaler, HPA) изначально были ориентированы на CPU/Memory и относительно предсказуемые веб-нагрузки, а не на высоковариативный трафик ИИ-инференса с тяжёлыми GPU-подами.arXiv+1

Авторы предлагают KIS-S - фреймворк, состоящий из:

1. KISim — GPU-aware симулятор Kubernetes-инференса;
2. KIScaler — autoscaler на основе Proximal Policy Optimization (PPO), обученный в симуляции и затем развёртываемый в реальном кластере.arXiv+1

Основная идея: вместо ручной настройки порогов HPA под конкретный workload, обучать RL-политику на симуляторе и переносить её в прод без переобучения, получая умное, “осознанное” масштабирование GPU-подов.

2. Проблематика автомасштабирования GPU-инференса

2.1. Ограничения стандартного HPA

HPA в Kubernetes:

1. реагирует реактивно — только после роста метрик;
2. опирается на простые агрегаты (CPU %, memory %, иногда кастомные метрики);
3. не учитывает внутренние GPU-метрики (SM-utilization, memory bandwidth, GPU memory, concurrency);
4. плохо справляется с бурстовым трафиком и SLA-параметрами (p95/p99 latency).arXiv+1

Для production-инференса LLM это приводит к:

1. либо перемасштабированию (слишком много подов, перерасход GPU);
2. либо недосозданию подов (SLA-нарушения, рост очередей, timeouts).

2.2. Проблема отсутствия безопасной среды для экспериментов

Тюнинг autoscaler'ов на живом кластере с дорогостоящими GPU:

1. риск нарушить SLA,
2. дорого по деньгам,
3. сложно воспроизводить эксперименты.

Отсюда логичный шаг — сначала научить политику в реалистичном симуляторе Kubernetes-инференса, а уже потом использовать её в проде.

3. Архитектура KIS-S

3.1. KISim — симулятор Kubernetes-инференса

KISim — это GPU-aware симулятор, который:

1. моделирует Kubernetes-кластер с GPU-нодами;
2. эмулирует запуск подов с AI-моделями;
3. воспроизводит реальные трафик-паттерны;
4. использует реальные метрики GPU через интеграцию с Prometheus.arXiv+1

Особенности:

1. Traffic-controllable: можно задавать профили нагрузки (burst, diurnal, random spikes), что важно для LLM API.
2. Real hardware in the loop: симулятор опирается на реальные измерения производительности GPU, а не на абстрактные модели.
3. Prometheus-интеграция: собираются метрики, аналогичные тем, что будут доступны в проде — чтобы RL-агент "видел" те же признаки.

3.2. KIScaler — RL-autoscaler на базе PPO

KIScaler — это RL-агент, который заменяет/дополняет HPA.arXiv+1

Алгоритм: Proximal Policy Optimization (PPO) — устойчивый к шумным оценкам градиента, популярный в задачах управления.

Состояние (state):

текущие метрики latency (p50/p95),

длины очередей,

GPU-utilization,

количество активных подов по моделям,

профиль трафика.

Действия (actions):

масштабирование реплик вверх/вниз для групп подов,

возможно, дифференцированное масштабирование по классам моделей.

Reward:

штрафы за SLA-нарушения (latency выше таргета),

штрафы за перерасход GPU,

бонусы за стабильность и минимизацию колебаний.

Ключевая фишка: политика обучается полностью в симуляции (на KISim), а затем переносится в реальный Kubernetes-кластер без переобучения (sim-to-real transfer).

4. Экспериментальные результаты

Авторы сравнивают KIScaler с:

1. стандартным HPA (CPU-based, иногда с кастомными метриками),
2. несколькими простыми эвристиками масштабирования.

Результаты:arXiv+1

1. **P95 latency**

   KIS-S позволяет значительно снизить p95-латентность инференса — для некоторых workloads до **6.7× улучшения по сравнению с CPU-бейслайном**, где inference шёл без GPU-ускорения.

   По сравнению с HPA — стабильное снижение хвостовой латентности (p95, p99), особенно при бурстовом трафике.

2. **Эффективность GPU-ресурсов**

   GPU-utilization более высокая и ровная, без длительных периодов недозагрузки.

   Меньше “раскачки” — нет постоянных колебаний числа подов, как при грубых порогах.

3. **Стабильность масштабирования**

   RL-политика ведёт себя более “плавно”: избегает частых масштабирований туда-сюда (churn).

   Это важно и с точки зрения стабильности, и с точки зрения стоимости (меньше overhead на контейнеризацию и warmup моделей).

4. **Sim-to-real перенос**

   Показано, что политика, обученная в KISim, может быть применена к реальному кластеру без переобучения, с сохранением преимуществ по SLA и ресурсам.

5. Тенденции, отражённые в статье

Тенденция 1. Появление специализированных симуляторов для AI-нагрузок в Kubernetes

KISim — пример того, что **инфраструктурой начинают управлять через симуляцию**:

1. перед развёртыванием новых политик (autoscaling, scheduling) их тестируют в close-to-real симуляторе;
2. симулятор становится частью CI/CD для инфраструктурных решений.

Это прямой шаг к “цифровым двойникам” Kubernetes-кластеров.

Тенденция 2. RL как стандартный инструмент для autoscaling

Если в первой статье RL использовался для планирования задач (scheduling), то здесь RL прямо применяется к **автомасштабированию GPU-инференса**:

1. RL выбирает не просто “сколько реплик”, а **как именно масштабировать в зависимости от динамики трафика**;
2. policy-based autoscaling постепенно вытесняет threshold-based подход.

Тенденция 3. Сближение autoscaling с “serverless AI”-моделью

Хотя статья формально не позиционирует KIS-S как serverless-платформу, на идейном уровне:

1. пользователю не нужно думать о количестве подов;

2. autoscaler, обученный на RL, сам решает, как и когда поднимать/опускать реплики;
3. это приближает Kubernetes с GPU-инференсом к **serverless AI-парадигме**, где разработчик платит “за вызовы модели”, а не за инфраструктуру.

6. Ограничения и вызовы

Авторы отмечают ряд ограничений фреймворка:arXiv

1. Необходимость создания и поддержки сложного симулятора (KISim).
2. Возможные **расхождения между симуляцией и реальным кластером** (drift) при изменении аппаратной платформы или трафика.
3. Трудность интерпретации решений RL-политики.
4. Потенциальные риски при ошибке в симуляторе — RL-политика научится “оптимизировать” неправильную модель мира.

7. Выводы по статье

KIS-S демонстрирует, как можно перейти от классического HPA к **интеллектуальному autoscaling’у**, который:

1. понимает GPU-специфику,
2. использует RL для балансировки SLA и стоимости,
3. обучается в симуляции, а не “на живых пользователях”.

Эта работа логически продолжает направление, начатое в статье Xu et al. по интеллектуальному планированию, и показывает, что AI проник не только в планировщик задач, но и в слой масштабирования.

Часть III. Анализ статьи “Intelligent Orchestration of Distributed Large Foundation Model Inference at the Edge” (2025)

Авторы: Fernando Koch, Aladin Djuhera, Alecio Binotto. ArXiv: 2504.03668.

https://arxiv.org/abs/2504.03668

1. Введение

Третья статья смещает фокус с “классического” облачного Kubernetes-кластера на edge-среду и Multi-Access Edge Computing (MEC).
Задача — обеспечить инференс больших foundation-моделей (LLM, мультимодальные модели) на периферии, где ресурсы ограничены и гетерогенны: слабые GPU, CPU-only узлы, нестабильная сеть.arXiv+1

Авторы описывают интеллектуальную систему оркестрации Distributed Split Inference (DSI), которая:

1. разбивает большую модель на сегменты (слои/блоки),
2. распределяет их по нескольким edge-узлам,
3. мигрирует части модели при изменении нагрузки или сетевых условий,
4. управляет этим в реальном времени с учётом QoS (latency, throughput, privacy).arXiv+1

Система интегрируется с существующими оркестраторами (Kubernetes, Ray), расширяя их возможности на edge.

2. Проблема: LLM-инференс на периферии

2.1. Ограниченная вычислительная мощность

Большие foundation-модели (LLaMA, Qwen и др.) требуют:

1. десятки гигабайт памяти,
2. мощные GPU,
3. высокую пропускную способность.arXiv

Edge-узлы же:

1. могут быть вообще без GPU,
2. имеют ограниченный RAM,
3. подключены по относительно нестабильным каналам.

2.2. Требования по задержке и конфиденциальности

В MEC-сценариях:

1. нельзя гнать всё в центральное облако из-за задержки;
2. есть требования по локальной обработке данных (privacy, GDPR, отраслевые регуляции).

Стандартный Kubernetes-инференс “как в облаке” здесь не работает — нужен распределённый подход, учитывающий “раздробленность” инфраструктуры.

3. Концепция Distributed Split Inference (DSI)

DSI — это стратегия, при которой модель физически разделяется на части (например, по слоям трансформера), и разные сегменты выполняются на разных узлах:arXiv+1

1. Начальные слои — могут работать ближе к источнику данных (на edge-узле).
2. Средние/тяжёлые слои — на более мощных нодах (например, в локальном “микро-датацентре”).
3. Финальные слои — снова ближе к пользователю, если нужно локальное post-processing или фильтрация.

Система оркестрации:

1. Профилирует ресурсы узлов (CPU, GPU, память, сеть).
2. Определяет схему разбиения модели.
3. Решает, где какой сегмент должен жить.
4. При изменении условий (нагрузка, отказ узла, изменение сети) — мигрирует сегменты модели.

4. Интеллектуальная оркестрация: архитектура решения

Система Koch et al. включает:arXiv+2arXiv+2

1. **Мониторинг и профилирование**

   Собираются метрики узлов (CPU/GPU, память),

   сетевые параметры (RTT, пропускная способность),

   характеристики нагрузки (число запросов, типы запросов, SLA).

2. **Планировщик разбиения и размещения модели**

   Рассматривает модель как граф (слои, блоки, подзадачи),

   решает, где исполнять каждый сегмент,

   учитывает trade-off:

Latency,

Throughput,

Энергопотребление,

Privacy (где можно/нельзя держать данные/веса).

3. **Модуль миграции и адаптации**

   При перегрузке узла — переносит часть модели на другой узел,

   При деградации сети — меняет конфигурацию DSI (например, сдвигает больше слоев ближе к пользователю или наоборот),

   Стремится делать это прозрачно для клиента (без заметного разрыва сервиса).

4. **Интеграция с Kubernetes / Ray**

   Для контейнеризации и деплоя используются стандартные механизмы Kubernetes,

   Ray и подобные системы помогают организовать распределённое выполнение задач и управление актёрами.

5. Эксперименты и результаты

Экспериментальные сценарии включают различные MEC-конфигурации, где:arXiv+1

часть узлов GPU-capable, часть — только CPU;

сеть может быть:

1. высокоскоростной,
2. низкоскоростной,
3. с переменной задержкой.

Результаты показывают, что Intelligent DSI-оркестратор:

1. Снижает задержку инференса по сравнению с:

   чисто облачным inference (когда вся модель в центральном датацентре),

   наивным размещением модели на одном edge-узле.

2. Повышает пропускную способность (больше параллельных запросов при тех же ресурсах).
3. Поддерживает QoS при отказах и изменениях сети, динамически мигрируя части модели.
4. Обеспечивает лучший баланс между privacy и производительностью, чем подход “всё в облаке” или “всё на edge”.

6. Тенденции, отражённые в статье

Тенденция 1. Выход оркестрации AI за пределы датацентров в сторону edge/MEC

Kubernetes, Ray и подобные фреймворки всё чаще работают не только в облаке, но и в пограничных вычислениях:

1. фабрики, умные города, телеком-станции, транспорт;
2. здесь важно не только “где дешевле GPU”, но и где быстрее и безопаснее обрабатывать данные.

Тенденция 2. Модели рассматриваются как распределённые объекты

Раньше модель = “чёрный ящик внутри одного контейнера”.
Теперь модель:

1. разбивается на части,
2. эти части становятся opchestratable units — объектами управления, как поды и сервисы.

Это новый уровень - оркестрация не только контейнеров, но и самих нейросетей.

Тенденция 3. Оркестраторы становятся multi-objective системами

Решения принимаются не по одной оси (“минимизировать latency” или “минимизировать cost”), а по нескольким:

1. латентность,
2. пропускная способность,
3. энергопотребление,
4. приватность данных,
5. устойчивость к сбоям.

7. Ограничения и исследования будущего

Отмеченные ограничения:arXiv+1

1. Сложность реализации DSI на реальных больших LFM (десятки миллиардов параметров).
2. Потребность в стандартизированных интерфейсах для разбиения моделей и их миграции между узлами.
3. Обратная связь на уровне моделей (нужно учитывать, как разбиение влияет на качество и стабильность).

4. Интеграция с существующими MLOps-пайплайнами и системами управления версиями моделей.

8. Выводы по статье

Статья Koch et al. показывает, как современная оркестрация AI выходит на уровень edge и начинает оперировать не только контейнерами, но и частями нейросетей. Это логично продолжает тренды двух предыдущих работ, но в новой плоскости — дистрибутивного инференса foundation-моделей в гетерогенной среде.

Часть IV. Сводный анализ тенденций и перспективные технологии

1. ИИ-оптимизация распределения ресурсов

Во всех трёх статьях прослеживается общий вектор: распределение ресурсов перестаёт быть ручной задачей и передаётся ИИ-моделям.

В статье Xu et al. DL+RL-планировщик прогнозирует состояние кластера и динамически подбирает стратегию расписания задач, минимизируя idle и ускоряя выполнение.arXiv+1

В KIS-S RL-агент (PPO) оптимизирует GPU-автомасштабирование, учитывая метрики latency и загрузки GPU.arXiv+1

В работе Koch et al. оркестратор DSI распределяет части модели по узлам, оптимизируя latency, throughput и privacy в MEC-среде.arXiv+1

Общий тренд:

Распределение ресурсов (CPU, GPU, память, сеть, сами модели) становится задачей для ML/RL, а не для статических правил.

Перспективные технологии в этом направлении:

1. Гибридные DL+RL-планировщики для Kubernetes и других оркестраторов.
2. GPU-aware/TPU-aware autoscalers, использующие реальную телеметрию (Prometheus, DCGM).
3. Adaptive model placement — динамическое размещение моделей и их сегментов в гибридных (cloud+edge) средах.

2. Бессерверный ИИ в Kubernetes

Формально ни одна статья прямо не использует термин “serverless”, но по сути все они двигают Kubernetes в сторону “серверлесс-подобного” опыта для AI:

У Xu et al. разработчики в идеале не думают о том, на каких нодах и как будут выполняться их задачи — за это отвечает интеллектуальный планировщик.

В KIS-S политика RL сама решает, сколько нужно подов с ИИ-моделями и когда их масштабировать, исходя из SLA и нагрузки — это очень похоже на serverless-идею “заплати только за реально использованные ресурсы и не думай о серверах”.arXiv+1

У Koch et al. пользовательский сервис просто вызывает foundation-модель, а оркестратор сам решает, на каком наборе edge/облачных узлов и как разнести её слои — опять же, типичное “serverless ощущение”.

Можно сказать, что формируется подтип:

Serverless AI on Kubernetes - когда разработчик работает на уровне “модель/инференс/quality”, а не на уровне “nodes/pods/replicas”.

Перспективные технологии:

Serverless фреймворки для AI поверх K8s (Knative + GPU, KServe, BentoML в режиме autoscaling).

Интеграция function-as-a-service моделей (LLM, CV) с GPU-aware autoscaling (идеи KIS-S).

Появление “AI PaaS” над Kubernetes, где все три уровня из статей (scheduling, scaling, split inference) спрятаны под одной абстракцией.

3. Использование RL для планирования задач

Во всех трёх работах Reinforcement Learning не просто игрушка, а реальный рабочий инструмент:

Xu et al.: RL принимает решения о размещении задач при заданном предсказанном состоянии системы.

KIS-S: RL (PPO) управляет autoscaling’ом, балансируя SLA и cost.

Koch et al. напрямую RL не детализируют на уровне алгоритма в тексте, но их оркестратор вполне может быть реализован как multi-objective RL или multi-agent RL система, что предлагается как направление будущих исследований.arXiv+1

Тренд:

RL становится “двигателем” для всех контуров управления оркестрацией: от планировщика до автомасштабирования и распределённого инференса.

Перспективы:

Multi-agent RL для координации множества оркестраторов (cloud + edge).

Hierarchical RL — верхний уровень решает, где и как исполнять большие модели, нижний — как масштабировать конкретные поды, ещё ниже — как маршрутизировать запросы.

Safe RL — важное направление, так как ошибки в политике могут приводить к SLA-нарушениям и финансовым потерям.

4. Общие выводы по трендам оркестрации и контейнерам

1. Kubernetes эволюционирует от “оркестратора контейнеров” к “интеллектуальной системой управления вычислениями для AI”.

2. AI/ML и RL входят в контрольный контур инфраструктуры:

планирование задач

масштабирование

распределённый инференс на edge.

3. Serverless-парадигма проникает в AI на K8s — разработчики всё меньше думают о pod’ах и нодах, всё больше — о моделях и SLA.
4. Foundation-модели становятся распределёнными объектами оркестрации — их слои и сегменты рассматриваются как ресурс, который можно переносить и адаптировать.
5. Симуляторы и цифровые двойники кластеров начинают играть ключевую роль:

обучение RL-политик,

тестирование новых стратегий без риска.