

И. В. Чугунков¹, М. М. Ровнягин¹, К. А. Федоров^{1✉}, К. В. Сысоев¹

Архитектура адаптивного планировщика в геораспределенном кластере

¹Национальный исследовательский ядерный университет «МИФИ»,
г. Москва, Российская Федерация
e-mail: klim.fedorov2001@yandex.ru

Аннотация. В работе рассматривается задача планирования размещения контейнеризированных приложений в геораспределенном кластере Kubernetes. Показано, что стандартный планировщик kube-scheduler, ориентированный на заявленные ресурсы и текущее состояние узлов, не учитывает в достаточной мере сетевую топологию, задержки межсервисного взаимодействия и временную динамику нагрузки, что в условиях геораспределения приводит к снижению эффективности использования ресурсов. Геораспределенный кластер формализован в виде четверки из множества узлов, множества Pod'ов, функции сетевого расстояния и матрицы интенсивностей взаимодействия Pod'ов; состояние кластера разделено на инфраструктурный и прикладной слои, различающиеся характерным временем изменения. На этой основе предложена архитектура адаптивного планировщика, в которой решение формируется несколькими специализированными контурами – первичного планирования, периодической адаптации и упреждающего управления, – а итоговое размещение принимает отдельный компонент-арбитр. Исполнение решения выполняют компоненты scheduler и descheduler через Kubernetes API; единым хранилищем состояния выступает PostgreSQL, источником показателей – VictoriaMetrics. Практическая значимость заключается в расширяемой архитектурной основе для построения планировщика, учитывающего специфику геораспределенных сред.

Ключевые слова: планирование, кластер, архитектура

I. V. Chugunkov¹, M. M. Rovniagin¹, K. A. Fedorov^{1✉}, K. V. Sysoev¹

Adaptive scheduler architecture in a geo-distributed cluster

¹National Research Nuclear University "MEPhI" Moscow, Russian Federation
e-mail: klim.fedorov2001@yandex.ru

Abstract. This paper addresses the problem of scheduling the placement of containerized applications in a geo-distributed Kubernetes cluster. It is shown that the standard kube-scheduler, which relies on declared resource requests and the current state of nodes, does not sufficiently account for network topology, inter-service communication latency, or the temporal dynamics of workloads, which leads to reduced resource utilization efficiency under geo-distribution usage scenario. The geo-distributed cluster is formalized as a quadruple consisting of a set of nodes, a set of Pods, a network distance function, and a matrix of pairwise Pod interaction intensities; the cluster state is decomposed into an infrastructure layer and an application layer, which differ in their characteristic time scales of change. Based on this formalization, an architecture of an adaptive scheduler is proposed, in which the decision is formed by several specialized loops – primary scheduling, periodic adaptation, and proactive control – while the final placement is produced by a separate arbiter component. The decision is executed by the scheduler and descheduler components through the Kubernetes API; PostgreSQL serves as the unified state store, and VictoriaMetrics as the source of metrics. The practical significance of the work lies in providing an extensible architectural foundation for building a scheduler that takes into account the specifics of geo-distributed environments.

Keywords: scheduling, cluster, architecture

Введение

В последние годы Kubernetes стал стандартной платформой оркестрации контейнеризированных приложений, однако по мере роста масштабов кластеров и усложнения микросервисных взаимодействий задача эффективного размещения Pod'ов становится существенно сложнее. В геораспределенном кластере качество планирования определяется не только доступностью CPU и RAM, но и сетевой топологией, задержками между узлами и интенсивностью обмена данными между сервисами. В то же время стандартные механизмы планирования Kubernetes ограничены в данных текущим состоянием узлов и статически заданных requests.

Вследствие этого возникает необходимость в адаптивном подходе к планированию, сочетающем реакцию на текущие события, периодическую корректировку размещения и упреждающее управление на основе прогноза нагрузки. Актуальность работы определяется потребностью в адаптивном планировщике и необходимостью проектирования архитектуры, способной учитывать специфику геораспределенной среды для кластеров различных размеров, и принимать решения на основе нескольких классов показателей. Целью работы является разработка архитектуры адаптивного планировщика для Kubernetes, обеспечивающего более эффективное использование ресурсов кластера и снижение топологической стоимости размещения.

Постановка задачи

Целью настоящей работы является разработка архитектуры адаптивного планировщика для оркестратора Kubernetes с учетом геораспределенной топологии сети. В случае плотного расположения узлов кластера (размещения внутри одного датацентра или одной серверной стойки) топологию взаимодействия Pod'ов можно не учитывать, поскольку задержками внутри сети кластера можно пренебречь [1–6]. Геораспределение узлов требует учета степени взаимодействия между Pod'ами, а также пропускную способность сети узлов, через которую будет осуществляться это взаимодействие [7, 8].

В таком случае геораспределенный кластер рассматривается как четверка:

$$C = (N, P, E, \Psi), \quad (1)$$

где $N = \{n_1, \dots, n_{|N|}\}$ – множество узлов, $P = \{p_1, \dots, p_{|P|}\}$ – множество Pod'ов, подлежащих размещению, $E: N \times N \rightarrow R_{\geq 0}$ – сетевая топология узлов: неотрицательная симметричная функция расстояния между узлами – латентность в мс, число шагов по топологическому дереву и т.п. $\Psi \in [0, 1]^{|P| \times |P|}$ – топология Pod'ов: нормированная матрица попарных интенсивностей взаимодействия. В общем случае не симметрична. Координата ψ_{ij} показывает, насколько сильно Pod p_i играет от близости к p_j , $\psi_{ii} = 0$.

В рамках одного цикла планирования удобно разделить четверку C на два слоя, различающиеся характерным масштабом времени изменений. Инфраструктурный слой C_{infra} :

$$C_{infra}(t) = (N(t), E(t)), \quad (2)$$

описывающий состав узлов кластера и сетевую топологию. Он меняется редко: при добавлении или выводе узлов: при реконфигурации сети. Для него характерно время порядка часов и суток. Прикладной слой C_{wl} :

$$C_{wl}(t) = (P(t), \Psi(t)), \quad (3)$$

описывает состав Pod'ов, подлежащих размещению, и интенсивность их попарного взаимодействия. Он изменяется динамично: при порождении и удалении Pod'ов; при изменении профиля обращений сервисов друг к другу. Он характеризуется временем порядка секунд и минут. Полное состояние кластера в момент t выражается как $C(t) = C_{infra}(t) \cup C_{wl}(t)$.

Формула (1) описывает срез кластера в момент принятия решения t_0 . Уже исходя из изменчивости кластера с течением времени, становится ясно, что планировщик в геораспределенном кластере должен опираться в решении не столько на статические ограничения, сколько быть адаптивным к изменениям как на инфраструктурном слое, так и на прикладном.

Каждому узлу $j \in \{1, \dots, |P|\}$ сопоставляется вектор емкости ресурсов:

$$r(n_j) = (r_j^{CPU}, r_j^{RAM}, r_j^{disk}, r_j^{disk-bw}, r_j^{net}, r_j^{GPU}, \dots) \in R_{\geq 0}^K, \quad (4)$$

где n_j – узел кластера, $r_j^{(k)}$ – значение емкости типа ресурса k для узла j , K – число типов ресурсов, учитываемых планировщиком. Различные контуры могут не учитывать все типы ресурсов, однако их набор должен единственным для всего планировщика. Каждому Pod'у i в момент его размещения сопоставляется вектор заявленных или, в случае повторного размещения, спрогнозированных потребностей:

$$d(p_i) = (d^{res}(p_i), \psi_i) \in R_{\geq 0}^K, \quad (5)$$

где $d^{res}(p_i) = (d^{(1)}(p_i), \dots, d^{(K)}(p_i)) \in R_{\geq 0}^K$ – ресурсный блок: вектор потребности Pod'а в ресурсах, $d^{(K)}(p_i)$ – значение потребности Pod'а p_i в ресурсе типа k . $\psi_i = (\psi_{i,1}, \dots, \psi_{i,|P|})$ – i -я строка матрицы Ψ : вектор интенсивностей взаимодействия Pod'а p_i с остальными Pod'ами – топологический блок потребностей Pod'а.

Размещение Pod'ов задается отображением:

$$\pi: P \rightarrow N, \quad (6)$$

Одновременно удовлетворяющим по координатным ресурсным ограничениям:

$$u_j^{(k)}(\pi) = \frac{1}{r_j^{(k)}} \sum_{p: \pi(p)=n_j} d^{(k)}(p) \leq 1, \forall k \in \{1, \dots, K\}, \forall j \in \{1, \dots, |N|\}, \quad (7)$$

и минимизирующим топологическую стоимость:

$$T_{topo}(\pi) = \sum_{i=1}^{|P|} \sum_{j=1}^{|P|} \psi_{ji} E(\pi(p_i), \pi(p_j)), \quad (8)$$

где $u_j^{(k)}$ – утилизация ресурса типа k на узле n_j после применения размещения π , лежащая в диапазоне $[0, 1]$, где 0 – ресурс не востребован, 1 – используется полностью. $\sum_{p: \pi(p)=n_j} d^{(k)}(p)$ – суммарная потребность в ресурсе типа k всех Pod'ов размещенных на узле n_j .

Таким образом, допустимым размещением может называться то размещение, при котором $u_j^{(k)} \leq 1$, а недопустимым – $u_j^{(k)} > 1$.

Архитектура системы

Исходя из многофакторности задачи планирования размещения в геораспределенном кластере, в архитектуре планировщика предлагается предусмотреть несколько контуров планирования. Таким образом, каждый из них сможет решать узкую задачу, что позитивно скажется на их эффективности по отдельности. В то же время возможность принятия конечного решения по размещению будет возложена на отдельный сервис, который будет иметь доступ к решениям всех контуров и их прошлым решениям.

В качестве плюсов данного подхода можно отметить расширяемость системы планировщика – при необходимости учета новых показателей в размещении можно будет добавить новый контур, который будет учитывать их. Это могут быть, например, показатели о стоимости аренды узлов кластера, о связанном с кластером углеродном следе [9, 10]. Более того, необязательно ограничиваться показателями самого кластера – при необходимости контур может учитывать сколь угодно специфичные для сервисов кластера показатели, включая бизнес, QoS-метрики [11–13]. А также метрики, связанные с анализом киберинцидентов [14–17]. Так, планировщик будет способен закрыть потребности практически любой команды разработки.

На данный момент в планировщике планируется реализовать три контура: контур первичного планирования – реакция на событие создания пода/деплой-мента: решение принимается быстро, опираясь на текущие показатели утилизации узлов и политики scoring/filtering; контур периодической адаптации – решения принимаются для достижения сбалансированности в утилизации узлов и минимизации топологической стоимости $T_{topo}(\pi)$ с некоторой периодичностью, для сбора необходимых для анализа данных о Pod'ах и реальном состоянии узлов, и контур упреждающего управления – решения опираются на прогноз или

оценку будущего спроса/нагрузки для превентивного релевантного воздействия, что особенно релевантно при сезонности запросов.

Исходя из этого, можно представить планировщик в виде следующих компонентов (рис. 1).

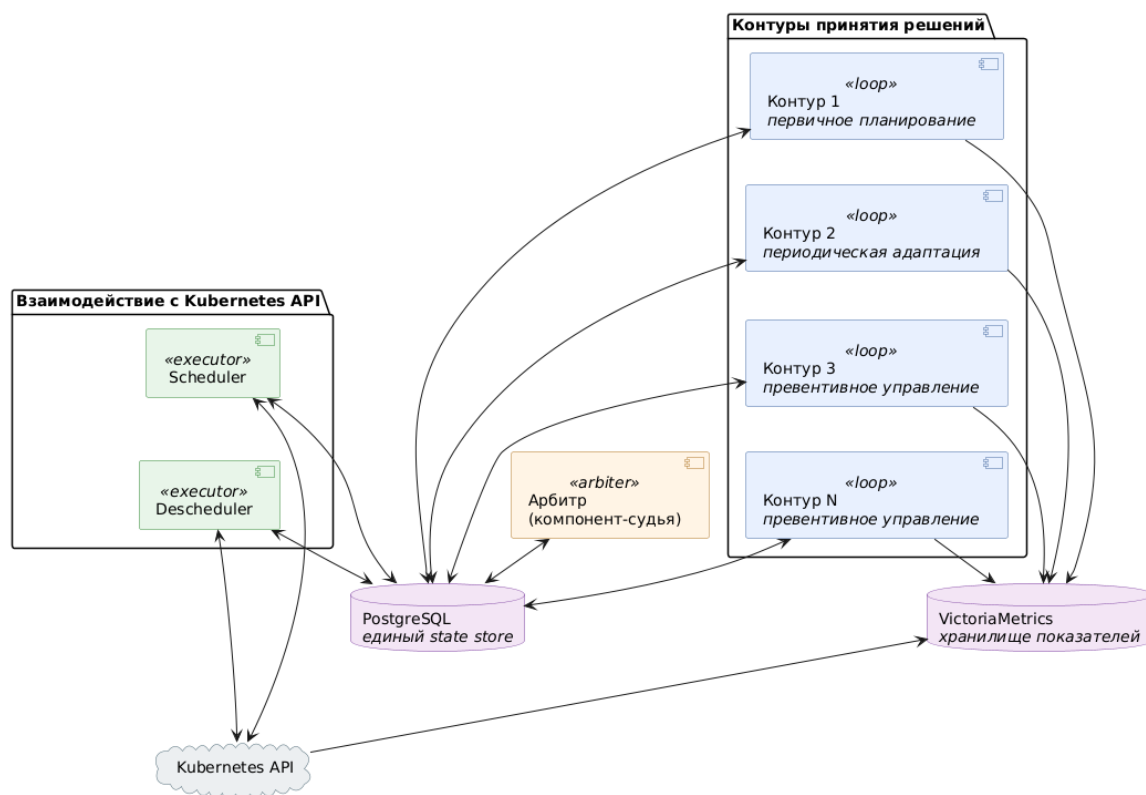


Рис. 1. Диаграмма компонентов планировщика

Компоненты, входящие в состав адаптивного планировщика, делятся на две группы: компоненты, благодаря которым формируется решение о размещении; и компоненты, реализующие размещение из решения, взаимодействуя с Kubernetes API. Отдельным компонентом выступает arbiter, принимающий итоговое решение. В качестве источника показателей для решений контуров выступает база данных временных рядов VictoriaMetrics. В нее записывают свои показатели сборщики, работающие внутри кластера Kubernetes.

Решения каждого контура в отдельности и итоговое размещение хранятся в реляционной базе данных PostgreSQL. Итоговое решение используют компоненты scheduler и descheduler. Они относятся ко второй группе. Scheduler привязывает Pod'ы к узлам, а descheduler отвязывает уже размещенные Pod'ы от несоответствующих новому решению компонента arbiter, инициализируя процесс их размещения внутри кластера заново. Необходимость компонента descheduler обусловлена тем, что Kubernetes не перемещает Pod'ы, которые уже перешли из состояния PENDING в состояние RUNNING.

Таким образом, один цикл планирования в диаграмме будет выглядеть следующим образом (рис. 2).

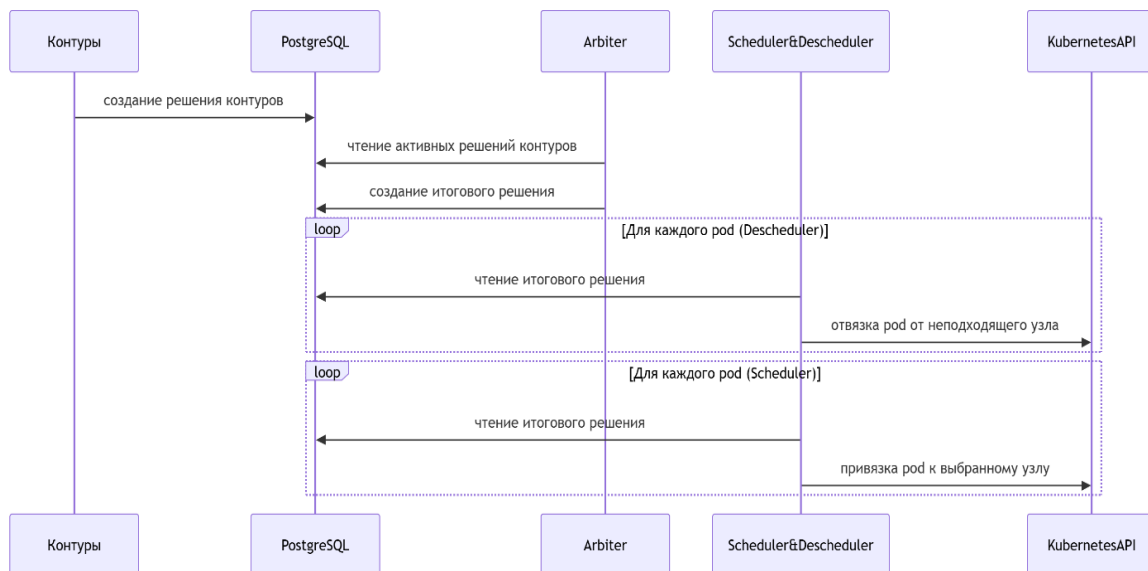


Рис. 2. Диаграмма последовательности цикла планирования

Сначала каждый контур будет принимать собственное решение о размещении и размещать его в БД. Затем arbiter, проанализировав его, создаст итоговое решение. Далее в действие вступает descheduler – вытесняя Pod’ы с неподходящих узлов. После того как таких Pod’ов не осталось scheduler назначает узел для неразмещенных Pod’ов.

Заключение

В работе предложена архитектура адаптивного планировщика для геораспределенного кластера Kubernetes, учитывающая как ресурсные ограничения узлов, так и топологию сетевого взаимодействия между Pod’ами. Показано, что в условиях геораспределения задача размещения не может эффективно решаться только на основе локальных метрик утилизации или статичных ограничений, поскольку существенное влияние на качество размещения оказывают сетевые задержки и интенсивность взаимодействия между сервисами, изменяющиеся с течением времени. Для учета этой специфики состояние кластера представлено в виде инфраструктурного и прикладного слоев, а процесс принятия решения организован как взаимодействие нескольких специализированных контуров планирования и отдельного компонента arbiter, формирующего итоговое решение.

Предложенная архитектура обеспечивает расширяемость и адаптивность системы: при появлении новых требований или метрик QoS можно добавлять новые контуры без переработки всей системы планирования. Разделение компонентов на формирующие решение и исполняющие его через Kubernetes API позволяет упростить развитие системы и повысить прозрачность процесса принятия решений. Практическая значимость работы заключается в формировании архитектурной основы для построения адаптивного планировщика, способного учитывать специфику геораспределенных вычислительных сред и динамику нагрузки.

Дальнейшие исследования целесообразно направить на разработку формальных правил арбитража между контурами, экспериментальную оценку качества размещения на реальных и синтетических нагрузках, а также на исследование устойчивости предложенного подхода при изменении топологии сети и профиля межсервисного взаимодействия.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Khaldoun S. et al. A survey of Kubernetes scheduling algorithms [J] //Journal of Cloud Computing. – 2023. – Т. 12. – №. 1.
2. Jakobović K. PROPOSAL FOR DEVELOPMENT OF A CUSTOM KUBERNETES OPERATOR: дис. – Algebra University, 2023.
3. Creane B., Gupta A. Kubernetes Security and Observability. – "O'Reilly Media, Inc.", 2021.
4. Carrión C. Kubernetes scheduling: Taxonomy, ongoing issues and challenges //ACM Computing Surveys. – 2022. – Т. 55. – №. 7. – С. 1-37.
5. Dakić V. et al. Optimizing Kubernetes Scheduling for Web Applications Using Machine Learning //Electronics. – 2025. – Т. 14. – №. 5. – С. 863.
6. Jian Z. et al. DRS: A deep reinforcement learning enhanced Kubernetes scheduler for microservice-based system //Software: Practice and Experience. – 2024. – Т. 54. – №. 10. – С. 2102-2126.
7. Ahmed A. M., Kashmola M. Y. Advances in Distributed Scheduling Algorithms: A Three-Layer Architecture Integrating Deep Reinforcement Learning and Energy Optimization (2023-2024) //Journal of Al-Qadisiyah for Computer Science and Mathematics. – 2025. – Т. 17. – №. 1. – С. 62–71-62–71.
8. Furnadzhiev R., Shopov M., Kakanakov N. Efficient Orchestration of Distributed Workloads in Multi-Region Kubernetes Cluster //Computers. – 2025. – Т. 14. – №. 4. – С. 114.
9. Lakshan S., Hussain S. A Review of AI-Driven Techniques for Cost Optimization in Kubernetes Environments //2025 International Research Conference on Smart Computing and Systems Engineering (SCSE). – IEEE, 2025. – С. 1-5.
10. Moriabadi M. Reducing carbon emissions in Kubernetes clusters through temporal and spatial workload management: дис. – Universitätsbibliothek der Universität Stuttgart, 2023.
11. Vogl R., Blank-Burian M. An Update on the Münster University Cloud-technical architecture and user adoption //Proceedings of EUNIS. – 2025. – Т. 105. – С. 126-135.
12. Moritz de Carvalho Neto P., Castro M., Siqueira F. Dynamic Load Balancing in Kubernetes Environments with Kubernetes Scheduling Extension (KSE) //Concurrency and Computation: Practice and Experience. – 2025. – Т. 37. – №. 3. – С. e8344.
13. Rejiba Z., Chamanara J. Custom scheduling in kubernetes: A survey on common problems and solution approaches //ACM Computing Surveys. – 2022. – Т. 55. – №. 7. – С. 1-37.
14. Ascensão P. et al. Assessing kubernetes distributions: A comparative study //2024 IEEE 22nd Mediterranean Electrotechnical Conference (MELECON). – IEEE, 2024. – С. 832-837.
15. БАСЫНЯ Е. А., САПЕГИН В. Ю. КИБЕРНЕТИКА И ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ" КИБ-2025 //Москва. – 2025. – Т. 3. – С. 130-131.
16. Стрельцов А. С., Французова Г. А., Басыня Е. А. Разработка системы сбора, обработки, анализа, идентификации корреляции событий информационной инфраструктуры предприятия //Системы анализа и обработки данных. – 2023. – №. 1 (89). – С. 101-113.
17. Басыня Е. А. Распределенная система сбора, обработки и анализа событий информационной безопасности сетевой инфраструктуры предприятия / Безопасность информационных технологий. 2018. Т. 25. № 4. С. 43-52.

© И. В. Чугунков, М. М. Ровнягин, К. А. Федоров, К. В. Сысоев, 2026