Княжев Алексей Викторович, студент
Московский государственный технический университет

МЕТОДЫ ПЛАНИРОВАНИЯ ЗАПУСКА ПРИЛОЖЕНИЙ ПРИМЕНИМЫЕ В СИСТЕМЕ ОРКЕСТРАЦИИ КОНТЕЙНЕРОВ KUBERNETES

Аннотация. Проведен обзор архитектуры системы оркестрации контейнеров Kubernetes и механизма работы планировщика kube-scheduler. Описаны методы, которые можно применить для решения задачи планирования запуска приложений в распределенной системе. Проведено сравнение описанных методов.

Ключевые слова: Планировщик, Kubernetes, распределенная система, кластер, оркестрация, контейнеризация.

Введение. На данный момент многие организации выбирают микросервисный подход к разработке информационных систем. Для запуска приложений используется технология контейнеризации [1]. Системы оркестрации контейнеров, такие как Kubernetes, который фактически стал стандартом индустрии, упрощают развертывание контейнеризированных приложений. Ключевой задачей оркестратора является распределение физических ресурсов вычислительного кластера между контейнерами [2].

В общем случае, задачу распределения ресурсов в вычислительном кластере можно описать так: необходимо выбрать, какой рабочий узел будет выполнять условное задание. В основном, под рабочим узлом подразумевается физический сервер, а под заданием – приложение [3].

Задача распределения ресурсов в вычислительном кластере является NP-трудной [4, 5] и не может быть решена за полиномиальное время [6].

Для задачи планирования традиционным решением являются эвристические алгоритмы [6]. Эвристические алгоритмы нацелены на универсальность, а также легко понимаются и реализуются. Примеры таких алгоритмов включают «Round-Robin» [7], «First Fit» [8] и «Max-Min» [9], которые хорошо работают при определённых нагрузках. Однако эвристические алгоритмы обычно используют фиксированные параметры и политики планирования. Они не могут адаптироваться к изменяющейся среде, из-за чего не достигают оптимальной производительности.

Некоторые исследования предлагают использовать метаэвристические алгоритмы для улучшения эвристических подходов. Эти алгоритмы комбинируют эвристику с рандомизированными алгоритмами и алгоритмами локального поиска, такими как алгоритм муравьиной колонии, алгоритм имитации отжига и генетический алгоритм [10]. Однако разработка этих метаэвристик является сложной и трудоёмкой задачей. Как правило, сначала разрабатывается простой эвристический алгоритм в соответствии с данным сценарием планирования. Затем параметры эвристического алгоритма пытаются настроить вручную в зависимости от характеристик приложений и целей планирования. Этот процесс требует многократного ручного тестирования и настройки, а также наличия экспертных знаний у разработчика о состоянии кластера и его рабочей нагрузке [11].

Архитектура Kubernetes. Kubernetes – система для запуска контейнеризированных приложений (оркестратор) с открытым исходным кодом, позволяет управлять жизненным циклом работы контейнеров: запускать, удалять, перезапускать в случае возникновения ошибок [12]. При этом пользователь абстрагирован от внутренней логики выполнения приложений, такой как расположение контейнера на каком-то из физических узлов кластера.

Основным понятием в Kubernetes является ресурс. Ресурс – описание некоторого типа объекта в Kubernetes. Все понятия, с которыми оперирует оркестратор, представлены в виде объектов, и хранятся в базе данных [13]. Для того, чтобы избежать коллизии понятий, в для задачи распределения ресурсов вычислительном кластере будет использоваться термин «задача планирования запуска приложений в кластере».

Kubernetes оперирует со следующими основными типами ресурсов.

- *Pod (под)* описывает экземпляр приложения. В рамках одного пода может быть запущено несколько связанных контейнеров. Является минимальной единицей планирования в Kubernetes, то есть все контейнеры одного пода обязательно будут запущены на одном узле кластера [14].
- *ReplicaSet (набор реплик)* описывает набор реплик запускаемого приложения. На основе него Kubernetes создает поды в количестве, указанном в свойствах объекта [15].
- *Deployment* описывает приложение без сохранения состояния. На основе него Kubernetes создает набор реплик и регулирует процесс обновления этого набора [16].
- *PersistentVolume (том)* описывает том постоянной области данных, это может быть область диска узла кластера, сетевое хранилище, и так далее [17]. Так как том постоянно области данных может быть реализован по-разному, в Kubernetes существует спецификация Container Storage Interface, через которую оркестратор создает тома. Производители оборудования и разработчики могут создавать собственные реализации CSI [18].
- *PersistentVolumeClaim (запрос на том)* описывает пользовательский запрос на том постоянного хранилища данных. Пользователь кластера, если ему необходимо постоянно хранить данные не взаимодействует напрямую с *PersistentVolume*, а создает запрос на том [17].
- *StatefulSet* описывает приложение с сохранением состояния. На основе него Kubernetes создает запросы на тома и поды в количестве, указанном в спецификации объекта. Каждый под будет использовать уникальный запрос на том [19].

Для описания и отображения объектов пользователю используются манифесты. Манифест – текстовое описание объекта в Kubernetes в формате *YAML* [13]. Пример манифеста пода представлен в листинге 1.

Листинг 1

Пример манифеста пода

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: app
    image: app:v4
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
  - name: log-aggregator
    image: log-aggregator:v6
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

В разделе *spec* содержится спецификация пода (экземпляра приложения). Спецификация пода содержит [14]:

- *containers* – спецификации контейнеров, запускаемых в этом поде, для каждого контейнера могут быть указаны запросы и ограничения на ресурсы (resources), используемые тома (volumes), названия образа Docker (image), название (name);
- *volumes* – используемые контейнерами тома;
- *tolerations* – допущения к рабочим узлам. Например, если рабочий узел содержит в спецификации ограничение (taint) вида *network=low*, то на нем могут быть запущены только те поды, в спецификации которых есть допущение к этому свойству [20];
- *affinity* – правила, влияющие на планирование подов, например, запрет планирования подов приложения на одних узлах с другими экземплярами этого приложения.

Система оркестрации контейнеров Kubernetes содержит следующие модули, поставляемые в формате независимых исполняемых файлов [21].

- *kube-apiserver* – сервис, представляющий Kubernetes API. Работа со всеми объектами, хранящимися в Kubernetes, происходит посредством запросов в *kube-apiserver*. Кроме информации о всех объектах, сервис управляет информацией о состояниях этих объектов: статусах работы контейнеров, доступными ресурсами узлов кластера, состояниях узлов кластера и пр.
- *etcd* – распределенная база данных формата ключ-значение, в которой *kube-apiserver* хранит информацию о всех объектах кластера [22].
- *kube-scheduler* – планировщик Kubernetes. При создании нового пода, получает информацию о запросах на ресурсы, о свободных ресурсах узлов из *kube-apiserver*, определяет, на каком узле будет запущено приложение.
- *kube-controller-manager* – приложение, обрабатывающее такие ресурсы Kubernetes, как *Deployment*, *ReplicaSet* и другие. При создании этих объектов, получает информацию о них в *kube-apiserver*, и производит соответствующую обработку. Например, для *ReplicaSet* создаст заданное в спецификации объекта количество подов.
- *kubelet* – демон, запускаемый на каждом рабочем узле кластера Kubernetes. Запускает контейнеры, относящиеся к подам, запланированным для запуска на данном узле [23].

Планирование запуска приложений в Kubernetes. За планирование запуска приложений в Kubernetes отвечает компонент *kube-scheduler*. При создании нового пода, под переходит в состояние *Pending*. Поды в этом состоянии из *kube-apiserver* получает *kube-scheduler*, и на основании заданных правил и доступных ресурсов на рабочих узлах, информацию о которых планировщик также получает из *kube-scheduler*, назначает под на подходящий рабочий узел кластера [24].

Планирование приложений состоит из двух этапов [2, 25]: фильтрация и ранжирование.

При фильтрации из потенциально подходящих рабочих узлом исключаются те, которые не удовлетворяют заданным условиям, которые иногда называют предикатами [25]. Например, *kube-scheduler* поддерживает следующие модули, реализующие проверки предикатов [26].

- *TaintToleration* – допустимыми считаются те узлы, которые не содержат таких ограничений, допущений к которым нет в спецификации подов.
- *NodeName* – допустимым считается только тот узел, имя которого задано в спецификации пода.
- *NodePorts* – допустимыми считаются только те узлы, у которых доступны указанные порты.
- *NodeAffinity* – допустимыми считаются только те узлы, которые удовлетворяют условиям, заданным в спецификации пода.
- *PodTopologySpread* – допустимость узлов определяется географическим расположением узлов.

- *NodeUnschedulable* – допустимыми считаются узлы, доступные для планирования.
- *NodeResourcesFit* – допустимыми считаются те узлы, каждый из доступных ресурсов которых больше соответствующих запрашиваемых ресурсов пода.
- *VolumeBinding* – допустимыми считаются те узлы, которые содержат нужные поду тома данных.
- *VolumeZone* – допустимыми считаются те узлы, на которых есть тома, расположенные в нужной геозоне.
- *NodeVolumeLimits* – допустимыми считаются те узлы, на которых достаточное количество ресурсов по хранению данных.
- *InterPodAffinity* – допустимость узлов определяется, исходя их расположения экземпляров приложения друг относительно друга. Например, у экземпляров приложения может быть установлено правило, что они не могут быть запущены на одном узле. В этом случае допустимыми будут считаться те узлы, на которых не запущены экземпляры указанного приложения.

Фильтрация рассчитана на отсечение заведомо неподходящих узлов, поэтому их нет смысла как-либо оптимизировать: нельзя изменить фильтрацию так, чтобы допустимыми считались узлы, которые не подходят для запуска приложения.

Основные исследования в области распределения ресурсов связаны с задачей ранжирования [27]. При ранжировании происходит приоритизация наиболее подходящих узлов. Далее будут рассмотрены методы, связанные с ранжированием подходящих для выполнения приложения узлов.

Методы планирования запуска приложений. Подразумевается, что все нижеперечисленные методы могут быть реализованы, или уже реализованы в планировщике Kubernetes. Планировщик представляет собой отдельный сервис, который получает информацию об ожидающих планирования подах и свободных ресурсах на рабочих узлах из *kube-apiserver*.

Метод случайного распределения. С использованием данного метода, в качестве рабочего узла для выполнения задачи выбирается случайный узел [28]. Так как планировщик получает из *kube-apiserver* список рабочих узлов кластера, затем фильтрует его, то из списка оставшихся потенциально подходящих узлов можно выбрать номер узла с использованием генератора псевдослучайных чисел. Так как на вход алгоритму ранжирования не могут поступить узлы, на которых под не может быть запущен, можно использовать любой метод генерации псевдослучайных чисел [29].

К данной категории также можно отнести метод «Round Robin» [30]: для задачи выбирается следующий доступный в кольцевой очереди узел.

Данные методы нельзя в полной мере считать подходящими для целевой задачи, так как они не учитывают распределение ресурсов, текущую нагрузку на узлах, запросы задачи на ресурсы.

Выбор наименее запрашиваемого узла. В качестве подходящего рабочего узла выбирается наименее запрашиваемый узел [31, 29]. Данный метод не учитывает распределение ресурсов, текущую нагрузку на узлах, запросы задачи на ресурсы. При этом, за счет выбора наименее запрашиваемого ресурса в некоторых случаях позволяет равномерно распределить нагрузку между рабочими узлами, например в том случае, когда узлы имеют одинаковые ресурсы, а задачи одинаковые запросы на ресурсы [32].

Выбор наиболее запрашиваемого узла. В качестве подходящего рабочего узла выбирается наиболее запрашиваемый узел [31]. В отличие от метода выбора наименее запрашиваемого узла, такой метод не распределяет нагрузку равномерно: узлы заполняются задачами «по очереди», задачи начнут запускаться на узле только в том случае, когда остальные узлы максимально загружены и не могут принять ее. При этом, метод также не учитывает распределение ресурсов, текущую нагрузку на узлах, запросы задачи на ресурсы.

Метод сбалансированного распределения. Для выбора подходящего узла рассчитывается коэффициент загрузки задачей свободных ресурсов для каждого узла [33, 34]:

$$Load_{ij} = \frac{Requests_j}{Available_{ij}},$$

где

- K – количество системных ресурсов;
- N – количество доступных рабочих узлов;
- $Load_{N \times K}$ – матрица, в которой содержатся потенциальные коэффициенты загрузки рабочего узла задачей для каждого вида ресурсов, строка соответствует рабочему узлу, столбец – ресурсу;
- $Requests_K$ – вектор, содержащий запросы на ресурсы приложением;
- $Available_{N \times K}$ – матрица, в которой содержатся доступные ресурсы рабочих узлов, строка соответствует рабочему узлу, столбец – ресурсу;
- i – номер рабочего узла;
- j – номер ресурса.

Зачем рассчитывается вектор, содержащий средние показатели загрузки каждого узла:

$$Mean_i = \frac{\sum_{j=1}^{K} Load_{ij}}{K}.$$

На основе средних показателей загрузки можно рассчитать оценку для каждого узла:

$$Score_i = \sqrt{\frac{\sum_{j=1}^{K} \left(Load_{ij} - Mean_i\right)^2}{K}}.$$

Подходящим узлом считается узел с наибольшей оценкой. Таким образом, метод учитывает текущую загрузку узлов и запросы на ресурсы задачей. Данный метод стремится равномерно распределить ресурсы узлов, например, таким образом, чтобы процессор и память на узлах были загружены одинаково.

Метод используется в Kubernetes по умолчанию [34].

Метод сбалансированного распределения по доминантному ресурсу. На практике вышеизложенный метод может приводить к неравномерной нагрузке рабочих узлов [35]: на одном узле может быть полностью загружен процессор, а на другом – память.

Например, в кластере, состоящем из трех серверов:

1) *server 1* – CPU: 6000 единиц процессорного времени, Memory: 4.5ГБ;
2) *server 2* – CPU: 3000 единиц процессорного времени, Memory: 3ГБ;
3) *server 3* – CPU: 3000 единиц процессорного времени, Memory: 6ГБ;

пользователь пытается запустить семь подов в указанном порядке:

1) *приложение 1* – запрашивает CPU: 2500 единиц процессорного времени, Memory: 1ГБ;
2) *приложение 2* – запрашивает CPU: 1000 единиц процессорного времени, Memory: 2.5ГБ;
3) *приложение 3* – запрашивает CPU: 2000 единиц процессорного времени, Memory: 1ГБ;
4) *приложение 4* – запрашивает CPU: 1000 единиц процессорного времени, Memory: 2.5ГБ;
5) *приложение 5* – запрашивает CPU: 2000 единиц процессорного времени, Memory: 1ГБ;
6) *приложение 6* – запрашивает CPU: 1000 единиц процессорного времени, Memory: 2ГБ;
7) *приложение 7* – запрашивает CPU: 1000 единиц процессорного времени, Memory: 2.5ГБ.

Результат работы метода сбалансированного распределения проиллюстрирован на рисунке 1, приложения отмечены цветными фигурами, где высота фигуры – потребление ресурса системы, цифры – порядковые номера, в которых запускались приложения.

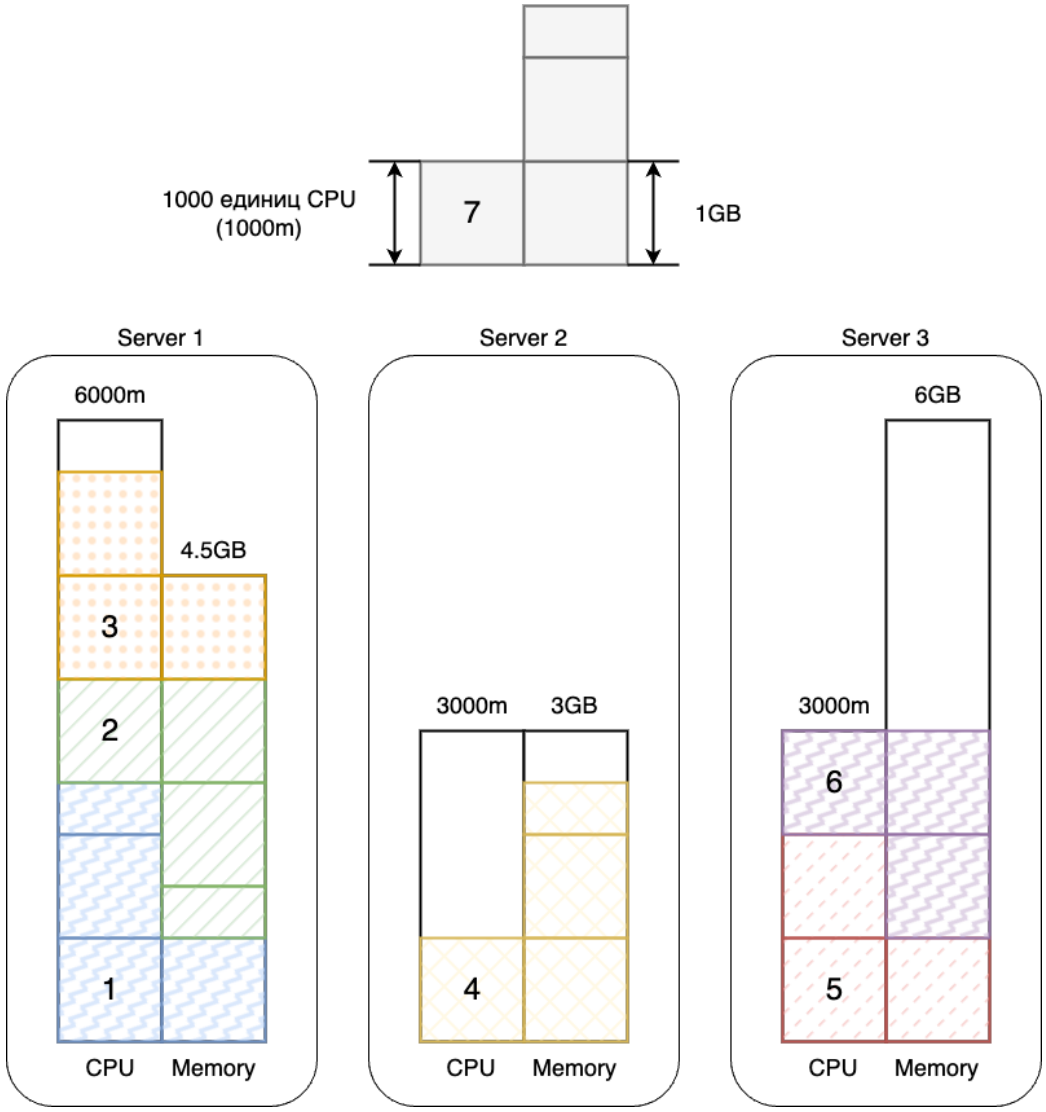

Рисунок 1
Возможное распределение приложений по серверам
при использовании метода сбалансированного распределения

Видно, что приложение №7 не может быть запущено в кластере, так как на всех серверах недостаточно свободных ресурсов. При этом, сервера имеют следующие относительные показатели загрузки, округленные до целых:

- *server 1* – CPU: 92%, Memory: 100%;
- *server 2* – CPU: 33%, Memory: 83%;
- *server 3* – CPU: 100%, Memory: 50%.

Для того, чтобы решить эту проблему, существует модификация этого метода. Его отличие заключается в том, что для оценки каждого узла учитывается только доминантный ресурс, то есть оценка узлов происходит на основе коэффициента $Dominant$ [36], расчитываемого для каждого узла i:

$$Dominant_i = \max_{j=1}^{K} Loadij.$$

Подходящим считается узел с наименьшим $Dominant$. Иллюстрация работы данного метода приведена на рисунке 2.

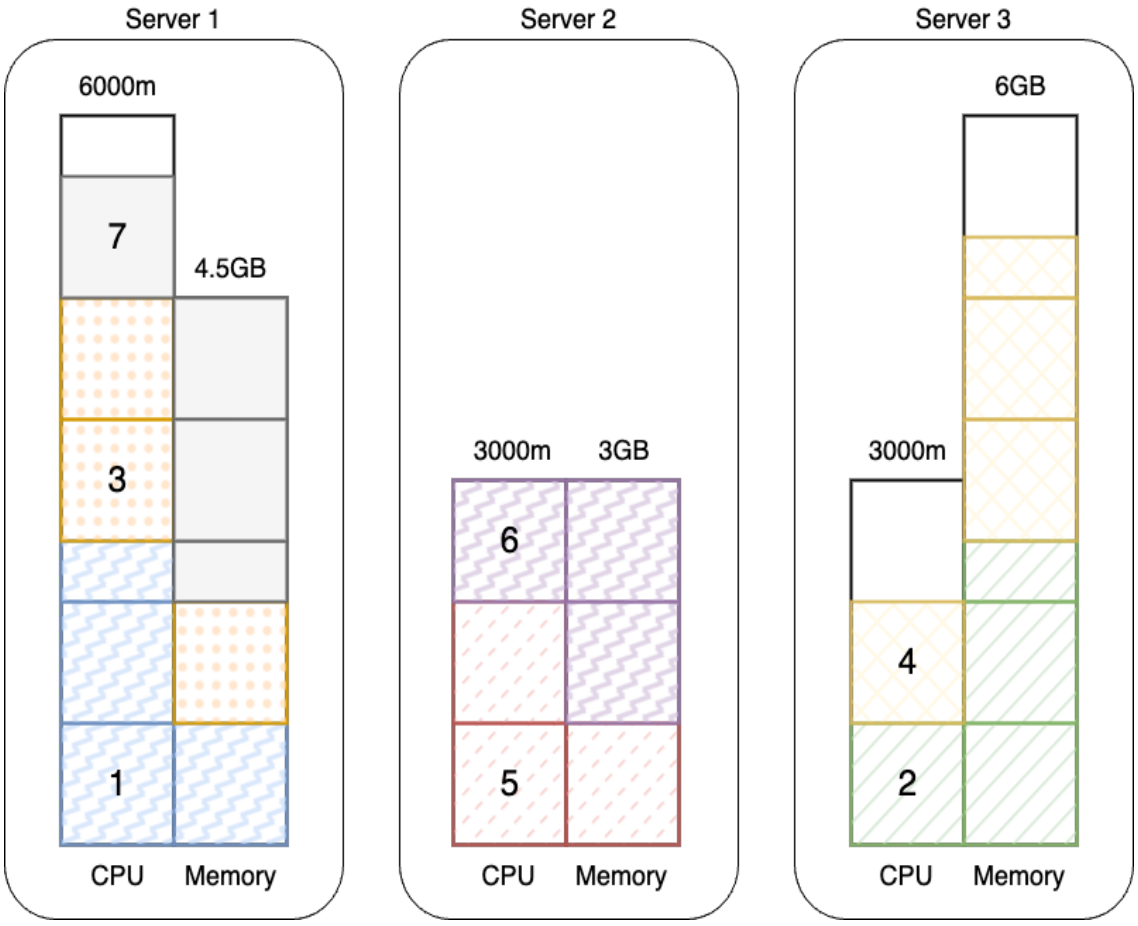

Рисунок 2
Возможное распределение приложений по серверам при использовании метода сбалансированного распределения по доминантному ресурсу

В данном примере, приложение №7 было запущено, а сервера имеют следующие относительные показатели загрузки, округленные до целых:

- *server 1* – CPU: 92%, Memory: 100%;
- *server 2* – CPU: 100%, Memory: 100%;
- *server 3* – CPU: 67%, Memory: 83%.

Генетический алгоритм. Генетический алгоритмы – поисковые алгоритмы, основанные на механизмах натуральной селекции и натуральной генетики [37]. Случайным образом выбирается какое-либо решение, далее происходит череда мутаций, в процессе которых решение незначительно изменяется с использованием элементарных операций. Полученное решение проверяется на предмет допустимости. В результате многочисленных мутаций выбирается наиболее оптимальное решение.

Важно отметить, что генетические алгоритмы обычно используются при большом количестве учитываемых параметров [38]. В обозреваемой задаче количество параметров можно считать небольшим:

- максимальное количество рабочих узлов в кластере Kubernetes – 5000 [39], однако на практике количество рабочих узлов может быть ограничено меньшими значениями, например, 512 [40];
- в стандартном случае при планировании подов учитываются два системных ресурса – процессор и память [41], однако пользователи кластера могут задавать пользовательские типы системных ресурсов, например, пропускную способность дискового ввода-вывода [42].

Нейронные сети. Нейронные сети – вычислительные системы или машины, созданные для моделирования аналитических действий, совершаемых человеческим мозгом.

Нейронные сети относятся к направлению искусственного интеллекта (ИИ) и применяются для распознавания скрытых закономерностей в необработанных данных, группировки и классификации, а также решения задач в области ИИ, машинного и глубокого обучения.

Искусственные нейронные сети состоят из нескольких слоев: входных, скрытых, выходных. В каждом из них есть несколько узлов, которые соединены со всеми узлами в сети с помощью разных связей и имеют свой «вес», влияющий на силу передаваемого сигнала. Такая архитектура позволяет вести параллельную обработку данных и постоянно сравнивать их с результатами обработки на каждом из этапов. Нейронные сети изначально обучаются на размеченных наборах данных с очевидными закономерностями, а после используют полученные навыки для самообучения и достижения результата. При этом нейросеть может совершать миллионы попыток для достижения таких же результатов, как и предоставленном для обучения примере [43].

В данном случае, входными данными для нейронной сети будут запросы на ресурсы приложения, и доступные ресурсы рабочих узлов. Результатом работы нейронной сети будет номер наиболее подходящего узла.

Важно отметить, что для обучения нейронной сети необходима большая выборка [44], которых на данных момент не существует в открытом доступе для указанной задачи. Кроме того, тестировать и отлаживать компоненты, использующие нейронные сети, крайне сложно [45].

Метод выселения. Так как такие системы, как Kubernetes, являются очень динамичными, то может возникнуть ситуация, когда в какой-то момент времени уже существующее распределение задач по узлам может оказаться неоптимальным. Например, когда не хватает ресурсов для запуска какого-либо приложения, при этом если перераспределить задачи по-другому, то задача может быть запущена [46]. Для таких случаев существует метод выселения, который заключается в том, что при обнаружении неоптимального распределения задач в кластере, происходит их перераспределение. В некоторых случаях использование выселения позволяет сократить количество использованных узлов для запуска приложений на 16.7% [47].

Особенность этого метода заключается в том, что он не может быть бесконтрольно применяем в рамках кластера, так как бесконтрольные выселения приложений, по сути представляющие собой перезапуски, могут негативно влиять на доступность системы.

Кроме того, для баз данных, перезапуск одной из реплик, в случае недоступности нескольких других реплик, может привести к недоступности всей базы данных [48].

Выбор критериев для сравнения. Для сравнения вышеперечисленных методов использовались следующие критерии.

- Учет запросов на ресурсы задачей – метод не может эффективно планировать задачи, если не учитывает потребности приложения в ресурсах.
- Учет доступных ресурсов рабочих узлов – метод не может эффективно планировать задачи, если не учитывает доступные на рабочих узлах ресурсы.
- Учет доминантного ресурса – на практике, методы, не учитывающие доминантный ресурс, могут неэффективно распределять приложения по рабочим узлам.
- Возможность перераспределения – вычислительные кластеры, в которых запускаются приложения, являются очень динамичной средой, поэтому распределение, оптимальное в какой-либо момент, может перестать таковым быть. Это может быть связано, например, с окончанием работы какого-либо приложения, значительно нагружающего систему. По этой причине, возможность перераспределения является важной составляющей распределения ресурсов.

Сравнение методов. В таблице 1 приведено сравнение методов распределения ресурсов в вычислительном кластере.

Таблица 1

Сравнение методов распределения ресурсов в вычислительном кластере

Метод	Критерий сравнения			
	Учет запросов на ресурсы	Учет доступных ресурсов	Учет доминантного ресурса	Возможность перераспределения
Случайное распределение	Нет	Нет	Нет	Нет
Наименее запрашиваемый	Нет	Нет	Нет	Нет
Наиболее запрашиваемый	Нет	Нет	Нет	Нет
Сбалансированное	Да	Да	Нет	Нет
Сбалансированное с доминантным ресурсом	Да	Да	Да	Нет
Генетический алгоритм	Да	Да	Нет	Нет
Нейронные сети	Да	Да	Да	Нет
Выселение	Нет	Нет	Нет	Да

На основании приведенного сравнения было решено выбрать комбинацию методов сбалансированного распределения по доминантному ресурсу и выселения для дальнейшего изучения.

Заключение. Проведен обзор архитектуры системы оркестрации контейнеров Kubernetes и механизма работы планировщика *kube-scheduler*. Описаны методы, которые можно применить для решения задачи планирования запуска приложений в распределенной системе: метод случайного распределения, выбор наименее запрашиваемого узла, выбор наиболее запрашиваемого узла, метод сбалансированного распределения, метод сбалансированного распределения по доминантному ресурсу, генетический алгоритм, нейронные сети и выселение. Показано, что текущая реализация *kube-scheduler*, основанная на методе сбалансированного распределения, не всегда позволяет оптимально использовать ресурсы кластера, так как в результате работы алгоритма не все приложения могут быть запущены. В некоторых случаях, сбалансированное распределение по доминантному ресурсу позволяет более экономично использовать ресурсы кластера. Такие методы, как генетические алгоритмы, хорошо показывают себя при большом количестве параметров и обширной обучающей выборке, что не характерно для указанной задачи. По результатам сравнения принято решение в дальнейшем рассматривать подробнее комбинацию методов сбалансированного распределения по доминантному ресурсу и выселения.

Список литературы:

1. Кочер П. С. Микросервисы и контейнеры Docker. – Litres, 2022.
2. Carri´on C. Kubernetes scheduling: Taxonomy, ongoing issues and challenges // ACM Computing Surveys. – 2022. – Т. 55, № 7. – С. 1 – 37.
3. Scheduling in distributed systems: A cloud computing perspective / L. F. Bittencourt [и др.] // Computer science review. – 2018. – Т. 30. – С. 31 – 54.
4. Mor B., Shabtay D., Yedidsion L. Heuristic algorithms for solving a set of NP-hard single-machine scheduling problems with resource-dependent processing times // Computers & Industrial Engineering. – 2021. – Т. 153. – С. 107024.
5. Cloud resource scheduling with deep reinforcement learning and imitation learning / W. Guo [и др.] // IEEE Internet of Things Journal. – 2020. – Т. 8, № 5. – С. 3576 – 3586.
6. Marrouche W. Unlocking the Potential of Metaheuristics for NP-Hard Problems: дис.... канд. / Marrouche Wissam. – University of Portsmouth, 2024.
7. An ameliorated Round Robin algorithm in the cloud computing for task scheduling / N. Ghazy [и др.] // Bulletin of Electrical Engineering and Informatics. – 2023. – Т. 12, № 2. – С. 1103 – 1114.

8. Keshri R., Vidyarthi D. P. Communication-aware, energy-efficient VM placement in cloud data center using ant colony optimization // International Journal of Information Technology. – 2023. – Т. 15, № 8. – С. 4529 – 4535.

9. Advanced cost-aware Max–Min workflow tasks allocation and scheduling in cloud computing systems / M. Raeisi-Varzaneh [и др.] // Cluster computing. – 2024. – С. 1 – 13.

10. Mishra A., Goel L. Metaheuristic algorithms in smart farming: An analytical survey // IETE Technical Review. – 2024. – Т. 41, № 1. – С. 46 – 65.

11. DRS: A deep reinforcement learning enhanced Kubernetes scheduler for microservice-based system / Z. Jian [и др.] // Software: Practice and Experience. – 2024. – Т. 54, № 10. – С. 2102 – 2126.

12. Главная страница официальной документации Kubernetes [Электронный ресурс]. – URL: https://kubernetes.io/docs/home (дата обращения: 04.12.2024).

13. Официальная документация по объектам в Kubernetes [Электронный ресурс]. – URL: https://kubernetes.io/docs/concepts/overview/working-with-objects (дата обращения: 04.12.2024).

14. Официальная документация по Pod в Kubernetes [Электронный ресурс]. – URL: https://kubernetes.io/docs/concepts/workloads/pods (дата обращения: 04.12.2024).

15. Официальная документация по ReplicaSet в Kubernetes [Электронный ресурс]. – URL: https://kubernetes.io/docs/concepts/workloads/controllers/replicaset (дата обращения: 04.12.2024).

16. Официальная документация по Deployment в Kubernetes [Электронный ресурс]. – URL: https://kubernetes.io/docs/concepts/workloads/controllers/deployment (дата обращения: 04.12.2024).

17. Официальная документация по PersistentVolume в Kubernetes [Электронный ресурс]. – URL: https://kubernetes.io/docs/concepts/storage/persistent-volumes (дата обращения: 04.12.2024).

18. Анонс в официальном блоге Kubernetes про CSI [Электронный ресурс]. – URL: https://kubernetes.io/blog/2019/01/15/container-storage-interface-ga (дата обращения: 04.12.2024).

19. Официальная документация по StatefulSet в Kubernetes [Электронный ресурс]. – URL: https://kubernetes.io/docs/concepts/workloads/controllers/statefulset (дата обращения: 04.12.2024).

20. Официальная документация по механизмам taints и tolerations [Электронный ресурс]. – URL: https://kubernetes.io/docs/concepts/scheduling-eviction/taint-and-toleration/ (дата обращения: 04.12.2024).

21. Обзор архитектуры Kubernetes [Электронный ресурс]. – URL: https://kubernetes.io/docs/concepts/architecture/ (дата обращения: 04.12.2024).

22. Официальная документация etcd [Электронный ресурс]. – URL: https://etcd.io/docs/ (дата обращения: 04.12.2024).

23. Официальная документация kubelet [Электронный ресурс]. – URL: https://kubernetes.io/docs/reference/command-line-tools-reference/kubelet/ (дата обращения: 04.12.2024).

24. Официальная документация kube-scheduler [Электронный ресурс]. – URL: https://kubernetes.io/docs/reference/command-line-tools-reference/kube-scheduler/ (дата обращения: 04.12.2024).

25. 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.

26. Официальная документация по профилям планировщика Kubernetes [Электронный ресурс]. – URL: https://kubernetes.io/docs/reference/scheduling/config/#profiles (дата обращения: 11.12.2024).

27. A survey of Kubernetes scheduling algorithms / K. Senjab [и др.] // Journal of Cloud Computing. – 2023. – Т. 12, № 1. – С. 87.

28. Psychas K., Ghaderi J. Randomized algorithms for scheduling multi-resource jobs in the cloud // IEEE/ACM Transactions on Networking. – 2018. – Т. 26, № 5. – С. 2202 – 2215.

29. K8sSim: A Simulation Tool for Kubernetes Schedulers and Its Applications in Scheduling Algorithm Optimization / S. Wen [и др.] // Micromachines. – 2023. – Т. 14, № 3. – С. 651.

30. Singh A., Goyal P., Batra S. An optimized round robin scheduling algorithm for CPU scheduling // International Journal on Computer Science and Engineering. – 2010. – Т. 2, № 07. – С. 2383 – 2385.

31. Liu K., Lee V. C. Performance analysis of data scheduling algorithms for multi-item requests in multi-channel broadcast environments // International journal of communication systems. – 2010. – Т. 23, № 4. – С. 529 – 542.

32. Beltre A., Saha P., Govindaraju M. Kubesphere: An approach to multi- tenant fair scheduling for kubernetes clusters // 2019 IEEE cloud summit. – IEEE. 2019. – С. 14 – 20.

33. Balanced resource allocations across multiple dynamic MapReduce clusters / B. Ghit [и др.] // The 2014 ACM international conference on Measurement and modeling of computer systems. – 2014. – С. 329 – 341.

34. Исходный код планировщика Kubernetes: сбалансированное распределение [Электронный ресурс]. – URL: https://github.com/kubernetes/kubernetes/blob/master/pkg/scheduler/framework/plugins/noderesources/balanced_allocation.go (дата обращения: 11.12.2024).

35. El Haj Ahmed G., Gil-Casti˜neira F., Costa-Montenegro E. KubCG: A dynamic Kubernetes scheduler for heterogeneous clusters // Software: Practice and Experience. – 2021. – Т. 51, № 2. – С. 213 – 234.

36. Wang W., Li B., Liang B. Dominant resource fairness in cloud computing systems with heterogeneous servers // IEEE INFOCOM 2014-IEEE Conference on Computer Communications. – IEEE. 2014. – С. 583 – 591.

37. Курейчик В. М. Генетические алгоритмы // Известия Южного федерального университета. Технические науки. – 1998. – Т. 8, № 2. – С. 4 – 7.

38. Гладков Л., Курейчик В., Курейчик В. Генетические алгоритмы. – Litres, 2022.

39. Официальная документация по обслуживанию больших кластеров Kubernetes [Электронный ресурс]. – URL: https://kubernetes.io/docs/setup/best-practices/cluster-large/ (дата обращения: 04.12.2024).

40. Официальная документация по ограничениям Kubernetes от DigitalOcean [Электронный ресурс]. – URL: https://docs.digitalocean.com/products/kubernetes/details/limits/ (дата обращения: 04.12.2024).

41. Официальная документация по по управлению ресурсами в Kubernetes [Электронный ресурс]. – URL: https://kubernetes.io/docs/concepts/configuration/manage-resources-containers (дата обращения: 04.12.2024).

42. Доклад «Как достать шумного соседа: ограничение дискового ввода-вывода в Kubernetes» на конференции Positive Hack Days 2 [Электронный ресурс]. – URL: https://www.youtube.com/watch?v=GJqzKRZg3a8 (дата обращения: 04.12.2024).

43. Cloud.ru: Нейронные сети [Электронный ресурс]. – URL: https://cloud.ru/blog/neural-networks (дата обращения: 11.12.2024).

44. Elman J. L. Learning and development in neural networks: The importance of starting small // Cognition. – 1993. – Т. 48, № 1. – С. 71 – 99.

45. Testing deep neural networks / Y. Sun [и др.] // arXiv preprint arXiv:1803.04792. – 2018.

46. Github: descheduler [Электронный ресурс]. – URL: https://github.com/kubernetes-sigs/descheduler (дата обращения: 11.12.2024).

47. Larsson O., Klein C., Elmroth E. The Impact of Directed Pod Eviction on Kubernetes Resource Utilization // 2023 IEEE International Conference on Service-Oriented System Engineering (SOSE). – IEEE. 2023. – С. 81 – 90.

48. Ongaro D., Ousterhout J. The raft consensus algorithm // Lecture Notes CS. – 2015. – Т. 190. – С. 2022.