УДК 681.32

ПРИМЕНЕНИЕ АЛГОРИТМОВ ДИСКРЕТНОЙ МАТЕМАТИКИ ДЛЯ ПЛАНИРОВАНИЯ РАСПИСАНИЯ ПРИ РАЗВЕРТЫВАНИИ ПРИЛОЖЕНИЙ ИСПОЛЬЗУЯ СРЕДУ KUBERNETES

Хадорик М.Д, студент гр.453503

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Егорова Н.Г. – канд. технических наук, доцент

Аннотация. Приведен способ применение алгоритмов дискретной математики для решения задачи планирования маршрутов и последовательностей развертывания микросервисов в распределённых системах на базе Kubernetes. Основное внимание уделено топологической сортировке, как способе для установления порядка запуска сервисов с учётом их зависимостей. Путём моделирования зависимостей в виде ориентированного ациклического графа.

Ключевые слова. Микросервисы, топологическая сортировка, распределенные системы.

В нынешних тенденция популярностью пользуется микросервисная архитектура построения программного обеспечения. Для более автоматизированного развертывания таких систем может использоваться программное обеспечение для аркестровки контейнеров – Kubernetes. Однако возникает проблема: как определить порядок запуска сервисов при наличии зависимостей между ними?

Представим упрощенный интернет-сервис, состоящий из нескольких микросервисов:

– Database: Хранит всю информацию о нашем сервисе в разных SQL таблицах.

– UserService: Управляет информацией о пользователях. Имеет зависимость от Database.

– ProductService: Предоставляет информацию о товарах. Имеет зависимость от Database.

– OrderService: Обрабатывает заказы. Зависит от UserService и ProductService.

– Frontend: Веб-интерфейс для пользователей. Зависит от UserService, ProductService и OrderService.

При попытке запустить все эти сервисы одновременно или в случайном порядке в среде Kubernetes возникнут проблемы. OrderService не сможет запуститься, пока не функционируют UserService и ProductService. Те, в свою очередь, зависят уже от конкретных таблиц в Database. Неправильный порядок запуска может привести к ошибкам, неправильному распределению ресурсов и проблемам. И появляется вопрос, как выбрать последовательность для запуска микросервисов.

Зависимости между микросервисами можно представлить с помощью ориентированного ациклического графа (DAG), важно то, что орграф должен быть ациклическим, т.к если UserService зависел бы от OrderService, а OrderService – от UserService, возник бы цикл, который намекает на плохое архитектурное решение. В таком графе: вершины – это микросервисы (Database, UserService и т.д.); ребра – это зависимости. Если сервис A зависит от сервиса B, то строится ребро от A к B. Это означает, что B должен быть запущен до A.

Примерная зависимость для рассматриваемого сервиса представлена на рисунке 1.

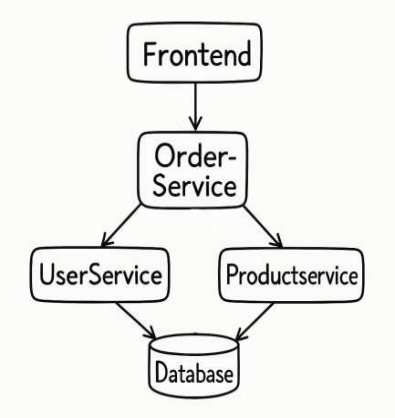

Рисунок 1 – Граф зависимостей нашего сервиса

Топологическая сортировка и ее применения для запуска сервиса:

Топологическая сортировка – это алгоритм, который для заданного DAG строит линейное упорядочение его вершин так, что для каждого ребра (зависимости между сервисами) A-B вершина B в этом порядке идет раньше вершины A. То есть, алгоритм находит последовательность, учитывающая все зависимости.

Один из распространенных методов топологической сортировки (алгоритм Кана [1]) работает следующим образом:

1. Подсчитать "входящую степень" для каждой вершины (в данном случае это кол-во зависимостей)

2. Найти все вершины с входящей степенью 0 (в данном случае это сервисы, которые ни от кого не зависят). Поместить их в очередь для обработки.

3. Пока очередь не пуста:

– Взять вершину U из очереди. Добавить U в итоговый отсортированный список.

– Для каждой вершины V, в которую ведет ребро из U (т.е. для каждого сервиса V, который зависит от U):

– Уменьшить входящую степень V на 1 (выполнена одна из ее зависимостей).

– Если входящая степень V стала равна 0, добавить V в очередь.

1. Результат: Если итоговый список содержит все вершины исходного графа, то это и есть одна из возможных топологических сортировок. Если в конце работы алгоритма в итоговом списке не все вершины, значит в графе был цикл.

Пошагово применим алгоритм к рассмотренному примеру:

1. Начальные состояния:

– Database: 0

– UserService: 1 (от `Database`)

– ProductService: 1 (от `Database`)

– OrderService: 2 (от `UserService`, `ProductService`)

– Frontend: 3 (от `UserService`, `ProductService`, `OrderService`)

2. Очередь: [Database] (единственная вершина со степенью 0).

3. Шаг 1:

– Берем Database. Итоговый список: [Database].

– Уменьшаем степени UserService (1-1=0) и ProductService (1-1=0).

– Добавляем UserService и ProductService в очередь. Очередь: [UserService, ProductService].

4. Шаг 2:

– Берем UserService. Итоговый список: [Database, UserService].

– Уменьшаем степени OrderService (2-1=1) и Frontend (3-1=2).

– Ничья степень не стала 0. Очередь: [`ProductService`].

5. Шаг 3:

– Берем ProductService. Итоговый список: [Database, UserService, ProductService].

– Уменьшаем степени OrderService (1-1=0) и Frontend (2-1=1).

– Добавляем OrderService в очередь (степень стала 0). Очередь: [OrderService].

6. Шаг 4:

– Берем OrderService. Итоговый список: [Database, UserService, ProductService, OrderService].

– Уменьшаем степень Frontend (1-1=0).

– Добавляем Frontend в очередь. Очередь: [Frontend].

7. Шаг 5:

– Берем Frontend. Итоговый список: [Database, UserService, ProductService, OrderService, Frontend].

– Зависимых вершин нет. Очередь пуста.

8. Алгоритм завершен. Список содержит все вершины.

Полученная последовательность Database, UserService, ProductService, OrderService, Frontend является корректным порядком развертывания (заметим, что если бы на шаге 2 был взят ProductService, итоговый порядок мог бы быть другим, но все равно оставался корректным).

Реализованный алгоритм топологической сортировки на языке программирования Golang, представленный на рисунке 2, позволил автоматизировать порядок нахождения правильной последовательности.

```go
// topologicalSort выполняет топологическую сортировку DAG графа зависимостей между сервисами.
// Возвращает порядок развертывания (от зависимостей к зависимым) и успешность/неуспешность сортировки.
func topologicalSort(graph map[string][]string) ([]string, bool) {

  inDegree := map[string]int{} // подсчет входящих рёбер
  adj := map[string][]string{} // обратный граф (от зависимости к зависимым

  // строим обратный граф
  for node, deps := range graph {
    if _, ok := inDegree[node]; !ok {
      inDegree[node] = 0
    }
    for _, dep := range deps {
      adj[dep] = append(adj[dep], node)
      inDegree[node]++
      if _, ok := inDegree[dep]; !ok {
        inDegree[dep] = 0
      }
    }
  }

  // создаем очередь с нодами без входящих рёбер
  queue := []string{}
  for node, degree := range inDegree {
    if degree == 0 {
      queue = append(queue, node)
    }
  }

  var result []string
  for len(queue) > 0 {
    current := queue[0]
    queue = queue[1:]
    result = append(result, current)

    for _, dependent := range adj[current] {
      inDegree[dependent]--
      if inDegree[dependent] == 0 {
        queue = append(queue, dependent)
      }
    }
  }

  // если в результате меньше нод, чем в графе — был цикл
  if len(result) != len(inDegree) {
    return nil, false
  }

  return result, true
}
```

Рисунок 2 – Реализация алгоритма топологической сортировки

Результат топологической сортировки напрямую задает последовательность, в которой инструменты автоматизации (например, скрипты в CI/CD пайплайне) должны применять конфигурации сервисов к Kubernetes (kubectl apply ...). Это гарантирует, что к моменту запуска очередного сервиса его зависимости уже будут развернуты, в CI/CD пайплайнах результат сортировки может использоваться напрямую как последовательность команд, либо как входной файл, по которому скрипт итеративно будет вызывать манифесты. Пример .yaml файла, представленный на рисунке 3, будет представлять часть возможной конфигурации для CI/CD пайплайна, используемого для автоматического развертывания Kubernetes-объектов.

```yaml
deploy:
  stage: deploy
  script:
    - kubectl apply -f manifests/database.yaml
    - kubectl apply -f manifests/userservice.yaml
    - kubectl apply -f manifests/productservice.yaml
    - kubectl apply -f manifests/orderservice.yaml
    - kubectl apply -f manifests/frontend.yaml
  environment:
    name: production
  only:
    - main
```

Рисунок 3 – Примерная реализация скрипта для пайплана

Таким образом, управление зависимостями является сложной задачей при работе с микросервисами. Использование топологической сортировки позволяет: уменьшить вероятность ошибок при развертывании сервиса из-за неудовлетворенных зависимостей, модернизировать сценарии развертывания распределенных систем, выявить потенциально проблемные участки в архитектуре проекта. Это не только повышает надежность процесса, но и служит инструментом для проверки архитектуры на возможную некорректность, конечно не решая всех возможных проблем.

Список использованных источников:

1. Алгоритмы: построение и анализ : учебник / Т. Кормен [и др.] ; пер. с англ. – 3-е изд. – М. : Вильямс, 2013. – Гл. 22. – С. 609–644.

2. Ньюман, С. Создание микросервисов / С. Ньюман – СПб. : Питер, 2016. – 352 с.

3. GeeksforGeeks [Electronic resource] : GeeksforGeeks: Topological Sorting. – Mode of access: https://www.geeksforgeeks.org/topological-sorting/. – Date of access: 13.04.2025.

4. Kubernetes Documentation [Electronic resource]. – Mode of access: https://kubernetes.io/docs/. – Date of access: 13.04.2025.

UDC 681.32

APPLYING DISCRETE MATHEMATICS ALGORITHMS TO SCHEDULING WHEN DEPLOYING APPLICATIONS USING THE KUBERNETES ENVIRONMENT

Khadorik.M.D.

Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Egorova N.G. – Ph.D. in Engineering, Associate Professor

Annotation. A method is given for applying discrete mathematics algorithms to solve the problem of route planning and microservice deployment sequences in distributed Kubernetes-based systems. The main focus is on topological sort ing as a way to establish the order in which services are launched, taking into account their dependencies. By modeling dependencies in the form of a DAG.

Keywords. Microservices, topological sorting, distributed systems.