Выполнение JavaScript-композиций WPS-сервисов в распределенной гетерогенной среде*

И. В. Бычков[1], Г. М. Ружников[2], Р. К. Фёдоров[2], А. С. Шумилов[1,†]

[1]Институт динамики систем и теории управления им. В.М. Матросова СО РАН, Иркутск, Россия

[2]Иркутский научный центр СО РАН, Россия

[†]Контактный e-mail: shumilov@icc.ru

Предлагается использовать язык программирования JavaScript для задания и выполнения композиций сервисов обработки пространственных данных в распределенной гетерогенной среде. Основным отличием данного подхода от существующих является автоматическое планирование вызова сервисов и возможность обработки результатов выполнения сервисов в теле сценария стандартными средствами языка.

Ключевые слова: композиции сервисов, распределенные гетерогенные среды, сервисы, геопортал.

Библиографическая ссылка: Бычков И.В., Ружников Г.М., Фёдоров Р.К., Шумилов А.С. Выполнение JavaScript-композиций WPS-сервисов в распределенной гетерогенной среде // Вычислительные технологии. 2019. Т. 24, № 3. С. 44–58. DOI: 10.25743/ICT.2019.24.3.004.

Введение

В области распределенных вычислений получил широкое распространение сервис-ориентированный подход, позволяющий публиковать различные программы, алгоритмы, источники данных в виде независимых атомарных сервисов [1]. Распределенные сервисы активно применяются для обработки больших объемов пространственных данных с использованием открытых стандартов форматов данных и интерфейсов сервисов OGC (Open Geospatial Consortium). С развитием и увеличением количества тематических сервисов для решения сложных междисциплинарных проблем стало актуальным формирование и выполнение их композиций — объединений существующих сервисов с определенными между ними взаимодействиями, ориентированными на решение конкретных задач [2]. Внутри композиции результаты работы сервисов могут выступать в качестве входных данных для других ее сервисов. Сервисы, участвующие в композиции, могут находиться на разных вычислительных узлах.

Стандартами задания композиций сервисов являются BPEL (Business Process Execution Language), BPMN (Business Process Modeling Notation), BPML (Business Process Management Language) и XPDL (XML Process Definition Language). Для задания композиций сервисов активно используются языки программирования. Применение языков программирования для управления потоками задания (workflow) позволяет обрабатывать промежуточные данные с помощью средств языка и его библиотек, а также

*Title translation and abstract in English can be found on page 57.

использовать промежуточные данные в управляющих конструкциях языка, что значительно упрощает использование сервисов. Существующие программные инструменты, позволяющие задавать композиции сервисов с помощью языков программирования, не производят автоматического планирования вызова сервисов в условиях изменяющейся распределенной среды.

Таким образом, возникает необходимость разработки такого программного средства, которое позволяло бы формировать композиции распределенных сервисов в виде программ и автоматически распараллеливать вызовы сервисов в целях уменьшения общего времени выполнения композиций, а также делало выполнение композиций сервисов устойчивым к изменениям гетерогенной среды.

1. Обзор

Для формирования композиций сервисов существует ряд систем workflow, достигших значительных результатов в различных предметных областях: Pegasus [3], Kepler [4], Swift [5], KNIME [6], Taverna [7], Galaxy [8], Trident [9] and Triana [10], Everest (Mathcloud) [11], CLAVIRE [12]. Для обработки пространственных данных активно используется Geo-Processing Workflows [13]. Все перечисленные системы задают композицию сервисов на уровне заданий и их зависимостей, где задание — это вызов сервиса, а передача данных выполняется с помощью файлов.

Общепризнанным стандартом постановки задачи планирования выполнения композиций сервисов является определение зависимостей между заданиями с помощью направленного ациклического графа (Directed Acyclic Graph, DAG) [14, 15], в котором вершинами являются вызовы сервисов, а дугами — зависимости между сервисами. Разработано много алгоритмов планирования композиций сервисов, основанных на DAG, которые по способу нахождения расписания можно разделить на две группы: эвристические и метаэвристические алгоритмы. Эвристические алгоритмы обычно основаны на списковых алгоритмах, сортирующих список заданий на основе определенной величины, рассчитываемой с помощью эвристики. Наиболее эффективными списковыми алгоритмами являются: HEFT (Heterogeneous Earliest Finish Time) [16], SDBATS (Standard Deviation Based Algorithm for Task Scheduling) [17], PEFT (Predict Earliest Finish Time) [18] и HSIP (Heterogeneous Scheduling with Improved Task Priority) [19].

Среди метаэвристических алгоритмов активно используются генетические алгоритмы, алгоритмы муравьиной колонии, имитации отжига и др. Генетические алгоритмы определяют приближенные решения из широкого пространства поиска, применяя эволюционные принципы, причем они обычно находят более точные решения, нежели эвристические алгоритмы, но проигрывают по времени поиска [20].

В междисциплинарных научных исследованиях отмечается тенденция использования алгоритмов обработки данных, в том числе пространственных, в виде сервисов или их композиций. При этом необходимо учитывать следующие факторы. Во-первых, разработка сервиса может занять некоторое время. Во-вторых, реализованный сервис может быть очень специфичным и нужным только в этой композиции. В-третьих, время выполнения некоторых алгоритмов может быть небольшим, и реализация их в виде сервисов приведет к значительным накладным издержкам, связанным с инициализацией сервиса, передачей данных и т. д. Рассмотрим существующие реализации применения языков программирования для композиций сервисов.

В статье [21] для задания workflow используется язык программирования Python. Разработано API, с помощью которого внутри Python-программы пользователь самостоятельно формирует DAG, зависимости DAG задаются при создании вершин, реализован асинхронный вызов сервисов. В указанном подходе не реализованы планирование выполнения DAG и адаптация выполнения DAG к изменениям распределенной среды.

Существует гибридный подход, когда для задания DAG используется язык разметки (преимущественно XML). Для управления выполнением DAG применяется специализированное API на одном из языков программирования. Например, в работе [22] управление выполнением DAG совершается посредством вызова обработчиков на языке JavaScript, указываемых в качестве атрибутов элементов XML-документа, описывающего композицию сервисов. Гибридный подход не обрабатывает промежуточные данные и не адаптируется к изменениям вычислительной среды.

В статье [23] описан BPELscript, задающий workflow в виде программы, с похожим на JavaScript синтаксисом. Достоинством BPELscript является возможность его трансляции в WS-BPEL и обратно, что позволяет использовать существующие методы выполнения и планирования для DAG. Возможна частичная обработка данных стандартными средствами языка, но при трансляции в WS-BPEL она не может быть применена. При интерпретации BPELscript нет возможности применить существующие методы выполнения и планирования для DAG.

В подходе [24] workflow задается с помощью языка Python. В процессе интерпретации программы строится DAG. Для этого в языке предусмотрены специальные методы, создающие вершины DAG (вызовы сервисов), зависимости по данным задаются путем указания соответствия входных данных выходным. Полученный workflow может выполняться с помощью Apache Storm, MPI и последовательно на обычном компьютере. Такой подход не позволяет изменять DAG в зависимости от промежуточных данных, производить их обработку, выполнять планирование на изменяющейся среде.

Подход к созданию workflow для методов геообработки [25, 26] использует сервисы стандарта WPS. При этом workflow задается с помощью графического редактора и хранится в XML. Для выполнения workflow транслируется в программу на языке Python. Такой подход не учитывает изменения состояния среды и не проводит планирование и выполнение workflow. Применение языков программирования для задания workflow позволяет производить обработку промежуточных данных с помощью средств языка и его библиотек, использовать промежуточные данные в управляющих конструкциях языка, что значительно упрощает использование сервисов. Существующие программные решения, использующие языки программирования для формирования композиций сервисов, не производят автоматическое планирование вызовов сервисов в условиях распределенной гетерогенной среды, процесс выполнения композиций сервисов неустойчив к изменениям, происходящим в среде.

2. Модель выполнения композиций сервисов обработки пространственных данных

В соответствии со стандартом WPS-обработки пространственных данных на момент выполнения сервис может находиться в любой точке сети Интернет. Предсказать изменения состояния среды невозможно, так как в любой момент времени могут изме-

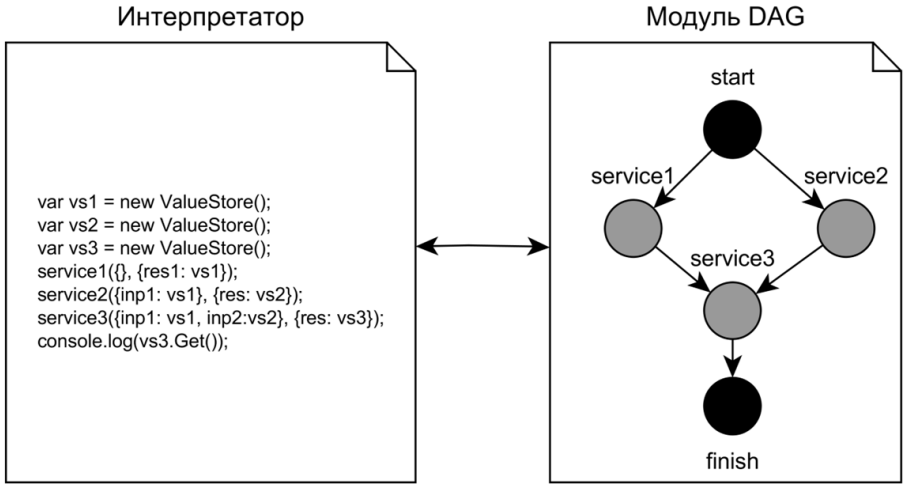

Рис. 1. Взаимодействие интерпретатора и модуля DAG

ниться количество вычислительных узлов, их загруженность, длительность выполнения задания и т. д.

Каждый сервис может выполняться на нескольких вычислительных узлах. Предполагается, что имеется ограниченное количество вычислительных узлов и на одном вычислительном узле может выполняться только один вызов сервиса в определенный момент времени. Обработка пространственных данных часто может быть распараллелена по пространственной сетке.

2.1. Основная идея

Основная идея заключается в развитии подхода, в котором композиция сервисов задается в виде программы (сценария) на языке программирования (в работе используется JavaScript), в процессе выполнения сценария формируется DAG с использованием оптимизирующих методов. Для упрощения в тексте статьи модуль выполнения программы будет называться интерпретатором, а модуль выполнения DAG — модулем DAG. Оба модуля работают одновременно и обмениваются данными. Интерпретатор в процессе выполнения сценария передает модулю DAG новые вершины с зависимостями и обработанные в сценарии данные, в том числе промежуточные. Модуль DAG передает интерпретатору промежуточные данные при необходимости их обработки или использования в управляющих конструкциях (рис. 1).

Обработка пространственных данных может быть распараллелена по пространственной сетке. Разделение и объединение пространственных данных по сетке выполняют методы библиотеки GDAL/OGR. Соответственно, если сервис обработки пространственных данных может работать на нескольких вычислительных узлах, то для ускорения его выполнения можно автоматически распределить его копии по свободным вычислительным узлам. Специфика объединения и разделения данных может быть описана в спецификации сервиса. В рамках метода модуль DAG может автоматически разделить выполнение сервиса обработки пространственных данных на несколько вершин DAG, выполняющихся на разных вычислительных узлах.

Рассмотрим подробнее взаимодействие двух модулей.

2.2. Формирование DAG

Формирование вершин DAG проводится способом, изложенным в работе [11], т. е. для добавления вершины вызывается одна из заранее определенных функций, соответствующая зарегистрированному в каталоге сервису (функция-обертка). При вызове функций-оберток происходит добавление вершины в граф композиции DAG с указанием сервиса, его характеристик, передаваемых и получаемых данных. Функция-обертка одного и того же сервиса может вызываться произвольное количество раз, в каждом случае это будет отдельная вершина DAG со своими зависимостями от других вершин.

Формирование ребер DAG, т. е. зависимостей между вершинами по данным, производится при регистрации узла DAG. Вершина может использовать данные, полученные из следующих источников:

- другие вершины;
- обработка данных в коде сценария;
- начальные данные, переданные пользователем.

Каждая функция-обертка имеет описание входных и выходных параметров, которое можно автоматически получить в соответствии со стандартом WPS. При вызове функции-обертки достаточно сформировать зависимости входных параметров. Зависимости выходных параметров автоматически формируются при добавлении следующих вершин.

Формирование зависимостей на основе анализа текста программы и выявления потоков данных является нетривиальной задачей, так как результаты выполнения сервисов могут помещаться в переменные, значения переменных затем могут переопределяться и т. д. В случае, когда источником данных является другая вершина, сложность заключается в идентификации соответствия данных входных параметров регистрируемой вершины и выходных параметров других вершин и в отсутствии их модификации в коде сценария. Поэтому предлагается при вызове функции-обертки в качестве параметров передавать объекты специального класса ValueStore (рис. 2), выступающие в роли контейнеров для данных и позволяющие однозначно идентифицировать передаваемые и получаемые данные.

При генерации функций-оберток автоматически формируется интерфейс, требующий использования контейнеров для всех параметров. Контейнер данных может использоваться в качестве входных параметров для нескольких вершин. Он может использоваться только один раз в качестве как выходной параметр функции-обертки, либо его значение задается в коде сценария, т. е. данные в него могут быть записаны единственный раз. Каждый контейнер имеет свой идентификатор, генерируемый автоматически. Запись данных осуществляется методом Set() в коде сценария или мо-

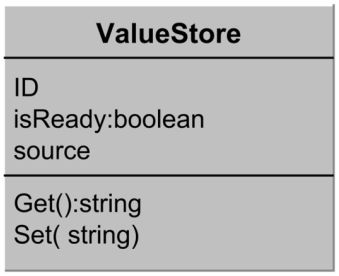

Рис. 2. Класс ValueStore

дулем DAG при завершении выполнения сервиса. Получение данных из контейнера производится с помощью метода Get(). Свойство isReady возвращает TRUE, если данные помещены в контейнер. Если контейнер получил или должен получить данные из другой вершины DAG, то свойство source содержит ссылку на эту вершину.

Рассмотрим определение зависимостей вершин на примере:

```
var vs1 = new ValueStore();
var vs2 = new ValueStore();
var vs3 = new ValueStore();
service1({}, {res1: vs1});
service2({inp1: vs1}, {res: vs2});
service3({inp1: vs1, inp2:vs2}, {res: vs3});
console.log(vs3.Get());
```

При вызове функции-обертки service1 в качестве выходного параметра указан объект vs1. Объект vs1 используется в качестве входного параметра при вызове функций-оберток service2 и service3, объект vs2 — в качестве входного параметра при вызове функции-обертки service3. В результате получается граф DAG, представленный на рис. 3.

Определение зависимости между вершинами по данным происходит следующим образом: при выполнении функции-обертки сервиса происходит анализ ее входных параметров. Для каждого контейнера происходят запоминание его идентификатора и проверка, использует ли какая-либо вершина этот контейнер в качестве выходного параметра. В случае если проверка находит такую вершину, в граф добавляется ребро, т. е. обнаружена зависимость между вершинами по данным. Алгоритм формирования зависимостей задания приводится на рис. 4.

Получение данных внутри сценария для последующей обработки средствами выбранного языка программирования производится с помощью метода Get класса ValueStore. Если на момент вызова метода Get данные еще не готовы, то метод блокирует выполнение сценария. Выполнение сценария композиции продолжится только после задания данных в контейнере, так как возможна ситуация, что в зависимости от результатов выполнения какого-либо сервиса будет происходить выбор определенной ветви сценария. Задание входных данных сервисов выполняется с помощью метода Set() (не производит блокирование).

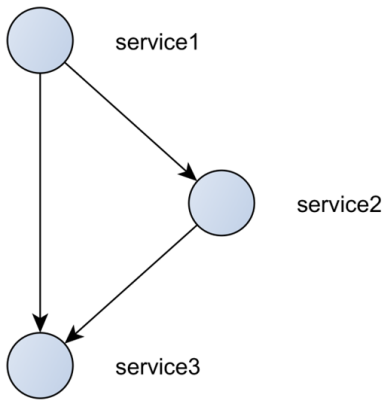

Рис. 3. Граф, созданный по сценарию

```
paramlist — массив параметров добавляемого узла
numParameters — количество параметров добавляемого узла
ValueStorelist — массив ValueStore
numValueStore — количество элементов массива ValueStore
elist — массив зависимостей
elist = []
//цикл по входным параметрам узла
for I <- 1, numParameters do
     in_param <- paramlist[i]
     //если значение параметра задано
     if in_param.ValueStore.isReady then
          //то переходим к следующему параметру
          continue
     //цикл по ValueStorelist
     for j <- 1, numValueStore do
          cur_VS <- ValueStorelist[j]
          if in_param.ValueStore.ID = cur_VS.ID
               and cur_VS.source <> NULL then
               elist[elist.length+1] = cur_VS.source
```

Рис. 4. Алгоритм формирования зависимостей

Рассмотрим, как производится выполнение сценария интерпретатором и как он взаимодействует с модулем DAG. Последовательные вызовы функций-оберток в коде сценария приводят к регистрации вершин в DAG. Интерпретатору после регистрации вершины нет необходимости останавливать работу. Соответственно, блокирование выполнения сценария не требуется. Блокирование выполнения сценария производится в двух случаях: а) если в коде сценария происходит ветвление на основе результатов одного или нескольких сервисов; б) если производится обработка данных, полученных в результате работы сервиса. В обоих случаях необходимо получить данные с помощью метода Get(). Блокирование производится до тех пор, пока необходимые результаты сервисов не будут получены.

В циклах кода сценария могут присутствовать вызовы функций-оберток. Если внутри цикла отсутствует обработка результатов сервисов, то блокировка выполнения сценария не производится. Соответственно, с помощью цикла можно зарегистрировать множество вершин DAG, которые могут выполнять сценарий параллельно при наличии соответствующих вычислительных узлов. Если же внутри цикла осуществляется работа с результатами сервисов, то итерации цикла будут останавливаться на периоды ожидания результатов выполнения сервисов.

2.3. Выполнение модуля DAG

Модуль DAG производит планирование, непосредственный вызов и получение результатов выполнения сервисов. Для нахождения плана, приближенного к оптимальному по времени выполнения, используется модификация спискового эвристического алгоритма составления расписания HEFT (Heterogeneous Earliest Finish Time), реализующего метод поиска в глубину. Особенностями алгоритма, служащими для ускорения

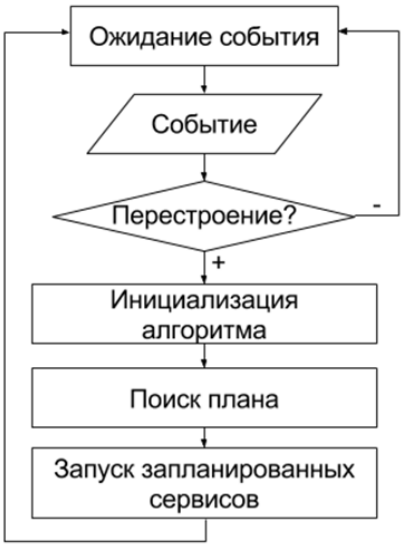

Рис. 5. Алгоритм работы модуля DAG

нахождения приемлемого плана, являются сортировка вершин DAG, вычислительных узлов и отсечение неперспективных ветвей графа поиска решения, использование информации о текущем состоянии вычислительной среды для более быстрого и точного нахождения плана.

При выполнении DAG необходимо учитывать изменение состояния вычислительной среды и добавление новых вершин в DAG. Все эти изменения требуют перестроения плана. Имеет смысл проводить перестроение только при определенных событиях:

- включение или отключение вычислительного узла, поддерживающего хотя бы один сервис DAG;
- добавление новой вершины с определенными зависимостями в DAG;
- отличие фактического времени выполнения одной из вершин DAG от ожидаемого;
- завершение выполнения вершины DAG с ошибкой.

При наступлении события необходимо сформировать новый план, минимизирующий время выполнения текущего состояния DAG. Разработанный в этих целях алгоритм представлен в виде блок-схемы на рис. 5.

При регистрации события происходит определение необходимости перестроения плана. Например, в случае добавления вершины в DAG и назначения его на менее загруженный вычислительный узел общее время плана не меняется, перестроение не требуется.

3. Апробация

Для апробации решения переборных задач в сервис-ориентированной среде рассмотрена следующая задача: существует набор прецедентов, определяющих местоположение представителей следующих типов растительности — степь, сосновый лес, болото, пихтовый лес, березовый лес, лиственничный лес (всего 6 типов). Местоположение представителей задается точками. Этот набор собран специалистами предметной области в полевых экспедициях. Также имеются наборы растровых файлов, в данном случае SRTM (Shuttle Radar Topography Mission, набор цифровых моделей местности),

NDVI (Normalized Difference Vegetation Index, простой количественный показатель количества фотосинтетически активной биомассы) и данные по высоте, склону и экспозиции местности. Требуется найти такие комбинации растровых файлов, которые обеспечивали бы наибольшую точность классификации типа растительности для каждой пары типов. Для классификации типов растительности применяется метод опорных векторов SVM (Support Vector Machine). На основе этого метода реализован WPS-сервис SVM_Learn, развернутый на четырех вычислительных узлах в пределах локальной облачной инфраструктуры. SVM_Learn принимает на вход набор растров, SHP-файл с прецедентами, название класса и значения класса для разделения прецедентов. Результатами работы сервиса являются файл классификатора, а также значение точности найденной модели классификации.

Для решения задачи разработан сценарий на языке JavaScript, который перебирает возможные комбинации растровых файлов. Для каждой комбинации прецедентов должна выбираться модель с наибольшей точностью. Результатом работы сценария будет массив, в котором содержатся модели-победители для каждой комбинации типов растительности.

```
function SVM_composite_full_sort(input, mapping){
  var classes =~[{id: 1, name: 'Step'}, .. ]
  for (var ai = 0; ai < classes.length; ai++) {
    for (var bi = ai; bi < classes.length; bi++) {
      /* ... */
      SVM_Learn_from_shp(localCommonParameters, (2)
        {Model: localModelValueStore,
        Precision: valueStores[vsl].precision});
      }
    }

    for (var k = 0; k < combinationIds.length; k++) { (3)
      var maxPrecision = false;
      for (var v = 0; v < valueStores.length; v++) {
        if (valueStores[v].precision.get() > maxPrecision) {
          maxPrecision = valueStores[v];
        }
      }

      print('Best precision is ' + ...);
  }
}
```

На рис. 6 приведено фактическое расписание выполняющихся заданий. Этот рисунок является скриншотом фрагмента веб-интерфейса запуска сценариев. Всего сценарий выполнялся в течение 158 с.

Решение данной задачи с помощью задания и выполнения сценария на языке программирования позволило автоматически распараллелить вызов сервисов по доступным вычислительным узлам в результате выполнения цикла по возможным комбинациям параметров в теле сценария без каких-либо дополнительных команд со стороны пользователя (автора сценария).

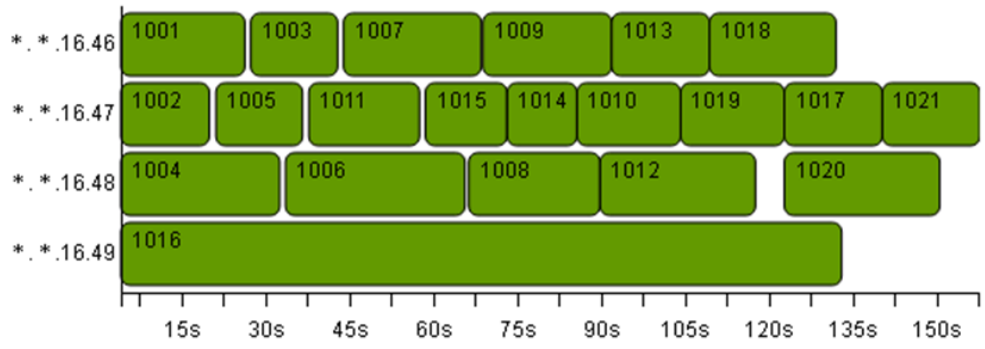

Рис. 6. Расписание без отключения узлов

Для апробации обработки промежуточных результатов сервисов внутри композиции решена следующая задача: существует набор точечных объектов, соответствующих расположению школ. Необходимо произвести расчет временно́й доступности каждого объекта на основании данных о дорожной сети и естественных преградах. Результатом решения задачи должен быть набор растровых файлов, в каждой ячейке которых содержится время, необходимое для достижения каждого объекта с помощью дорожной сети и с учетом естественных препятствий.

Решением задачи является композиция сервисов, которая осуществляет разбиение начального набора точечных объектов инфраструктуры и расчет временно́й доступности для каждого объекта. Разбиение объектов должно осуществляться внутри сценария композиции с помощью стандартных средств, предоставляемых языком программирования JavaScript.

В данном сценарии используется ряд сервисов. Сервис `shp_to_geojson_converter` осуществляет конвертацию векторного файла в формате SHP в текстовый формат GeoJSON. Сервис `geojson_to_shp_converter` осуществляет обратную операцию, конвертируя данные в текстовом формате GeoJSON в файл в формате SHP. Сервис `bufferize_vector`, принимая на вход набор точечных объектов в векторном формате SHP, преобразует их в полигональные объекты, представляющие собой буферные зоны, построенные вокруг входных точечных объектов с определенным радиусом в метрах. Сервис `road_analysis` осуществляет расчет времени, которое необходимо, чтобы добраться до каждого объекта образовательной инфраструктуры.

Композиция имеет следующие входные параметры:

- файл с набором точечных объектов в векторном формате SHP. Каждый точечный объект — объект образовательной инфраструктуры;
- файл с набором границ объектов, представляющих естественные преграды для расчета (водные объекты) в векторном формате SHP;
- файл, содержащий структуру дорожной сети в векторном формате SHP;
- область расчета.

В приведенном ниже сокращенном коде сценария сначала происходит вызов сервиса `shp_to_geojson_converter`, производится преобразование входного объекта с точечными объектами инфраструктуры из векторного формата в текстовый формат JSON. Сервис возвращает строку текста, которая обрабатывается JavaScript-методом JSON.parse() в фрагменте кода (сноска 1 в приведенном ниже коде сценария), результат выполнения сервиса становится доступным внутри сценария в виде стандартного объекта

JavaScript. Далее происходит выполнение последовательности оставшихся сервисов для каждого точечного объекта, полученного после обработки результата работы сервиса `shp_to_geojson_converter` внутри сценария композиции.

```
function School_availability(input, mapping){
  shp_to_geojson_converter({Sourse: input.schools},
    { ResultJSON: schools});
  if (schools.get().length > 0) {
    var parsedData = JSON.parse(schools.get());(1)
    for (var i = 0; i < limit; i++) {
      geojson_to_shp_converter({Sourse: parsedData[i]},
        {ResultFile: shp[i]});
      bufferize_vector({Sourse: shp[i], Buffer: 50},
        {Result: buffered[i]});
      road_analysis({Cities: buffered[i], Roads: input.roads},
        {Result: resultValueStores[i]});
    }
  }
}
```

Результатом выполнения композиции сервисов является набор растров, каждый из которых содержит данные о времени, необходимом для того, чтобы добраться до каждого из объектов образовательной инфраструктуры. На рис. 7 показана визуализация результата выполнения сценария для отдельной школы, расположенной в центре отображаемого растрового файла и отмеченной специальным маркером. На легенде спра-

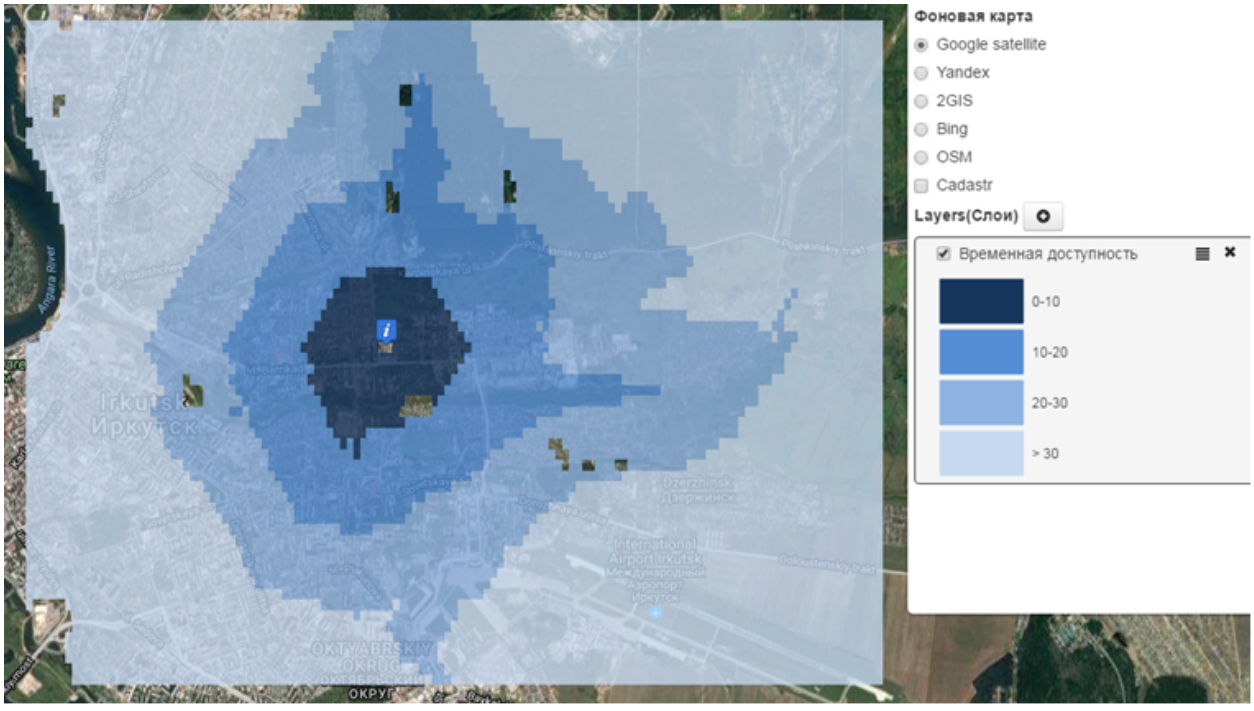

Рис. 7. Визуализация временно́й доступности отдельного объекта образовательной инфраструктуры

ва можно наблюдать оценки времени, необходимого для достижения образовательного объекта. Интервалы времени окрашиваются в соответствующий цвет.

Решение поставленной задачи с помощью задания и выполнения сценария на языке программирования позволило избежать создания дополнительного сервиса, занимающегося обработкой результатов выполнения сервиса `shp_to_geojson_converter`, так как результаты работы сервисов возможно обрабатывать стандартными средствами языка программирования.

Рассмотренный пример демонстрирует преимущество разработанной системы, заключающееся в возможности работы с результатами выполнения сервисов как с обычными данными внутри тела сценария (например, ссылка 1 в приведенном выше коде сценария). Такой подход позволяет использовать существующие программные библиотеки для выбранного языка написания сценариев.

Заключение

Предложен метод задания композиций сервисов обработки пространственных данных с помощью языка программирования JavaScript, позволяющий производить обработку промежуточных данных с помощью средств языка и его библиотек, использовать промежуточные данные в управляющих конструкциях языка. Метод выполнения композиций сервисов обработки пространственных данных позволяет применить существующие методы планирования выполнения композиций и параллельной обработки пространственных данных в гетерогенной динамической вычислительной среде. Метод упрощает подключение новых сервисов и передачу пространственных данных между сервисами.

Благодарности. Работа выполнена при финансовой поддержке РФФИ (гранты № 16-07-00411_а, 16-07-00554_а, 17-57-44006-монг, 17-47-380007-р-а, 18-07-00758-а), Программы Президиума РАН № 27, Интеграционных программ СО РАН, ИНЦ СО РАН и ЦКП ИИВС ИРНОК.

Список литературы / References

[1] **Bih, J.** Service oriented architecture (SOA) a new paradigm to implement dynamic e-business solutions // Ubiquity. 2006. Vol. 4. P. 1–17.

[2] **Hoffmann, J., Weber, I.** Web service composition // Encyclopedia of Social Network Analysis and Mining. N.Y.: Springer-Verlag, 2014. P. 118–128.

[3] **Deelman, E., Vahi, K., Juve, G.** Pegasus, a workflow management system for science automation // Future Generation Comput. Syst. 2015. Vol. 46. P. 17–35.

[4] **Ludscher, B., Altintas, C., Berkley, D. et al.** Scientific workflow management and the Kepler system // Concurrency and Computation: Practice & Experience. Special Issue: Workflow in Grid Systems. 2006. Vol. 18(10). P. 1039–1065.

[5] **Wilde, M., Hategan, M., Wozniak, J.M.** A language for distributed parallel scripting // Parallel Comput. 2011. Vol. 37(9). P. 633–652.

[6] **Berthold, M.R., Cebron, N., Dill, F.** The konstanz information miner // ACM SIGKDD Explorations News-letter. 2009. Vol. 11(1). P. 26–31.

[7] **Wolstencroft, K., Haines, R., Fellows, D.** The Taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud // Nucleic Acids Res. 2013. Vol. 41(W1). P. 557–561.

[8] **Blankenberg, D., Kuster, G.V., Coraor, N.** Galaxy: a web-based genome analysis tool for experimentalists. N.Y.: Wiley, 2010. P. 191–207.

[9] **Simmhan, Y., Barga, R., Ingen, C.** Building the trident scientific workflow workbench for data management in the cloud // Intern. Conf. on Advanced Eng. Comput. and Appl. in Sci. (ADVCOMP), Oct. 11–16, 2009. Sliema, Malta.

[10] **Churches, D., Gombas, G., Harrison, A.** Programming scientific and distributed workflow with Triana services: Research articles // Concurrency and Computation: Practice & Experience. 2006. Vol. 18(10). P. 1021–1037.

[11] **Smirnov, S., Sukhoroslov, O., Volkov, S.** Integration and combined use of distributed computing resources with everest // Procedia Comput. Sci. 2016. Vol. 101. P. 359–368.

[12] **Boukhanovsky, A.V., Vasilev, V.N., Vinogradov, V.N. et al.** CLAVIRE: Perspective technology for second generation cloud computing // Sci. and Technical J. "Priborostroenie". 2011. Vol. 54. P. 7–14.

[13] **Chen, N.C., Di, L.P., Yu, G.N., Gong, J.Y.** Geo-processing workflow driven wildfire hot pixel detection under sensor web environment // Computers & Geosciences. 2010. Vol. 36, No. 3. P. 362–372.

[14] **Xie, G., Li, R., Xiao, X., Chen, Y.** High-performance DAG task scheduling algorithm for heterogeneous networked embedded systems // Proc. of IEEE 28th Intern. Conf. Advanced Inform. Networking and Applications. Canada: Victoria, 2014. P. 1011–1016.

[15] **Kwok, Y.-K., Ahmad, I.** Static scheduling algorithms for allocating directed task graphs to multiprocessors // ACM Computing Surveys. 1999. Vol. 31, No. 4. P. 406–471.

[16] **Topcuoglu, H., Hariri, S., Wu, M.** Performance-effective and low-complexity task scheduling for heterogeneous computing // Parallel Distributed Syst. 2002. Vol. 13, No. 3. P. 260–274.

[17] **Munir, E., Mohsin, S., Hussain, A. et al.** SDBATS: A novel algorithm for task scheduling in heterogeneous computing systems // Proc. of Parallel and Distributed Proc. Symp. Works. USA: Boston, 2013. P. 43–53.

[18] **Arabnejad, H., Barbosa, J.G.** List scheduling algorithm for heterogeneous systems by an optimistic cost table // IEEE Trans. on Parallel and Distributed Syst. 2014. Vol. 25, No. 3. P. 682–694.

[19] **Wang, G., Guo, H., Wang, Y.** A novel heterogeneous scheduling algorithm with improved task priority // Proc. of IEEE 17th Intern. Conf. on High Perform. Comput. and Communications. Brazil: Rio de Janeiro, 2015. P. 1826–1831.

[20] **Gupta, S., Kumar, V., Agarwal, G.** Task scheduling in multiprocessor system using genetic algorithm // Proc. of Second Intern. Conf. on Machine Learning and Computing. 2010. P. 267–271.

[21] **Sukhoroslov, O., Volkov, S., Afanasiev, A.** Web-Based Platform for Publication and Distributed Execution of Computing Applications // 14th Intern. Symp. on Parallel and Distributed Comput., Limassol. 2015. P. 175–184.

[22] **Сидоров И.А., Опарин Г.А., Феоктистов А.Г.** Технология организации распределенных вычислений в инструментальном комплексе Discomp // Современные технологии. Системный анализ. Моделирование. 2009. № 2. С. 175–179.

Sidorov, I.A., Oparin, G.A., Feoktistov, A.G. Technology of organization of distributed computations using the instrumental complex Discomp // Sovremennye Tekhnologii. Sistemnyy Analiz. Modelirovanie. 2009. No. 2. P. 175—179. (In Russ.)

[23] **Bischof, M., Kopp, O., Lessen, T., Leymann, F.** BPELscript: A simplified script syntax for WS-BPEL 2.0 // 35th Euromicro Conf. on Software Eng. and Advanced Appl., Greece, Patras. 2009. P. 39–46.

[24] **Filguiera, R., Klampanos, I., Krause, A. et al.** Dispel4py: A Python framework for data-intensive scientific computing // Intern. Works. on Data Intensive Scalable Comput. Syst., New Orleans, LA. 2014. P. 9–16.

[25] **Bu, X., Yue, P., Wang, L., Zhang, M.** A scripting approach for integrating software packages and geoprocessing services into scientific workflows // Fourth Intern. Conf. on Agro-Geoinform., Istanbul. 2015. P. 15–18.

[26] **Zhang, M., Yue, P.** GeoJModelBuilder: A java implementation of model-driven approach for geoprocessing workflows // Second Intern. Conf, on Agro-Geoinform., Fairfax, VA. 2013. P. 393–397.

Поступила в редакцию 6 июня 2018 г.,
с доработки — 29 октября 2018 г.

Executing JavaScript compositions of WPS-services in the distributed heterogeneous environment

Bychkov, Igor V.[1], Rugnikov, Gennady M.[2], Fedorov, Roman K.[2], Shumilov, Aleksandr S.[1,*]

[1]Matrosov Institute for System Dynamics and Control Theory of SB RAS, Irkutsk, 664033, Russia
[2]Irkutsk Scientific Center of SB RAS, Irkutsk, 664033, Russia

[*]Corresponding author: Shumilov, Aleksandr S., e-mail: shumilov@icc.ru

The service-oriented approach (SOA) has recently gained wide implementation in the field of distributed computations. SOA allows publishing of various software packages, algorithms, data sources in a form of atomic services.

Distributed services are actively used for the processing of large volumes of spatial data using the open data formats and service interfaces standards. Because of the constant increase in number of developed services theirs compositions became widely used to solve complex interdisciplinary problems. The service composition is the set of services with defined interaction that is intended to solve specific complex task.

This work considers existing service composition and execution method and proposes an implementation of the distributed service compositions using the JavaScript programming language. The proposed approach differs from other ones which also assume usage of programming languages in order to create compositions. The current approach allows automatic scheduling of service calls in order to minimize the overall composition execution time, the actual composition execution process is tolerant to changes in computational environment and the intermediate results inside of compositions can be processed using the regular programming language tools.

The proposed approach of creating service compositions using the JavaScript programming language allows processing of intermediate data using the standard tools of the chosen programming language and compatible libraries, as well as using service results in control structures. The spatial service composition method allows applying existing scheduling algorithms and parallel spatial data processing techniques in heterogeneous computational environment. The approach is implemented as a multi-user internet-system.

Keywords: service compositions, distributed heterogeneous systems, services, geoportal.

Cite: Bychkov, I.V., Rugnikov, G.M., Fedorov, R.K., Shumilov, A.S. Executing JavaScript compositions of WPS-services in the distributed heterogeneous environment // Computational Technologies. 2019. Vol. 24, No. 3. P. 44–58. (In Russ.)
DOI: 10.25743/ICT.2019.24.3.004.

Acknowledgements. This research was supported by RFBR (grants No. 16-07-00411_a, 16-07-00554_a, 17-57-44006-монг, 17-47-380007-р-а, 18-07-00758-а), RAS Presidium program No. 27, Integration programs of SB RAS, ISC SB RAS and Shared Equipment Center of Integrated information and computing network of Irkutsk Research and Educational Complex.

Received June 6, 2018
Received in revised form October 29, 2018