ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ УЧРЕЖДЕНИЕ НАУКИ
ИНСТИТУТ ДИНАМИКИ СИСТЕМ И ТЕОРИИ УПРАВЛЕНИЯ
ИМЕНИ В.М. МАТРОСОВА
СИБИРСКОГО ОТДЕЛЕНИЯ РОССИЙСКОЙ АКАДЕМИИ НАУК

На правах рукописи

Шумилов Александр Сергеевич

СРЕДА РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ НА ОСНОВЕ WPS-СЕРВИСОВ

Специальность 05.13.11 – Математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Диссертация на соискание учёной степени
кандидата технических наук

Научный руководитель –
к.т.н. А.Е.Хмельнов

Иркутск – 2018

Оглавление

ОПРЕДЕЛЕНИЯ

Архитектура – формальное описание целей, функций, внешних видимых свойств и интерфейсов системы. Архитектура также включает описание внутренних компонентов системы и их отношений, наряду с принципами, управляющими ее дизайном, функционированием и возможной последующей эволюцией.

Веб-сервис – сервис, доступ к которому осуществляется посредством протокола HTTP.

Геопортал – это программно-технологическое обеспечение для работы с пространственными данными. Его основная задача – обеспечение пользователя средствами и сервисами хранения и каталогизации, публикации и загрузки пространственных (географических) данных, поиска и фильтрации по метаданным, интерактивной веб-визуализации, прямого доступа к пространственным данным на основе картографических веб-сервисов.

Инкапсуляция – упаковка данных и функций в единый компонент.

Композиция сервисов – объединение существующих сервисов, определяющее соотношения и взаимодействия между сервисами и нацеленное на решение конкретной задачи.

Концептуальная модель – модель предметной области, состоящая из перечня взаимосвязанных понятий, используемых для описания этой области, вместе со свойствами и характеристиками, классификацией этих понятий по типам, ситуациям, признакам в данной области и законам протекания процессов в ней.

Оркестрация сервисов – описывает то, как сервисы должны взаимодействовать между собой, используя для этого обмен сообщениями, включая бизнес-логику и

последовательность действий. Оркестровка подчинена какому-то одному из участников бизнес-процесса.

Организация выполнения композиций сервисов – разработка и предоставление специальных программных средств, позволяющих определять и осуществлять взаимодействие между сервисами в виде их композиций для решения сложных задач, а также сама технология применения данных программных средств.

Программная система – система, состоящая из программного и аппаратного обеспечения и данных, главная ценность которой создается посредством исполнения программного обеспечения.

Парадигма облачных вычислений – объединение в единое информационно-вычислительное пространство произвольного множества гетерогенных и пространственно-распределенных вычислительных узлов, находящихся в различных административных доменах, со своей локальной политикой безопасности.

Распределённая гетерогенная среда – вычислительная среда, характеризующаяся как различающимися программными и аппаратными характеристиками участвующих узлов, так и их географическим расположением.

Распределенная обработка данных – обработка различных процессов одной программы (задания) на различных узлах распределенной системы обработки и организация взаимодействия по сети ЭВМ.

Распределённая вычислительная сеть – совокупность вычислительных узлов, объединенных коммуникационной сетью.

Сервис – механизм, позволяющий получать доступ к одному или нескольким программным средствам, где доступ осуществляется посредством специального

интерфейса и в соответствии с ограничениями и правилами, определяемыми описанием сервиса.

Сервис (служба) – программный компонент, доступный для удаленного вызова посредством компьютерной сети и предоставляющий некоторые функциональные возможности запрашивающей стороне.

Сервис-ориентированная архитектура (Service-oriented architecture, SOA) – модульный подход к разработке программного обеспечения, основанный на использовании распределённых, слабо связанных заменяемых компонентов, оснащённых стандартизированными интерфейсами для взаимодействия по стандартизированным протоколам.

Хореография сервисов – глобальное описание для участвующих сервисов, определяющее набор правил взаимодействия и соглашений между сервисами. Основное отличие хореографии от оркестрации заключается в том, что хореография децентрализована, логика взаимодействия определена на самих узлах-участниках вычислений.

Экосистема – набор соединений между различными службами, функциональное назначение которых состоит в выполнении запросов клиента.

ВВЕДЕНИЕ

Актуальность темы

В последние годы активно развивается сервис-ориентированный подход реализации программных продуктов в виде сервисов и организации их распределенной обработки в вычислительных сетях. Сервис – механизм, позволяющий получать доступ к одному или нескольким программным средствам посредством специального интерфейса в соответствии с ограничениями и правилами, определяемыми описанием сервиса. Одной из разновидностей сервисов являются веб-сервисы, взаимодействие с которыми осуществляется с использованием протокола HTTP(S). Сервис-ориентированный подход обладает следующими преимуществами – сервисы кроссплатформенны, тестируемы, интерфейс доступа стандартизирован. Программные системы, реализованные с помощью данного подхода, имеют сервис-ориентированную архитектуру (service-oriented architecture, SOA). SOA обладает следующими преимуществами – сервисы кроссплатформенны, тестируемы, доступны по сети, интерфейс доступа стандартизирован [1]. Развитие SOA позволяет более тесно интегрировать разработчиков из разных предметных областей за счет упрощения применения программного обеспечения. В рамках SOA нет необходимости устанавливать, конфигурировать, обновлять программное обеспечение. Достаточно удаленно запустить программное обеспечение (сервис) через его интерфейс и получить результат. Вычислительные узлы, на которых работают сервисы, могут быть расположены в любой точке сети Интернет и иметь разные программные и аппаратные характеристики. На текущий момент известны десятки сервисов, решающих различные прикладные задачи.

С развитием и увеличением количества сервисов для решения сложных междисциплинарных проблем возникла необходимость создания и выполнения их композиций – объединений существующих сервисов с определенными между ними

соотношениями и взаимодействиями, нацеленными на решение конкретных задач [2]. Внутри композиции результаты работы сервисов могут использоваться в качестве входных данных для других сервисов. Сервисы, участвующие в композиции, могут находиться на разных вычислительных узлах.

Основным методом задания композиций сервисов является использование одного из существующих стандартов, как правило, основывающихся на языке разметки XML. Композиции могут задаваться в текстовом виде в соответствии с одним из стандартов, а также с помощью программных сред, использующих графические примитивы для построения взаимодействия сервисов. Одним из перспективных методов задания композиций является использование процедурных языков программирования, позволяющее использовать существующие библиотеки программ, производить штатными средствами языка обработку промежуточных данных, задавать сложные алгоритмы обработки данных, в том числе рекурсивные, и т.д. Значительный вклад в задачу построения композиций сервисов внесли Тельнов Ю.Ф.[3], Сухорослов О.В. [4], Фостер И. [5], Папазоглу М. [6], Паутассо К. [7], Тернер М. [8], Будген Д. [9], Пелц К. [10], Рипон С. [11].

Композиции сервисов выполняются в рамках распределенной и гетерогенной вычислительной среды, то есть вычислительные узлы отличаются своим географическим расположением и имеют разные программные и аппаратные характеристики. Выполнение некоторых сервисов, участвующих в композициях, может производиться одновременно. Сервис может выполняться на нескольких вычислительных узлах с разной производительностью. Зависимость сервисов по данным определяет упорядоченность запуска сервисов. Количество вычислительных узлов ограничено, поэтому требуется планирование выполнения сервисов с учетом различных критериев, например, уменьшение времени выполнения композиции, равномерное распределение нагрузки по вычислительным узлам и т.д. Данная задача

исторически проистекает из проблемы выполнения пакетов прикладных программ (ППП), что делает возможным применение уже существующих подходов и алгоритмов для решения задач. Необходимо отметить, что выполнение композиций сервисов в отличие от ППП проводится в постоянно изменяющейся среде, где появляются или становятся недоступными вычислительные узлы сервисов, изменяется вычислительная мощность узлов и скорость передачи данных. Заранее предусмотреть эти изменения невозможно из-за неконтролируемости узлов, поэтому активно развиваются методы, позволяющие в процессе выполнения корректировать план под текущее состояние среды. Задача планирования выполнения сервисов является NP сложной, что требует использования соответствующих алгоритмов. Например, активно применяются эвристические и генетические алгоритмы, а также их комбинации. Ученые, повлиявшие на область выполнения композиций сервисов и пакетов прикладных программ в распределенных средах – Горбунов-Посадов М.М. [12,13], Опарин Г.А. [14], Бухановский А.В.[15,16], Насонов Д.А. [17,18], Топкугло Х. [19], Харири С. [20], Гупта С. [21], Таи Л. [22].

Несмотря на значимые научно-прикладные результаты по разработке методов и теоретических основ организации выполнения композиций сервисов и пакетов прикладных программ в распределенных средах, существующие реализации методов построения и выполнения композиций сервисов с помощью процедурных языков программирования не учитывают распределенность и гетерогенность вычислительной среды, что не позволяет использовать существующие наработки по оптимизации выполнения и автоматической адаптации к изменяющимся условиям вычислительной среды. Это определяет актуальность разработки новых методов задания композиций сервисов на процедурных языках программирования, а также реализации программной системы, выполняющей автоматическое планирование и распараллеливание выполнения композиций сервисов, расположенных на узлах распределенной гетерогенной вычислительной среды.

Цели и задачи исследования

Цель диссертационной работы состоит в разработке и реализации программной системы организации выполнения заданных на процедурных языках программирования композиций сервисов в распределённой гетерогенной среде.

Основные задачи исследования:

1. Анализ существующих подходов, методов и программных средств организации выполнения композиций распределенных сервисов обработки данных, а также особенностей гетерогенной динамической среды;
2. Построение концептуальной модели выполнения композиций сервисов в гетерогенной динамической среде;
3. Разработка методов задания и планирования выполнения процедурных композиций распределенных сервисов в гетерогенной динамической среде;
4. Реализация и апробация программного средства организации выполнения процедурных композиций сервисов.

Объект исследования

Теоретические и информационные аспекты задания и выполнения процедурных композиций сервисов, распределенная гетерогенная динамическая информационно-вычислительная среда.

Предмет исследования

Методы задания композиций сервисов; методы планирования и выполнения процедурных композиций сервисов в распределенной гетерогенной динамической вычислительной среде.

Методы исследования

В работе использовались методы информационного моделирования, теории графов, системного и объектно-ориентированного программирования, проектирования баз данных, построения распределенных комплексов проблемно-

ориентированных программ, веб-технологий, планирования выполнения в статических и динамических средах.

Научная новизна

1. Оригинальный метод задания процедурных композиций сервисов с помощью языка программирования JavaScript, позволяющий скрыть: алгоритм планирования и выполнения сценария, обработку изменения в вычислительной среде, а также передачу промежуточных данных между сервисами внутри интерпретатора сценариев.
2. Впервые реализована организация планирования выполнения процедурных композиций сервисов с учетом промежуточных результатов в гетерогенной динамической вычислительной среде.
3. Реализована не имеющая аналогов программная система организации выполнения процедурных композиций сервисов в распределенной гетерогенной среде в виде многопользовательской интернет-системы.

Положения, выносимые на защиту

1. Метод задания композиций распределенных сервисов в распределенной гетерогенной среде с помощью процедурного языка программирования JavaScript.
2. Метод организации планирования выполнения процедурных композиций сервисов в распределенной гетерогенной вычислительной среде.
3. Реализация программной системы задания и выполнения процедурных композиций сервисов в распределенной гетерогенной среде в виде многопользовательской интернет-системы.

Теоретическая и практическая значимость

Основные научные результаты по теме диссертации получены в рамках: Программы № I.33П фундаментальных исследований Президиума РАН, проект

0348-2015-0007, Программы фундаментальных исследований государственных академий на 2013-2020 годы, проект IV.38.2.3. «Новые методы, технологии и сервисы обработки пространственных и тематических данных, основанные на декларативных спецификациях и знаниях», Интеграционной программы ИНЦ СО РАН, проект 0341-2016-0001, проектов РФФИ 17-47-380007_р_а, 16-57-44034 монг_а, 16-07-00411-а, 16-07-00554-а, 16-37-00110 мол_а и также ведущей научной школы НШ-8081.2016.9.

Созданное программное обеспечение зарегистрировано в Федеральной службе по интеллектуальной собственности, патентам и товарным знакам (№ 2017617913, № 2016663524, № 2014610274)

Разработанная и реализованная в диссертационной работе программная система организации выполнения композиций сервисов в распределённой гетерогенной среде с помощью процедурных композиций сервисов позволяет для задания композиций не определять порядок планирования, распараллеливания выполнения сервисов, механизм передачи данных между сервисами, а также обработку изменений в распределённой вычислительной среде. Это снижает трудозатраты и сокращает сроки разработки программных систем, а также повышает эффективность и надёжность процессов распределённой обработки данных.

Практическая значимость результатов подтверждена актами о внедрении, а также их использованием в учебном процессе в ИГУ в рамках курсов «Разработка программного обеспечения».

Достоверность результатов проведённых исследований

Подтверждается обоснованным использованием методов информационного моделирования, теории графов, системного и объектно-ориентированного программирования, проектирования баз данных, построения распределенных комплексов проблемно-ориентированных программ, веб-технологий, планирования выполнения в статических и динамических средах, опубликованных в открытой

печати, публикацией полученных результатов и согласованностью с исследованиями других авторов, представленных в печатных изданиях, а также работоспособностью программной системы организации выполнения композиций сервисов в распределённой гетерогенной среде при решении тестовых и прикладных задач.

Соответствие специальности

В соответствии с паспортом специальности 05.13.11 – Математическое и программное обеспечение вычислительных, комплексов и компьютерных сетей, диссертация охватывает исследование моделей, методов и алгоритмов проектирования и анализа программных систем; программные инструменты организации взаимодействия программ и их систем; создание программных систем параллельной и распределённой обработки данных. Отражённые в диссертационной работе положения соответствуют пунктам 1, 3, 8 и 9 области исследований специальности 05.13.11

Апробация результатов работы

Основные результаты работы докладывались и обсуждались на научных конференциях: Международная конференция «Second Russia and Pacific Conference on Computer Technology and Applications (*RPC)»* (Владивосток, 2017 г.); Международная научно-практическая конференция «Использование современных информационных технологий в ботанических исследованиях» (г. Апатиты, 2017 г.); Всероссийская конференция «Обработка пространственных данных в задачах мониторинга природных и антропогенных процессов» (г. Новосибирск, 2017 г.); XVIII Всероссийская конференция молодых учёных по математическому моделированию и информационным технологиям (г. Новосибирск, 2017 г.); Международная конференция «Информационные технологии и математическое моделирование в науке, технике и образовании» (г. Бишкек, 2016 г.); Национальный Суперкомпьютерный Форум (НСКФ-2016), (г. Переславль-Залесский, 2016 г.); «Байкальская Всероссийская конференция и школа-семинар научной молодежи» (г.

Иркутск, 2016, 2017 гг.); III Российско-Монгольская конференция молодых ученых по математическому моделированию, вычислительно-информационным технологиям и управлению (г. Иркутск, 2016 г.); Международная конференция «Вычислительные и информационные технологии в науке, технике и образовании» (CITech-2015), (г. Алматы (Казахстан), 2015 г.); «Российско-монгольской конференции молодых ученых по математическому моделированию, вычислительно-информационным технологиям и управлению» (г. Иркутск- Ханх (Монголия), 2013, 2015, 2016 гг.); Всероссийская конференция «Обработка пространственных данных в задачах мониторинга природных и антропогенных процессов» (с. Усть-Сема, Республика Алтай, 2014 г.); «XXXVIII Дальневосточная Математическая Школа-Семинар имени академика Е.В. Золотова» (г. Владивосток, 2014 г.); Всероссийская конференция «Обработка пространственных данных и дистанционный мониторинг природной среды и масштабных антропогенных процессов» (г. Барнаул, 2013 г.); III Всероссийская конференция «Математическое моделирование и вычислительно-информационные технологии в междисциплинарных научных исследованиях», (г. Иркутск, 2013 г.); «Ляпуновские чтения» (г. Иркутск, 2014-2017 гг.)

Публикации и личный вклад автора

Результаты диссертационной работы опубликованы в 39 печатных работах, в том числе 7 статей в рецензируемых журналах, рекомендованных ВАК для опубликования результатов диссертаций, 1 коллективная монография, 26 публикаций в трудах международных и всероссийских конференций, 3 свидетельства о государственной регистрации программ для ЭВМ.

Все выносимые на защиту научные положения получены соискателем лично. В основных научных работах по теме диссертации, опубликованных в соавторстве, лично соискателем разработаны: в [23] – метод задания процедурных композиций сервисов в виде сценариев на языке программирования JavaScript; [23–25] – метод

организации планирования выполнения процедурных композиций сервисов в распределенной гетерогенной динамической вычислительной среде; [26–29] – реализация программной системы задания и выполнения процедурных композиций сервисов в гетерогенной динамической среде в виде интернет-системы.

Структура и объем диссертации

Диссертационная работа состоит из введения, четырех глав, заключения и списка литературы, включающего 62 наименования, и 1 приложения. Объем составляет 131 страницу основного текста, включая 41 рисунок, список сокращений и условных обозначений.

ГЛАВА 1. АНАЛИЗ ОРГАНИЗАЦИИ РАСПРЕДЕЛЁННЫХ ВЫЧИСЛЕНИЙ В ГЕТЕРОГЕННЫХ СРЕДАХ

В современном мире активное развитие вычислительных, коммуникационных технологий, увеличение пропускной способности сетей, распределённость сред, а также большие объемы и разноформатность данных и рост количества программных систем значительно изменили способы создания распределенных вычислительных систем (РВС) и использования информационно-вычислительных ресурсов и услуг.

1.1 Сервисы

В области распределённых вычислений, получил широкое распространение сервис-ориентированный подход, когда различные программы, алгоритмы, источники данных публикуются в виде независимых атомарных сервисов . Сервис – это механизм, позволяющий получать доступ к одному или нескольким программным средствам с использованием специального интерфейса в соответствии с ограничениями, правилами и описанием. Сервисы могут выполнять вычисления, осуществлять управление устройствами, предоставлять доступ к данным, запускать выполнение цепочек других сервисов. Разновидностью сервисов являются веб-сервисы, доступ к которым осуществляется через сеть Интернет с использованием протокола HTTP(S). Такой подход к рассмотрению и реализации программного продукта в виде веб-сервиса имеет очевидные преимущества – веб-сервисы кроссплатформенны, легко тестируются, доступны из любой точки сети Интернет, доступ к сервисам, как правило, стандартизирован [30].

Сервис-ориентированный метод организации рассмотрения и реализации программных продуктов обладает преимуществами – сервисы кроссплатформенны, тестируемы, доступны по сети, интерфейс доступа стандартизирован. Программные

системы, реализованные с помощью данного подхода, используют сервис-ориентированную архитектуру (service-oriented architecture, SOA).

Сервис-ориентированный подход является одним из направлений развития области распределённых вычислений. В 1960-х годах, по мере развития многопроцессорных систем и параллельных вычислений, а также совершенствования аппаратного обеспечения (мэинфреймы, суперкомпьютеры), появилась необходимость передачи большого количества данных между вычислительными узлами, что обусловило развитие коммуникационных технологий. Первоначально передача данных производилась в пределах локальных сетей организаций, но уже в 1970-1980-х годах отдельные взаимодействия между локальными сетями организаций перерастают в глобальную сеть Интернет, появляются распределенные вычислительные системы (РВС), архитектуры параллельных вычислений, интерфейсы передачи сообщений между участниками РВС. В 1990-х годах появляются GRID-технологии, впоследствии развившиеся в облачные вычисления. GRID – это система, координирующая распределенные ресурсы посредством стандартных, открытых, универсальных протоколов и интерфейсов для обеспечения нетривиального качества обслуживания[5,31]. Целью GRID-систем является централизованное предоставление ресурсов для решения различного рода вычислительных задач.

Облачные вычисления являются развитием GRID-технологий, характеризуются масштабируемостью и виртуализацией [32]. Масштабируемость позволяет развертывать необходимые для решения задач ресурсы за приемлемое время, а также перераспределять нагрузку на свободные ресурсы, а виртуализация позволяет абстрагировать и унифицировать предоставляемые программные и аппаратные ресурсы для конечных пользователей. Развитие сервис-ориентированного подхода, обусловленное необходимостью обеспечения доступа к

GRID и облачным системам, привело к созданию большого количества стандартов интерфейсов сервисов, в том числе ориентированных на работу в сети Интернет (веб-сервисов). Пакеты прикладных программ, обычно выполняемых в рамках одного вычислительного узла, стало возможным запускать на наборе узлов, разнесенных географически, принадлежащих разным организациям и соединенных сетью Интернет.

Большой вклад в разработку методов и теоретических основ организации распределенных вычислений и выполнения пакетов прикладных программ внесли: Foster Y., Lamport L., Lynch N., Hansen P.B., Горбунов-Посадов М.М., Самарский А.А., Матросов В.М., Бахманн П., Опарин Г.А. и др. Таким образом, отдельные вычислительные модули стало возможным размещать на распределенных вычислительных узлах в виде веб-сервисов и использовать их для решения сложных задач в рамках РВС.

Сервис-ориентированный подход обладает рядом преимуществ:

- сервисы могут быть повторно использованы, что уменьшает стоимость разработки новых систем в силу наличия уже готовых модулей.

- использование сервисов предполагает организацию обмена сообщений между ними, что повышает мобильность и взаимозаменяемость сервисов.

- на основе контроля и анализа процесса обмена сообщениями можно обнаруживать атаки, обеспечивать необходимый уровень безопасности при передаче сообщений, осуществлять преобразование сообщений, осуществлять балансировку нагрузки на вычислительные узлы, на которых развернуты сервисы.

В настоящее время сервис-ориентированный способ организации часто используется для построения сложных многофункциональных систем, таких как системы международной онлайн торговли, системы банковского взаимодействия, системы предоставления государственных сервисов. Примером отечественной

сервис-ориентированной системы является «Система межведомственного электронного взаимодействия» (СМЭВ) [33], которая упрощает и централизует процесс получения государственных услуг населением. При этом каждый участник СМЭВ предоставляет свой набор стандартизированных сервисов, которые используют заранее оговоренные структуры данных и способы передачи сообщений.

Как правило, большие сервис-ориентированные системы характеризуются тем, что их внутренние сервисы взаимодействуют на основе единого набора протоколов. OASIS (Organization for the Advancement of Structured Information Standards) является некоммерческой организацией, разрабатывающей и внедряющей протоколы для сервис-ориентированных сред (Unified Business Language, UBL), электронной торговли, документооборота и т.д. OASIS также разработал известные стандарты UDDI (Universal Description Discovery and Integration), WS-BPEL (Web-Services Business Process Execution Language), WSS (Web Service Security).

Чаще всего сервисы являются инкапсуляцией каких-либо вычислительных алгоритмов и различаются по длительности, сложности, требованиям к программным и аппаратным ресурсам и т.д. Сервисы также характеризуются тем, что пользователь не знает внутреннюю логику работы сервиса, ориентируется только на его описание и взаимодействует с сервисом только через его стандартизированный интерфейс. Сервисы, в независимости от входных параметров, на любой корректный или некорректный входной параметр должны возвращать результат работы или сообщение с указанием ошибки в рамках стандарта интерфейса.

Говоря о сервис-ориентированной архитектуре, необходимо рассмотреть несколько аспектов работы сервисов – форматы представления данных, способы передачи данных между сервисами, способы описания сервисов, а также способы отображения доступных сервисов.

Существует два основных формата представления данных в сервис-ориентированной архитектуре – XML (eXtensible Markup Language) и JSON (JavaScript Object Notation). XML характеризуется большей сложностью синтаксиса и размером собственных данных формата, нежели JSON, но предоставляет большие возможности для задания отображения данных с помощью стандарта XSLT, а также поддерживает пространства имен [34]. Оба формата позволяют производить проверку сообщений с помощью стандартов XML Schema и JSON Schema соответственно. Третьим вариантом передачи данных является передача сырых данных (текстовая информация, сгенерированное сервисом изображение).

Между сервисами передача данных осуществляется путем включения набора данных в тело сообщения или передачи URL (Uniform Resource Locator) адреса файла данных. Передача данных внутри сообщений, формат которых определяется стандартом интерфейса сервиса, обычно реализовывается для данных малой размерности (строковые, числовые данные, URL-адреса). Если же сервис в качестве результата возвращает данные большого объема, например, набор изображений высокого разрешения, то целесообразнее будет вернуть URL данных, не включая непосредственный массив данных в тело сообщения. Передача данных больших объемов посредством их URL-адресов позволяет уменьшить нагрузку на сеть (данные запрашиваются принимающей стороной по мере необходимости), отдельный URL-адрес файла позволяет загружать файл несколько раз, не совершая запросов к самому сервису.

Среди основных способов описания сервисов можно выделить стандарт WSDL (Web Service Description Language). Стандарт WSDL описывает, как получить доступ к сервису и какие операции он может выполнять. WSDL изначально использовался для документирования сервисов, использующих SOAP в качестве формата обмена данными [35].WSDL состоит из четырех основных элементов:

1) Типы данных, используемых в работе сервиса (структура данных задается с помощью стандарта XML Schema, с помощью которого в дальнейшем можно производить проверку передаваемых данных);
2) Список сообщений, используемых сервисов. Каждое сообщение может содержать несколько частей, по аналогии с параметрами вызываемых функций в языках программирования;
3) Порты, то есть непосредственные методы, предоставляемые сервисом;
4) Форматы сообщений и детали протоколов для каждого метода, предоставляемого сервисом.

Для проверки корректности передаваемых данных используется стандарт XML Schema. Последняя версия WSDL 2.0 делает возможным описание REST-сервисов в рамках WSDL.

Сложность и перегруженность стандарта WSDL послужила толчком для создания более простого стандарта WADL (Web Application Description Language), специализирующегося на описании ресурсов, предоставляемых каким-либо сервисами, а также на описании отношений, установленных между ними. В какой-то степени WADL играет для REST сервисов ту же роль, что WSDL играет для сервисов, работающих на основе SOAP. WADL менее распространен чем WSDL, тем более что последние версии стандарта WSDL также поддерживают описание REST сервисов.

Протоколом удаленного вызова программных методов является XML-RPC (XML Remote Procedure Call), который использует XML файлы для удаленного вызова сервисов и передачи данных. Основной идеей данного протокола является представление передаваемых структур данных в виде XML документов, содержащих как строковые и числовые параметры, так и массивы и другие структур данных. Основным преимуществом XML-RPC является его понятность и легкость в работе.

Развитием идеи инкапсуляции запросов к удаленным сервисам посредством использования XML стал стандарт SOAP (Simple Object-Access Protocol). SOAP предназначен для работы со стандартом WSDL, что делает его интеграцию и использование удобным и регламентированным. SOAP может использоваться как с HTTP(S), так и с использованием альтернативных способов передачи данных, например, через электронную почту. SOAP обеспечивает высокую степени защищенности сервисов, он может использоваться как с базовой HTTP(S) авторизацией, так и со стандартами WS-Security [36]. SOAP сообщение обычно состоит из четырех частей – конверта, определяющего начало и конец сообщения, заголовка, содержащего опциональные атрибуты, тела и информации о возможных ошибках. В то же время, большое количество возможностей, а также специфика построения самих SOAP документов делают стандарт менее удобным в работе, чем XML-RPC.

В отличие от стандартизированных способов задания интерфейсов сервисов, таких как XML-RPC или SOAP, в последнее время все чаще используется стиль построения архитектуры веб-сервисов REST (REpresentational State Transfer) [37]. Основными отличиями данного стиля являются:

- клиент-серверная архитектура (единый интерфейс между клиентом и сервером, у клиента нет необходимости обращаться к каким-либо другим сервисам или источникам данных),

- отсутствие состояния (между двумя клиентскими запросами контекст не сохраняется на сервере),

- единообразие интерфейса (идентификация ресурсов посредством уникальных URL адресов, манипуляция ресурсов через представление).

Часто манипуляция с ресурсами, предоставляемыми через REST интерфейс осуществляется с использованием определенных в стандарте HTTP команд GET, POST, DELETE и т.д., что упрощает понимание и использование такого рода сервисов. Дополнительно в процессе обмена сообщениями между клиентом и сервисов используются HTTP-коды, определяющие статус ответа.

Для организации работы с веб-сервисами, ориентированными на обработку пространственных данных, часто используется XML стандарт WPS (Web Processing Service), разработанный консорциумом OGC [38]. Стандарт WPS охватывает как аспекты обнаружения и описания сервиса, так и непосредственный вызов и контроль выполнения сервисов. В стандарте имеется три вида запросов: GetCapabilities, DescribeProcess, Execute. Запрос GetCapabilities возвращает описание WPS службы (программной системы, организующей и координирующей процесс выполнения различных зарегистрированных в ней сервисов) и список доступных для выполнения сервисов. На основе идентификаторов сервисов, представленных в ответе на запрос GetCapabilities, в ответ на запрос DescribeProcess возвращается описание конкретного сервиса, включающее в себя название, описание, идентификаторы и типы входных и выходных параметров сервиса. На основании информации о конкретном сервисе совершается запрос Execute, в ответ на который клиенту может прийти**:**

1) сырые данные, произведённые сервисом (текст, изображение, видео файл);
2) XML документ с результатами выполнения (данные встраиваются в документ в явном виде);
3) XML документ со ссылками на результаты работы сервиса (клиент получает набор URL-адресов, ведущих на сохраненные результаты работы сервиса, которые будут доступны неограниченное время после завершения работы удаленного сервиса);

4) XML документ со статусом выполнения сервиса – стандарт WPS разрешает сколько угодно долгое выполнение сервисов. В случае если сервис выполняется длительное время, в документе со статусом выполнения сервиса указывается процент выполнения сервиса, а также адрес XML файла, в который будут помещены результаты выполнения сервиса по мере завершения работы сервиса.

Обычно WPS сервис представляет собой атомарную функцию, выполняющую какие-либо вычисления. Задание взаимодействия между WPS сервисами позволяет организовывать повторяющиеся рабочие процессы. WPS сервис может самостоятельно вызвать набор других WPS сервисов или принимать на вход список сервисов, которые необходимо выполнить, в качестве параметров Execute запроса.

1.2 Обнаружение сервисов

При большом количестве сервисов, используемых для решения различных задач, возникает проблема обнаружения и повторного их использования. Существует два подхода к организации обнаружения сервисов – централизованный и децентрализованный [39].

Одним из известных централизованных методов реализации обнаружения сервисов является использование стандарта UDDI (Universal Description Discovery and Integration), который является XML стандартом и определяет формат предоставления информации о зарегистрированных сервисах. При использовании UDDI реестров сервисов пользователь должен самостоятельно производить поиск требуемого сервиса в списке зарегистрированных на основании какого-либо критерия. Изначально UDDI разрабатывался для организации глобального хранилища сервисов, в котором различные научные и коммерческие организации могли бы регистрировать и предоставлять свои сервисы и одновременно использовать сервисы, зарегистрированные другими участниками. К сожалению, данная идея не нашла широкой поддержки и крупные участники рынка постепенно

стали предлагать собственные форматы организации реестров сервисов, более приспособленных к их потребностям.

Среди децентрализованных методов обнаружения сервисов одним из популярных является стандарт WS-discovery [40], разработанный совместными усилиями компаний Microsoft, IBM и т.д. и стандартизированный консорциумом OASIS. Основным отличием WS-discovery от UDDI является то, что UDDI ориентирован на создание глобального, межкорпоративного и межгосударственного индекса сервисов, а WS-discovery нацелен на обнаружение сервисов в рамках локальной сети организации. WS-discovery предполагает рассылку сообщений всем участникам сети. Сообщения содержат в себе координаты сервиса и краткую информацию о его предназначении и параметрах. Децентрализованный метод обнаружения сервисов получил широкое распространение, например, в операционных системах семейства Windows.

1.3 Композиции сервисов

Основным методом задания композиций сервисов является текстовый вид, когда последовательность вызова сервисов, а также порядок их взаимодействий, задаются в виде текстового файла с использованием одного из стандартов, базирующихся на языке разметки XML. Так как редактирование больших XML файлов затруднительно и требует навыков работы с языками разметки, существует программное обеспечение, позволяющее задавать композиции сервисов с помощью графических примитивов, соответствующих вызовам сервисов, зависимостям между ними, настройке каких-либо промежуточных операций обработки данных. Впоследствии заданные графическим образом композиции преобразуются в текстовый формат (BPEL, XPDL, и т.д.) без участия пользователя.

Графический метод задания композиций ограничен набором заранее определенных примитивов и их возможностями, например, используя графические

примитивы достаточно сложно определить рекурсивные переборные алгоритмы. Для сложных композиций сервисов применение графических средств сталкивается со сложностью восприятия пользователем диаграмм с большим количеством элементов и связей между этими элементами [41]. Метод задания композиции сервисов в виде файла в одном из стандартов задания композиций сервисов требует знания языков разметки и больших временных затрат при составлении композиций с большим количеством сервисов и их взаимодействий.

В настоящий момент основными текстовыми 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).

Стандарт BPEL – XML стандарт, определяет взаимодействие и обмен данными между сервисами в сервис-ориентированной среде. Как правило, BPEL чаще всего используется для описания бизнес-процессов, происходящих внутри организаций, (создание и доставка периодических отчетов, проверка состояния систем, автоматизация платежей и т.д.). Обычно бизнес-процессы, которые необходимо определить, требуют выполнения нескольких сервисов, совместную работу которых необходимо скоординировать, то есть задать композицию сервисов. Стандарт BPEL предоставляет широкие возможности для задания композиций – доступны логические операторы, операторы циклов, асинхронные вызовы сервисов, промежуточная обработка результатов работы сервисов, обработка происходящих событий и возникающих ошибок, выполнение сценариев отката действий. Заданная с помощью BPEL композиция сама по себе является веб-сервисом, который можно вызвать извне или задействовать в другой композиции сервисов. Так как композиции сервисов, заданные с помощью стандарта BPEL являются XML файлами, корректными относительно соответствующих XML Schema Definition (XSD) файлов

стандарта, задание композиций в формате BPEL является трудоемким и затратным по времени процессом. Стоит отметить, что актуальный стандарт называется WS-BPEL 2.0 (Web services BPEL), но в некоторых источниках можно встретить название BPEL4WS (BPEL for web services). В настоящее время принято использовать название BPEL, если не требуется указать конкретную версию стандарта.

Задание композиций сервисов в формате BPEL требует знаний и опыта работы с XML, то есть возникает проблема доступности задания композиций сервисов для специалистов, не знакомых с программированием и языком разметки XML. Для решения данной проблемы (с одновременным сохранением всех возможностей для настройки взаимодействия сервисов, предоставляемых BPEL) создан стандарт BPMN (Business Process Modeling Notation). Стандарт BPMN является набором графических примитивов, позволяющих моделировать процесс, состоящий из нескольких сервисов и включающий в себя условные, цикличные конструкции, асинхронные вызовы сервисов, обработки ошибок и происходящих событий. Как правило, программные средства, позволяющие задавать композиции сервисов с помощью графических примитивов в рамках стандарта BPMN, хранят созданные сценарии в формате BPEL [42], однако стоит отметить, что некоторые возможности BPMN (версия стандарта 2.0) на данный момент невозможно задать с помощью BPEL. В силу отсутствия поддержки некоторых возможностей BPMN в BPEL, затруднительно совместное использование данных инструментов, так как корректный перевод из одного формата в другой не гарантируется.

Стандарт BPML задает композицию в виде XML файла. BPML был разработан в одно время со стандартом BPEL и отличался от него поддержкой Pi-исчисления, что позволяло выполнять параллельные процессы, конфигурация которых может меняться в процессе вычислений. Основным отличием BPML от BPEL является то,

что с помощью BPML можно описать практически любой процесс, в то время как ограничения BPEL принуждают пользователя реализовывать дополнительную логику с помощью какого-либо языка программирования, например, Java. В настоящий момент BPML поддерживается в меньшем количестве систем, нежели BPEL.

Стандарт XPDL задает композицию сервисов в виде XML файла, позволяет описывать сразу несколько процессов в одном файле, поддерживает как веб-ориентированные, так и сервисы, работающие не через сеть Интернет. XPDL позволяет хранить информацию, относящуюся к графическому представлению композиций сервисов и используемую в специализированных программных средствах.

Для задания композиций сервисов существует большое количество программных средств, упрощающих как процесс задания, так и осуществляющих контроль выполнения композиций.

Одним из часто используемых программных средств для работы с композициями сервисов является Oracle BPEL Process Manager, коммерческий продукт компании Oracle, который является декстопным приложением, ориентированным на работу с веб-сервисами, работающими на основе XML, SOAP и WSDL и работающим с композициями сервисов, заданными с помощью стандарта BPEL. Приложение имеет графический интерфейс, композиции сервисов задаются с помощью графических примитивов путем перетаскивания требуемых элементов композиции на специальную рабочую область. Так как Oracle BPEL Process Manager ориентирован на работу с композициями, заданными в формате BPEL, он поддерживает такие отличительные особенности BPEL, как параллельное выполнение сервисов, обработка событий и исключительных ситуаций, выполнение длительных процессов. Oracle BPEL Process Manager обладает

кроссплатформенностью, так как поставляется в составе программного пакета SOASuite разработки компании Oracle, который, в свою очередь, запускается в виртуальной машине Java, которая поддерживается на всех современных операционных системах. Среди недостатков Oracle BPEL Process Manager можно отметить отсутствие поддержки сервисов, имеющих интерфейс, отличный от SOAP/WSDL, а также отсутствие механизмов планирования выполнения сервисов в рамках композиций. Также Oracle BPEL Process Manager требует установки дополнительного программного обеспечения из пакета Oracle SOA Suite, что предъявляет повышенные требования к аппаратным характеристикам машины, на которой запущен Oracle BPEL Process Manager.

Для проведения междисциплинарных научных исследований используется программный продукт Apache Taverna, позволяющий с помощью графического интерфейса определять последовательности выполнения заданий и хранить заданные композиции в специально разработанном формате SCUFL2. В отличие от императивного BPEL с явным заданием процесса выполнения, Apache Taverna использует функциональную модель с управлением данными. Рассматриваемый программный продукт представляет собой написанное на Java приложение, предоставляющее веб-интерфейс и управляющее и контролирующее процесс выполнения композиций сервисов. Apache Taverna распространена в научной среде и часто используется в астрономии, биоинформатике, физике. Однако, Taverna не осуществляет планирование выполнения композиций в сложных средах.

Другим известным программным средством для работы с композициями сервисов является Apache ODE. Apache ODE также, как и Oracle BPEL Process Manager, использует для работы с композициями сервисов стандарт BPEL, обеспечивая при этом поддержку всех основных возможностей, определенных в стандарте BPEL. Apache ODE не имеет графического интерфейса для задания

композиций сервисов. Рассматриваемый программный продукт имеет встроенный планировщик выполнения композиций сервисов, но его основная цель – обеспечение устойчивости выполнения долгосрочных композиций. Хранение состояния композиции производится в реляционной базе данных.

Среди отечественных разработок стоит отметить программную среду Everest [43], позволяющую публиковать сервисы, из которых впоследствии можно составлять сценарии композиции на языке программирования Python. Каждый вызываемый в композиции сервис соответствует некоей уникальной функции, которая принимает на вход параметры и вызывает сервис. Основным недостатком данной системы является синхронность запуска сервисов (по мере вызова функций в сценарии), а также неустойчивость процесса выполнения сценария к изменениям, происходящим в распределенной среде сервисов.

Рассмотрев все вышеперечисленные программные инструменты как в теоретическом, так и в практическом плане, можно отметить, что ни один из инструментов не предоставляет одновременную возможность задания композиций сервисов в виде программ на одном из распространенных языков программирования и планирования выполнения композиций в распределённых гетерогенных средах. Также стоит заметить, что само по себе задание композиций с помощью графических примитивов сложно в использовании при задании рекурсивных конструкций, при решении переборных задач. Внедрение промежуточных обработок на каком-либо языке программирования во всех представленных программных продуктах затруднительно.

1.4 Планирование выполнения композиции сервисов

Общепризнанным стандартом постановки задачи планирования композиции сервисов является определение зависимостей между заданиями с помощью направленного ациклического графа (Directed acyclic graph, DAG) [44,45]. В таком

графе вершинами являются вызовы сервисов, а дугами являются зависимости между сервисами. Сервисы, которые имеют только выходные дуги, называются входными сервисами, соответственно, те сервисы, которые имеют только входные дуги, называются выходными (результирующими сервисами). В композиции может быть несколько как входных, так и выходных сервисов. DAG представляет из себя множество пар (T, E), где T – множество вершин, представляющих собой задания, которые необходимо выполнить, а E множество рёбер графа, представляющих собой зависимости между заданиями. Если существует $e \in E$, соединяющее вершины (t_1, t_2), значит что t_2 не может начать выполняться, пока не завершит свою работу t_1 и не передаст данные n_2. Возможна ситуация, когда существует несколько таких $e_{\overline{i=1,T}} \in E$, где $e_1 = (t_1, t_K), e_2 = (t_2, t_K) \ldots e_T = (t_T, t_K)$, то есть задание t_K зависит от заданий $t_1, t_2 \ldots t_T$. Графическое представление DAG представлено на Рис. 1. Задания без зависимостей являются начальными (окружности с идентификаторами 1, 2 на Рис. 1), с которых начинается выполнение композиции, а задание без зависимых от него заданий является выходным (окружность с идентификатором 4 на Рис. 1). В DAG есть как минимум один входной и выходной (терминальный) узел.

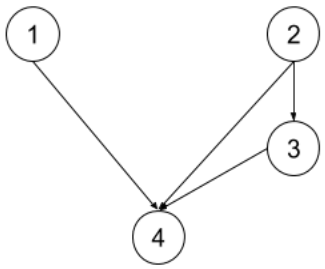

Рис. 1. Пример направленного ациклического графа

Рассматриваемая задача планирования выполнения композиций предполагает как различающееся время выполнения заданий, так и разное время, необходимое для передачи данных между заданиями.

Время, необходимое для выполнения T заданий на N вычислительных узлах определяется с помощью матрицы M размерности $t \times n$, где каждый элемент матрицы $m_{i,j}$ обозначает время выполнения задания t_i на узле n_j. Возможна такая ситуация, что $m_{i,j} = 0$, в таком случае подразумевается, что вычислительный узел n_j не может выполнить задание t_i.

Время, необходимое для передачи данных, определяется следующим образом: для каждой пары вычислительных узлов (n_i, n_j) определяется некая величина $r_{i,j}$, обозначающая среднюю пропускную способность сетевого канала между вычислительными узлами n_i и n_j. Для каждого $e \in E$ определяется объём данных $d_{i,j}$, который необходимо передать между заданиями t_i и t_j. Таким образом, величина $c_{i,j}$, обозначающая время, необходимое для передачи данных между заданиями t_i и t_j, определяется как $c_{i,j} = \frac{d_{i,j}}{r_{n,m}}$, где i и j – индексы заданий, n и m – индексы вычислительных узлов, на которых соответственно выполняются задания t_i и t_j.Множество $succ(t_i)$ содержит задания, напрямую зависимые от задания t_i. Для выходных узлов справедливо утверждение, что $succ(t) = \varnothing$.

Задача планирования расписания заключается в назначении заданий на вычислительные узлы таким образом, что общее время выполнения композиции будет минимальным. При этом должны учитываться зависимости между заданиями по данным, а также время выполнения заданий и время, необходимое для передачи данных между заданиями.

Определив задачу планирования выполнения композиции сервисов в гетерогенной распределённой среде, необходимо рассмотреть самые распространенные алгоритмы планирования, которые можно разделить на две

группы по способу нахождения расписания – эвристические и мета-эвристические алгоритмы.

1.4.1 Эвристические алгоритмы нахождения расписания

Эвристические алгоритмы нахождения расписания можно разбить на два класса – статические и динамические алгоритмы. Статические алгоритмы предполагают, что расписание выполнения строится перед непосредственным выполнением скрипта, т. е. все данные о времени выполнения заданий, зависимостях между заданиями, а также временных затратах на передачу данных между вычислительными узлами известны заранее. Динамические алгоритмы предполагают назначение задания на вычислительные узлы по мере их появления и решения планировщика принимаются по мере выполнения композиции.

В настоящее время существует большое количество эвристик нахождения расписания выполнения композиций, задаваемых в виде DAG. Обычно эвристики используются в списковых алгоритмах. Списковые алгоритмы предполагают сортировку списка заданий на основе определенной величины, рассчитываемой с помощью эвристики. Назначение заданий происходит в том порядке, в каком задания расположены в списке. Если несколько заданий в списке имеют одинаковое значение эвристики, в таком случае выбор назначаемого задания происходит случайным образом. Алгоритм планирования делится на этап расстановки приоритетов (рангов) для заданий и этап назначения заданий на вычислительные узлы. Одними из самых эффективных списковых алгоритмов являются алгоритмы HEFT (Heterogeneous Earliest Finish Time) [19], SDBATS (Standard Deviation Based Algorithm for Task Scheduling) [46], PEFT (Predict Earliest Finish Time) [47] и HSIP (Heterogeneous Scheduling with Improved Task Priority) [48].

Распространенным эвристическим алгоритмом является алгоритм Heterogeneous Earliest Finish Time (HEFT) [19]. Величина для каждого задания

определяется суммой среднего времени выполнения данного задания на всех вычислительных узлах и наибольшего значения эвристики для узлов, напрямую зависящих от данного задания, то есть ранг определяется по формуле $rank\ (t_i) = \overline{w_i} + \max_{t_j \in succ\ (t_i)} (\overline{c_{i,j}} + rank\ (t_j))$, где $\overline{w_i}$ – среднее время выполнения задания на всех вычислительных узлах, а $\overline{c_{i,j}}$ – средняя стоимость передачи данных между заданиями t_i и t_j для всех пар вычислительных узлов. Расчет данной величины начинается с выходных заданий. Ситуация, когда для одного заданий существует несколько значений данной величины (у разных зависимых заданий могут получаться разные величины), разрешается в пользу наибольшего значения данной величины. В случае, когда время выполнения одного задания на разных узлах сильно различается и время передачи данных между заданиями сравнительно больше, HEFT составляет расписание с большой степенью ошибки. Данный алгоритм на основании произведённых сравнений [49] был признан самым эффективным на данный момент из алгоритмов, не использующих специальных техник.

Алгоритм SDBATS исключает недостаток алгоритма HEFT, используя для вычисления ранга задания не среднее время выполнения задания на узлах, а используя среднеквадратичное отклонение значений времени выполнения задания на разных вычислительных узлах. Данное изменение, в свою очередь, приводит к составлению крайне неточных расписаний в случае если время, необходимое для передачи данных, довольно большое. Также алгоритм SDBATS предполагает запуск входных заданий на всех доступных вычислительных узлах, что ухудшает расписание в случае если время выполнения одного и того же задания сильно различается на разных вычислительных узлах.

Другим часто алгоритмом является алгоритм PEFT [47]. PEFT предлагает использовать вместо рассчитанных для каждого задания приоритетов специальные

таблицы OCT (Optimistic cost table), представляющих собой матрицы, в которых строки отображают задания, содержащиеся в рассматриваемом DAG, а столбцы отображают вычислительные узлы, на которых могут выполняться рассматриваемые сервисы. Каждый элемент матрицы $OCT\ (t_i, r_j)$ содержит максимальное из всех значений ранга для дочерних (напрямую зависимых от t_i) заданий, при условии, что выполнение происходит на вычислительном узле r_j. Также в алгоритме на каждом этапе назначения задания на вычислительный узел производится оценка возможных временных значений для решения на некоторое количество шагов вперед. Таким образом, алгоритм PEFT составляет более эффективные расписания по сравнению с общепризнанным алгоритмом HEFT.

Алгоритм HSIP является одним из последних списковых алгоритмов, показывающих лучшие результаты в составлении расписаний, как на искусственных, так и реальных задачах. HSIP акцентирует внимание на гетерогенной природе распределенной вычислительной среды, и предлагает подсчет эвристики не только с учетом нормального распределения, но и учетом затрат на передачу данных между вычислительными узлами. Также алгоритм предлагает дупликацию начальных заданий графа с учетом различающегося времени выполнения заданий на разных вычислительных узлах. В соответствии с расчетами, приведёнными в статье [46], алгоритм HSIP на данный момент является самым эффективным статическим эвристическим алгоритмом для построения расписания выполнения DAG.

1.4.2 Метаэвристические алгоритмы нахождения расписания

Среди метаэвристических алгоритмов часто используемыми алгоритмами являются генетические алгоритмы, а также алгоритмы муравьиной колонии и имитации отжига.

Метаэвристики — это общие эвристики, позволяющие находить близкие к оптимальным решения различных задач оптимизации за приемлемое время. Метаэвристики пытаются объединить основные эвристические методы в рамках алгоритмических схем более высокого уровня, направленных на эффективное изучение пространства поиска. Метаэвристики включают две категории: метаэвристики локального поиска и эволюционные алгоритмы.

Генетические алгоритмы позволяют находить приближенные решения из широкого пространства поиска, применяя эволюционные принципы, причем они обычно находят более точные решения, нежели эвристические алгоритмы, но проигрывают по времени выполнения [21]. Генетические алгоритмы работают по принципу улучшения требуемых свойств каждого следующего получаемого поколения в соответствии с принципами природной генетики. Алгоритмы такого рода в своей работе используют набор отдельных элементов (популяция) и набор операторов, аналогичных биологическим и определенным над популяцией, и используемых для манипуляций с популяцией в целях оптимизации и нахождения приемлемого решения. В соответствии с эволюционными теориями, только самые приспособленные элементы популяции выживают и производят потомство, наследующее требуемые характеристики. Генетический алгоритм переносит проблему планирования на набор строк (хромосом), каждая строка представляет потенциальное решение. Три основных аспекта использования генетических алгоритмов заключаются в следующем: определение целевой функции, определение и реализация генетического представления и определение и реализация генетических операторов.

Основной идеей алгоритма муравьиной колонии является симуляция поведения муравьев при поиске пропитания. Когда группа агентов (муравьев) ищет пропитание, они используют особый тип химического вещества для связи друг с

другом, обычно называемом феромоном. Изначально муравьи начинают поиск еды случайным образом, в случайном направлении. При нахождении еды муравей оставляет определенную величину феромона на пути к источнику еды. По мере работы алгоритма, большинство муравьев выбирает наикратчайшее расстояние до источника еды на основании концентрации феромонов, сосредоточенных на пути к источнику. Основным преимуществом такого рода алгоритма является механизм позитивной реакции, внутреннего параллелизма и расширяемости. Недостатком алгоритма является явление, когда на каком-то этапе работы алгоритма все муравьи приходят к одному и тому же источнику еду (решению), однако данное решение может быть лишь локальным минимумом. Алгоритм муравьиной колонии возможно применить практически к любой комбинаторной задаче [50].

Изначально алгоритм симуляции отжига был разработан по аналогии с химическими процессами, происходящими при отжиге металла. Отжиг включает в себя нагревание и остывание металла в целях изменения его физических свойств в результате изменений в его внутренней структуре. По мере остывания металла структура фиксируется, при этом сохраняя свои приобретенные свойства. В симуляции отжига постоянному изменению подвергается температура, при которой происходит процесс отжига. Изначально температура достаточно высока, затем она плавно понижается по мере работы алгоритма. Чем выше температура, тем чаще алгоритму будет разрешаться принимать решения, которые хуже текущего решения. Это дает алгоритму возможность преодолевать локальные оптимумы, которые находятся в начале вычисления. По мере понижения температуры шанс принятия заведомого неверного решения уменьшается, тем самым алгоритм больше концентрируется на области поиска, которая, скорее всего, приведет к нахождению оптимального решения. Постепенное понижение температуры обосновывает основное преимущество алгоритма – высокая близость решения к оптимальному решению при наличии большого изначального числа локальных оптимумов [51].

Алгоритм PSO (Particle swarm optimization) был опубликован в 1995 году и является алгоритмом, моделирующим поведение стаи, решающей оптимизационную проблему в определенном пространстве поиска [52]. Поведение стаи базируется на социально-биологических принципах и обеспечивает понимание социального поведения, а также находит свое применение в решении задач из области информационных технологии и инженерии. Алгоритм PSO предполагает, что даются некая функция, позволяющая оценить найденное решение, и структура сети, соединяющая отдельных участников стаи, то есть определяется связанность каждого участника с другими участниками. Затем случайным образом задается начальная популяция участников стаи. Далее запускается процесс движения участников стаи, на каждом из этапов производится оценка полученного решения, в случае получения удовлетворительной оценки участники стаи запоминают свое положение. Информация о текущем положении участника и о его положении, где решение было удовлетворительным, доступна его соседям. Передвижения участников стаи осуществляются на основании успешных передвижении соседей. В итоге обычно популяции участников стаи сходится, давая удовлетворяющее решение. Каждый участник стаи представляет собой решение оптимизационной проблемы. Расположение участника обуславливается как его собственным опытом, так и положением его соседей. Когда соседями участника стаи становится вся стая, тогда лучшее расположение рассматриваемого участника и является глобально лучшим решением оптимизационной задачи методом роя частиц. Если же используется меньшее число соседей, то тогда лучшее решение рассматриваемого участника является локально лучшим решением оптимизационной задачи. Производительность каждой частицы измеряется с помощью целевой функции, которая устанавливается в зависимости от самой оптимизационной проблемы.

Также в последнее время активно развивается SAT (propositional SATisfiability problem) подход [53], который применяется в задачах планирования. SAT

предполагает составление булевой формулы, описывающей искомый план. Далее специальный решатель пытается найти такие значения переменных в формуле, что формула была бы истинной. Найденные значения переменных дают план, гарантирующий наилучшее время выполнения задач. Проблема применения данного подхода заключается в том, что время нахождения плана невозможно предугадать. Немаловажным фактором является то, что часто в задаче поиска плана нахождение точного решения плана некритично, в то время как непосредственное время поиска имеет большое значение.

Стоить отметить, что большинство алгоритмов планирования разрабатывались с расчетом на использование в высокопроизводительных вычислениях, когда вычислительные узлы почти гарантировано способны выполнять сервисы и чьи характеристики, а также их степень занятости заранее известны. С развитием сервис-ориентированного подхода характеристики среды изменились в силу её гетерогенности, которая заключается в том, что вычислительные узлы, на которых развернуты сервисы, обладают различной производительностью. То же самое касается сетевой инфраструктуры, в которую интегрированы вычислительные узлы. В совокупности эти факторы дают различное время выполнения для одного и того же сервиса с одними и теми же входными параметрами на разных вычислительных узлах.

1.4.3 Гибридные методы планирования

В последнее время стали все чаще появляться гибридные алгоритмы, сочетающие в себе разные техники – например, алгоритм Hybrid Evolutionary Workflow Scheduling Algorithm for Dynamic Heterogeneous Distributed Computational Environment [18], сочетающий в себе HEFT и генетический алгоритм. Данный алгоритм способен адаптироваться к изменениям вычислительной среды, что особенно важно в распределенных вычислительных средах, когда работа отдельных

вычислительных узлов и каналов связи не гарантируется, а также когда время выполнения сервисов и передачи данных между ними также изменяется со временем. Также данный алгоритм учитывает изменения, происходящие в вычислительной среде – например, отключение вычислительного узла или добавление вычислительных сервисов к композиции ведет к мгновенной перестройке расписания, в то время как несоответствие фактического и ожидаемого времени выполнения сервиса будет учтено только в момент следующей плановой перестройки расписания.

1.5 Выводы

В настоящее время существует ряд методов разработки программных средств, составления композиций сервисов, также развита область планирования выполнения распределенных сервисов, но не существует программного продукта составляющего композиции сервисов в форме обычных программ на одном из распространенных языков программирования. Данный программный продукт должен самостоятельно, без участия пользователя, осуществлять планирование выполнения сервисов на основании характеристик вычислительных узлов, а также специфики распределенной среды сервисов, в том числе её гетерогенности. Причём планирование должно осуществляться динамически, то есть по мере выполнения сценария, в целях своевременного и корректного реагирования на возможные изменения в вычислительной среде.

ГЛАВА 2. ПЛАНИРОВАНИЕ ВЫПОЛНЕНИЯ И ЗАДАНИЕ КОМПОЗИЦИЙ СЕРВИСОВ

В данной главе рассмотрена проблема задания и планирования выполнения композиций сервисов в гетерогенной среде. Предлагается задавать композиции сервисов в виде программ на одном из распространенных языков программирования, так как методы задания композиций графический (с использованием графических примитивов) и текстовый (с использованием одного из современных стандартов), обладают рядом ограничений (см. Глава 1). В свою очередь, уже существующие методы задания композиций с помощью процедурных языков не предполагают распараллеливания выполнения композиций сервисов. Предлагаемый метод задания осуществляет автоматическое построение направленного ациклического графа (Directed acyclic graph, DAG) композиций.

Наравне с проблемой задания композиций сервисов рассмотрена проблема планирования их выполнения с учётом изменений характеристик вычислительной среды (время выполнения сервисов, доступность вычислительных узлов, время передачи данных между узлами, структура DAG композиции и т.д.).

2.1 Математическая модель планирования выполнения композиций сервисов

В распределённой вычислительной гетерогенной среде планирование выполнения композиции сервисов, является актуальным, т.к. сервисы могут обладать различной длительностью выполнения на разных вычислительных узлах в зависимости от входных данных, вызовы сервисов могут быть зависимы по данным друг от друга, а количество вычислительных узлов быть ограничено.

Композиция сервисов можно представить в виде направленного ациклического графа, вершины которого определяет задания, (вызовы сервисов), а рёбра

определяют зависимости между заданиями по данным. Задания, не имеющие зависимостей, называются входными, а задания, не имеющие зависимых заданий и зависимые от каких-либо других заданий, называются выходными, или терминальными заданиями. Пример такого графа приведен на Рис. 2, где задания с идентификаторами 1 и 2 являются входными, а задание с идентификатором 4 является выходным. Стоит отметить, что выходных вершин может быть несколько.

В процессе выполнения сценария сервисов список заданий может пополняться, то есть граф расширяется. На Рис. 2 представлена ситуация, когда уже существующий сценарий дополняется еще двумя заданиями с идентификаторами 5 и 6, таким образом, единственным выходным заданием становится задание с идентификатором 6.

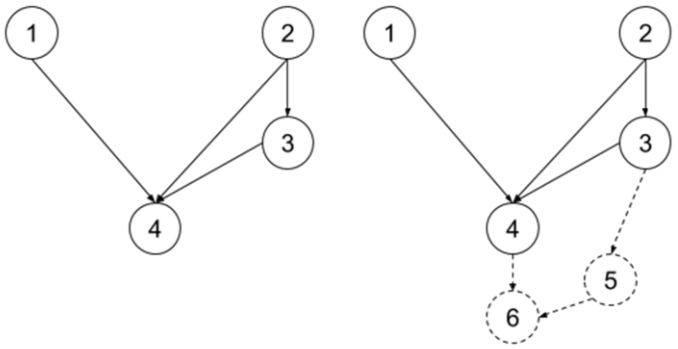

Рис. 2. Добавление заданий в граф зависимостей

На одном вычислительном узле могут быть развернуты один или несколько сервисов. Один и тот же сервис может быть развернут на нескольких вычислительных узлах. Соответственно задание может выполняться на нескольких узлах, на которых развернут сервис. Предполагается, что на одном вычислительном узле в один момент времени может выполняться только одно задание.

При выполнении композиции сервисов обычно имеется ограниченное количество вычислительных узлов. Некоторые сервисы в составе композиции могут

выполняться достаточно долгое время. Порядок назначения заданий на вычислительные узлы влияет на общее время выполнения композиции. Таким образом, возникает задача построения плана (расписания) выполнения заданий, определяющего последовательность и узлы выполнения заданий с целью уменьшения времени выполнения всех заданий композиции сервисов на ограниченном количестве вычислительных узлов.

Задача поиска расписания является NP-полной, т. е. нахождение оптимального решения в некоторых случаях для такого рода задач является практически невыполнимым. Таким образом, необходимо осуществлять поиск приемлемого решения, то есть решения, максимально приближенного к оптимальному решению.

Представим задачу выполнения композиции сервисов в виде математической модели *M=(N, S, T,E,W)*, где:

N – множество вычислительных узлов, которое может меняться с течением времени, например из-за изменения квоты на ресурсы, поломок узлов и т.д., предполагается в этой модели, что эти изменения трудно спрогнозировать;

S – множество сервисов, имеющихся в среде;

W – множество пар (s_i, n_j), показывающее, что сервис s_i установлен на узле n_j. Множество *W* может также меняться в процессе выполнения композиции, например, в результате сбоя работы сервиса на определенном узле пара (s_i, n_j) исключается из множества *W* чтобы не повторять ошибочное выполнение;

T – множество заданий, каждое задание выполняется определенным сервисом. Задание может выполнено на любом узле, на котором установлен сервис. Множество *T* может расти в процессе выполнения композиции, в частности из-за ветвления алгоритма в зависимости от результатов работы сервисов композиции могут добавлять разные задания;

E – множество рёбер графа, представляющих собой зависимости между заданиями данным, соответственно *(T,E)* представляют ациклический направленный граф (DAG) зависимостей заданий по данным. Изменение множества производится одновременно с множеством *T*. Добавление новых заданий может привести к росту множества E , добавляемые задания могут зависеть от уже существующих заданий. T^{start} и T^{exit} – подмножества T, содержащие начальные и выходные заданияDAG, то есть не существует таких заданий t_i, что выполняется $depends\ (t_{start}, t_i)$ и $depends\ (t_i, t_{exit})$ соответственно.

Определим $P(T, N)$ как некое расписание выполнения заданий, назначенных на какие-либо узлы. Расписание $P(T, N)$ является набор упорядоченных последовательностей заданий вида $(t_i, ..., t_k)_j$ для каждого узла n_j. Множеством допустимых расписаний называется набор всех расписаний для заданий T и вычислительных узлов N , для которых выполняется отношение предшествования заданий и возможности выполнения заданий на узлах.

Определим вспомогательные функции:

$duration\ (t_i, n_j)$ – оценка длительности выполнения задания t_i на n_j поддерживающем его узле;

$transmisit\ \ ion\ (t_i, t_j, n_k, n_l)$ – оценка длительности передачи данных задания t_i для t_j с узла n_k на узел n_l (если t_i и t_j выполняются на одном узле, то $transmisit\ \ ion\ (t_i, t_j, n_k, n_l) = 0$);

$busy\ (n_i, P)$ – время освобождения узла, то есть момент времени, когда на вычислительный узел можно назначать новые задания. Если на узел не назначены задания, то значение $busy\ (n_i, P)$ равно текущему времени (нельзя назначить задание на вычислительный узел в прошлом, то есть назначение задания должно происходить не раньше текущего момента времени). Если на вычислительный узел

назначены какие-либо задания, то значение $busy\ (n_i, P)$ равно либо времени завершения последнего назначенного задания, либо значению текущего времени, если оно больше чем время завершения последнего назначенного задания.

Определим целевую функцию $evaluate\ (P) = \max_{0 \le i < N} \{busy\ (n_i, P)\}$, определяющую время завершения работы всего расписания. Задача планирования заключается в построении такого расписания, которое бы минимизировало время выполнения композиции сервисов $evaluate\ (P) \to \min$ на множестве допустимых расписаний. Необходимо учитывать, что время построения расписания может быть большим и в некоторых случаях больше чем время вычисления любого допустимого расписания. Поэтому необходимо производить построение расписания в пределах интервала времени L. Стоит отметить, что реальная длительность выполнения задания может отличаться от оценки $duration\ (t_i, n_j)$. Также реальная длительность передачи данных может отличаться от оценки $transmisit\ ion\ (t_i, t_j, n_k, n_l)$.

Событие – это изменение вычислительной среды, требующее перепланирования выполнения сервисов.

События, влияющие на целевую функцию:

- добавление и удаление вычислительных узлов – количество выделенных вычислительных узлов в облаке может варьироваться;
- добавление задания в DAG – в процессе выполнения композиции сервисов возможно добавление новых заданий в зависимости от начальных условий и промежуточных значений;
- задание выполнялось меньше или дольше ожидаемого времени – в зависимости от текущих данных , состояния вычислительного узла время выполнения может измениться значительно, предполагается, что композиции выполняются многократно и можно использовать статистику выполнения заданий;

- задание завершилось с ошибкой – причиной может быть некорректное состояние вычислительного узла или сервиса и т.д. Система выполнения композиции сервисов должна произвести попытки выполнения на других узлах, возможно с другими реализациями сервиса.

2.2 Задание композиций сервисов

Существует несколько способов задания композиций сервисов. В соответствии с разделом 1.3, их использование осложняется тем, что текстовые и графические способы задания характеризуются сложностью задания определенных алгоритмов с как с использованием графических примитивов, так и с применением существующих XML стандартов.

Рассмотрим способ задания композиций на процедурном языке программирования, который обладает следующими преимуществами:

1) процедурное программирования является одной из самых распространенных парадигм программирования, что позволяет быстро освоить задание композиций;
2) использование существующих библиотек для выбранного языка программирования;
3) работа с результатами выполнения сервисов в теле композиций стандартными средствами языка.

Таким образом, задание композиций сервисов на процедурном языке программирования является перспективной альтернативой существующим способам и имеет определённые преимущества.

В распределенной гетерогенной среде, в условиях ограниченности ресурсов, жестко заданной последовательности заданий, сервисы имеют различную

длительность выполнения. Таким образом, при выполнении композиций сервисов необходимо осуществлять планирование выполнения сервисов.

Планирование выполнения композиций сервисов можно осуществлять непосредственно в программном коде композиций, например, назначая каждый вызов сервиса на определенный вычислительный узел. Однако, такой подход не учитывает изменения выделяемых ресурсов (изменение доступности вычислительных узлов может привести к невозможности завершения выполнения композиции) и усложняет программирование композиций (пользователь должен учитывать распределение всех вызовов сервисов по вычислительным узлам по мере задания композиции). Таким образом, распределение заданий по вычислительным узлам необходимо осуществлять автоматически, по мере выполнения композиции с учетом состояния гетерогенной вычислительной среды.

Для планирования выполнения композиций сервисов разработано большое количество алгоритмов, рассмотренных в разделе 1.4. Все рассмотренные алгоритмы требуют представление композиции в виде DAG. Таким образом, преобразование композиции, заданной на процедурном языке, в DAG даст возможности применения существующих алгоритмов планирования.

Код программы на процедурном языке, задающий композицию сервисов, далее в работе будет называться сценарием композиции.

2.2.1 Формирование узлов графа DAG (заданий)

Для формирования узлов графа в рамках выбранного языка программирования можно создать библиотеку, в которой для каждого зарегистрированного в системе сервиса сформирована уникальная функция (функция-обертка), с помощью которой осуществляется вызов сервиса. При вызове функций-оберток происходит регистрация нового задания, т.е. добавление вершины в граф композиции с

указанием сервиса, его характеристик и передаваемых и получаемых данных. Стоит отметить, что 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 заданий.

Вызов функций-оберток не требует блокировки выполнения сценария, так как это не приводит к непосредственному вызову сервиса. Блокирование выполнения сценария необходимо, если выполнение сценария зависит от данных, получаемых в процессе выполнения сервисов. Соответственно, граф DAG формируется по мере выполнения сценария и завершения работы сервисов. Учитывая ограниченное количество вычислительных ресурсов, при добавлении новой вершины в графе DAG требуется перепланирование.

2.2.2 Определение ребер

Определение ребер DAG, т.е. зависимостей между вызовами сервисов по данным, необходимо при планировании, так как для определения того, готов ли сервис к запуску, необходимо проверить, выполнились ли все сервисы, от которых он зависит по данным. Определение зависимостей между заданиями при процедурном задании композиции сервисов является нетривиальной задачей, так как результаты выполнения сервисов могут помещаться в переменные, значения переменных затем могут переопределяться, то есть отследить результаты выполнения сервисов на основе статического анализа кода практически сложно. Поэтому предлагается для определения ребер DAG при вызове функции-обертки сервиса в качестве параметров передавать объекты специального вида, выступающие в роли контейнеров для данных и позволяющие однозначно идентифицировать передаваемые и получаемые данные. В дальнейшем контейнеры могут передаваться в качестве входных параметров для других сервисов. Каждый контейнер имеет свой идентификатор, и данные в него могут быть записаны единственный раз. Запись данных осуществляется специальным методом или при

завершении выполнения сервиса результаты его работы автоматически помещаются в данные контейнеры. При генерации функций-оберток автоматически формируется интерфейс, требующий использование контейнеров.

Определение зависимости между вызовами сервисов по данным происходит следующим образом – при выполнении функции-обертки сервиса происходит анализ входных данных функции, если среди входных данных присутствует хотя бы один контейнер, то происходит запоминание его идентификатора и проверка, использует ли какой-либо уже произошедший вызов сервиса данный контейнер для хранения результатов выполнения. В случае если проверка находит такой сервис, происходит добавление ребра в граф, то есть обнаружена зависимость между сервисами по данным.

Стоит отметить, что использование такого рода контейнеров должно предполагать получение данных внутри сценария для последующей обработки средствами выбранного языка программирования. Получение данных должно осуществляться в блокирующем режиме, то есть выполнение сценария композиции продолжится только после наполнения контейнера данными, так как возможна ситуация, что в зависимости от результатов выполнения какого-либо сервиса будет происходить выбор определенной ветви сценария.

Данный подход избавляет пользователя системы от прямого указания зависимости вызовов сервисов друг от друга, то есть определение зависимостей между вызовами сервисов происходит автоматически.

2.2.3 Анализ блокирования выполнения сценария

В силу того, что композиция сервисов задается в виде сценария на процедурном языке программирования, необходимо рассмотреть, каким образом управляющие конструкции языка программирования влияют на составление направленного ациклического графа зависимостей заданий по данным. В

соответствии с теоремой Бёма-Якопини, любой исполняемый алгоритм может быть преобразован к структурированному виду, то есть такому виду, когда ход его выполнения определяется только при помощи трёх структур управления: последовательной (англ. sequence), ветвлений (англ. selection) и повторов или циклов (англ. repetition, cycle) [54]. Данная теорема и статья Эдсгера Дейкстры «Letters to the editor: go to statement considered harmful» [55] дали старт развитию парадигмы структурированного программирования. Таким образом, достаточно рассмотреть влияние основных управляющих структур на составление DAG:

1. Последовательные вызовы команд, в данном случае это могут быть функции-обертки, вызывающие соответствующие сервисы. При выполнении последовательностей команд DAG заполняется последовательно, как проиллюстрировано на Рис. 3 (зависимости между сервисами задаются отдельно, стрелки слева обозначают отношение предшествования вызовов функций в последовательности, стрелки справа, в DAG, обозначают очередность добавления заданий в граф). При последовательном вызове функций-оберток блокирование сценария не требуется;

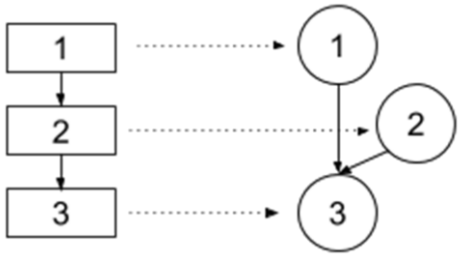

Рис. 3. Добавление задания в DAG в случае последовательности

2. Ветвления представляют собой условные операторы, которые выполняют код в зависимости от определенных условий. Соответственно, наполнение DAG будет зависеть от результата ветвления. Если ветвление происходит на основе результатов одного или нескольких сервисов, функции-обертки которых были

вызваны ранее, то ветвление не произойдёт то тех пор, пока необходимые для принятия решения результаты сервисов не будут получены – использование результатов работы сервисов в ветвлениях является блокирующим для выполнения композиции. На Рис. 4 проиллюстрирован процесс добавления задания в DAG по мере выполнения – добавление заданий 3 и 4 в граф происходит в зависимости от результата условного оператора, выполняющегося после добавления в граф задания 2;

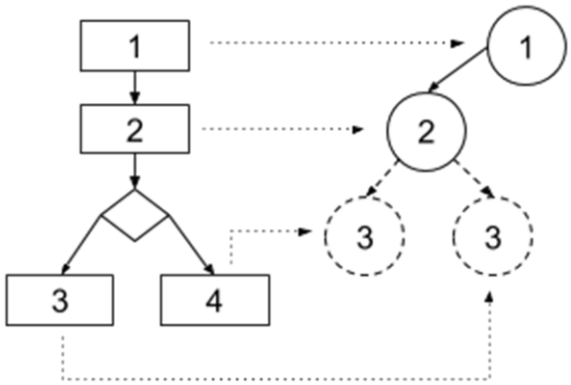

Рис. 4. Добавление задания в DAG в случае ветвления

3. Повторы представляют собой повторяющиеся вызовы определенного кода. Таким образом, если в повторяемом коде присутствуют вызовы каких-либо сервисов, но отсутствует обработка результатов выполнения вызванных сервисов, то есть отсутствуют блокирующие операции, в таком случае DAG наполняется последовательными вызовами повторяющихся сервисов (Рис. 5). Если же внутри повтора осуществляется работа с результатами выполнения вызванных сервисов, в таком случае процесс наполнения DAG будет разделён на периоды ожидания результатов выполнения сервисов с предыдущей операции.

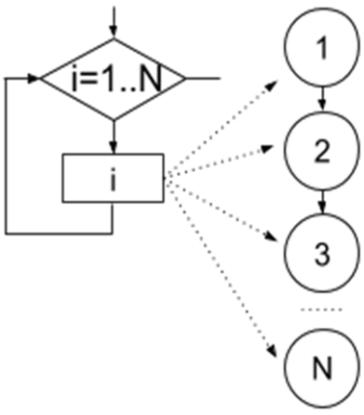

Рис. 5. Добавление задания в DAG в случае повтора

2.2.4 Язык программирования для задания композиций сервисов

В качестве основы для разработанного метода задания композиций выбран язык JavaScript – стремительно развивающийся, за последние несколько лет, ставший не только основой программирования для клиентской стороны веб-приложений (приложения для веб-браузеров), но и активно используемый в серверных приложениях, в том числе в production окружениях известных корпораций [56]. Основными характеристиками языка JavaScript, особенно важными для разрабатываемой системы динамического выполнения композиций сервисов, являются:

1) событийно-ориентированная природа языка;

2) наличие интерпретаторов языка с открытым исходным кодом, что потребуется при реализации собственных команд языка;

3) наличие большого количества библиотек для данного языка;

4) распространенность языка, легкость в обучении и использовании (высокоуровневый, нетипизированный язык), возможность написания кода в разных парадигмах (процедурное, объектно-ориентированное, императивное программирование).

2.3 Планирование выполнения композиций сервисов

2.3.1 Общий алгоритм выполнения композиций сервисов

При выполнении композиций сервисов необходимо учитывать изменение состояния вычислительной среды. Предсказать изменения состояния среды в рамках этой модели считается не возможным, в любой момент времени может поменять количество узлов, измениться длительность выполнения задания и т.д. При изменении состояния вычислительной среды требуется перестроение плана. Имеет смысл проводить перестроение плана только при определенных событиях:

- включение или отключение вычислительного узла, поддерживающего хотя бы один сервис из выполняемой композиции;
- добавление нового задания с готовыми входными данными;
- отличие фактического времени выполнения одного из заданий от ожидаемого;
- завершение выполнения задания с ошибкой.

При наступлении события необходимо сформировать новый план, минимизирующий время выполнения текущего состояния DAG $evaluate\ (P) \rightarrow \min$. Разработанный в данных целях алгоритм можно представить в виде блок-схемы, представленной на Рис. 6.

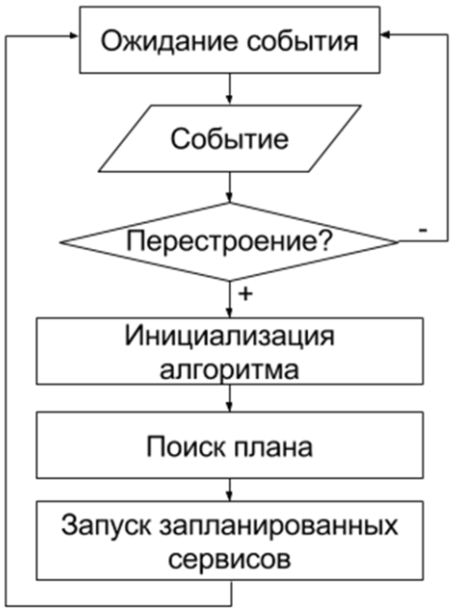

Рис. 6. Алгоритм планирования

При регистрации события происходит определение необходимости перестроения плана. Например, в случае когда при добавлении задания в DAG и назначении его на менее загруженный узел общее время плана не меняется, перестроение не требуется. В случае необходимости перестроения плана, при инициализации алгоритма происходит обновление состояния заданий DAG. Для каждого задания из множества T задается время начала выполнения, если задание уже запущено, время завершения работы если оно уже завершило свою работу, список зависимых заданий, текущее состояние выполнения, набор вычислительных узлов, на которых данное задание может быть запущено. Для каждого вычислительного узла из множества N задается его статус (включен или выключен).

Поиск плана осуществляется на основании DAG и информации о текущем моменте времени. Подробнее о поиске расписания выполнения сервисов будет рассказано в п. 2.3.2.

Результатом планирования является список заданий, для каждого из которых определена предполагаемая длительность выполнения, а для каждого вычислительного узла задана последовательность заданий, выполняющихся на нем.

Длительность выполнения задания помогает выявлять события, касающиеся обнаружения отличия фактического времени выполнения заданий от ожидаемого.

2.3.2 Поиск текущего расписания

Для нахождения расписания, приближенному 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 нахождения приемлемого расписания, являются сортировка заданий, узлов и отсечение неперспективных ветвей графа поиска решения, использование информации о текущем состоянии вычислительной среды для более быстрого и точного нахождения расписания. Рассмотрим каждый шаг алгоритма поиска расписания в отдельности на примере графа на Рис. 7, где заданы идентификаторы заданий, время выполнения заданий и стоимость передачи данных для каждого задания в некоторых абстрактных единицах времени.

Для отсечения неперспективных ветвей поиска используется эвристическая функция $aggregated\ \ Cost\ (t_i)$, вычисляющая суммарное время выполнения задания t_i и всех ее зависимых заданий путем прохождения графа DAG, начиная с выходного задания t_{end}, до самого задания t_i:

$$\textbf{\textit{если}}\ t_i \in T^{\text{exit}}\ \textbf{\textit{то}}\ aggregated\ \ Cost\ (t_i) = w(t_i) + dt(t_i) \qquad \textbf{(1)}$$

$$\textbf{\textit{иначе}}\ aggregated\ \ Cost\ (t_i) = w(t_i) + dt(t_i) + \max(\ aggregated\ \ Cost\ (depends\ (t_i))) \qquad \textbf{(2)},$$

где w – минимальное время выполнения задания среди всех поддерживающих его вычислительных узлов, dt – минимальное ненулевое время получения результатов выполнения данного задания (data transmission). Вычисление значений $aggregated\ \ Cost$ для всех вершин графа зависимостей начинается с выходных вершин, то есть в случае рассматриваемого графа на Рис. 7 – с вершины с идентификатором 4. Так как вершина 4 является выходной, то в соответствии с

формулой (1) значение $aggregated\ \ Cost\ (t_4) = 4$, так как минимальное время выполнения данного задания на всех узлах равно 4 единицам, а время получения результатов выполнения задания для выходных узлов не учитывается. Далее расчет значения $aggregated\ \ Cost$ производится для заданий, от которых зависит задание 4 – это задания с идентификаторами 1, 2, 3. Для заданий, которые не являются выходными, при расчете учитывается как минимальное время выполнения задания, так и время получения результатов его работы (в случае если результатами выполнения задания являются какие-либо файлы, то время получения данных может быть ненулевым). Например, для задания с идентификатором 3 значение $aggregated\ \ Cost$ рассчитывается как $aggregated\ \ Cost\ (t_3) = 4 + 2 + 4 = 10$, где 2 единицы – время получения результатов выполнения задания с идентификатором 3. Нередко возникает ситуация, когда у задания больше чем одно зависимое задание, как например, у задания с идентификатором 2. При расчете значения $aggregated\ \ Cost$ берется максимальное значение $aggregated\ \ Cost$ среди зависимых заданий 3 и 4 – $\max(\ aggregated\ \ Cost\ (t_3), aggregated\ \ Cost\ (t_4)) = \max(\ 10\ , 4) = 10$.

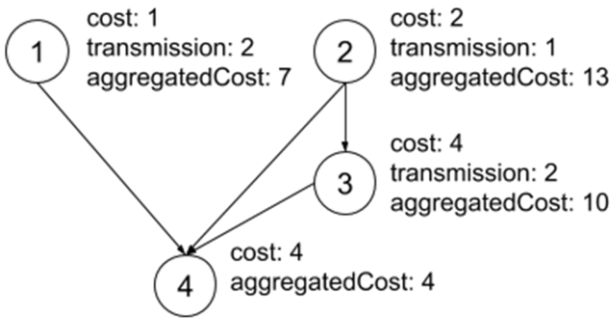

Рис. 7. Нахождение значения aggregatedCost

На вход алгоритма планирования поступает модель *M=(N, S, T,E,W)*, то есть набор описаний заданий с определенными между ними зависимостями и набор описаний вычислительных узлов. Алгоритм планирования содержит несколько шагов:

1. На основании поступившего описания DAG составляется список фиксированных заданий – заданий, которые либо начали выполняться, либо уже завершили свою работу. На основании списка фиксированных заданий воспроизводится фактическое состояние уже выполняющегося плана, что ускоряет и уточняет процесс планирования. Подробнее о данной технике рассказано в п. 2.3.4.
2. Сортировка по убыванию значения функции $aggregated\ \ Cost\ (t_i)$ для каждого нефиксированного задания происходит перед началом работы алгоритма, таким образом, алгоритм будет поочередно выбирать задания для анализа, начиная с самого затратного по времени задания.
3. После каждого назначения задания на определенный узел происходит сортировка узлов по возрастанию значения их занятости. На Рис. 8 приведен пример такой пересортировки при назначении задания 2 на узел 1, в то время как задание 1 уже назначено на узел 2. Данная сортировка позволяет сначала рассматривать случаи, когда самое затратное по времени задание (п. 1) назначается на самый свободный узел.

Рис. 8. Сортировка вычислительных узлов

4. В то время как первые две техники позволяют находить приближенное расписание как можно быстрее, техника пропуска ветвей графа позволяет сократить пространство перебора решений. Данная техника реализуется по

аналогии с алгоритмом A*, в соответствии с которым на каждом шаге решение о рассмотрении какой-либо ветки принимается на основании сравнения значения эвристической функции и текущего рекорда (значения лучшего решения). На каждом шаге рассматривается определенный вычислительный узел n_i и определенное задание t_j, назначаемое на этот узел. Функция эвристики h вычисляется как $h(n_i, t_j) = busy(n_i) + aggregated\ Cost(t_j)$. Пример пропуска ветви представлен на Рис. 9.

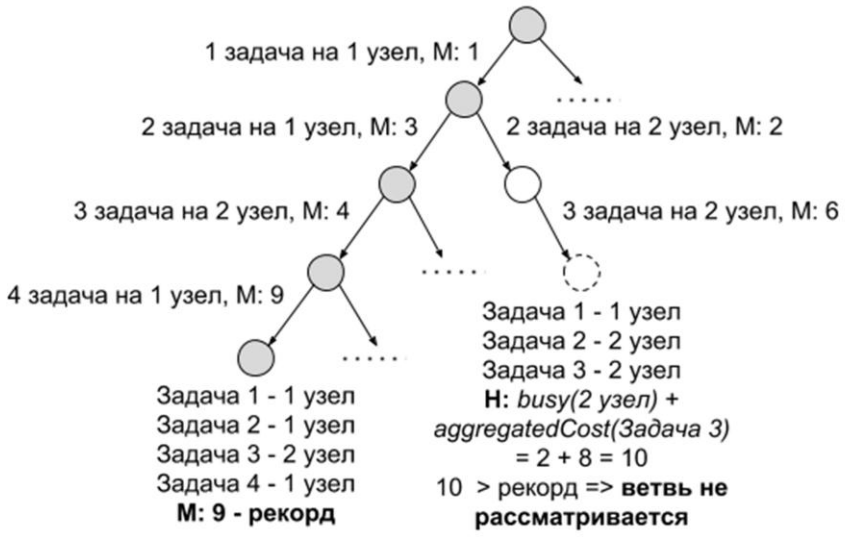

Рис. 9. Отсечение ветвей при выполнении алгоритма

5. Перебор вариантов расписания осуществляется до значения таймаута L, при достижении L планировщик возвращает самое лучшее из построенных на данный момент расписаний.

Таким образом, алгоритм планирования выполнения композиции сервисов формирует расписание, занимающее наименьшее время выполнения. Получившееся в результате расчета расписание можно представить в виде последовательности упорядоченных элементов (Рис. 10), каждый из которых является назначением какого-то сервиса t_i на узел n_j. Для каждого элемента также

определено предполагаемое время начала и завершения работы сервиса – данная информация используется для более точного времени запуска сервиса и более точного контроля его работы.

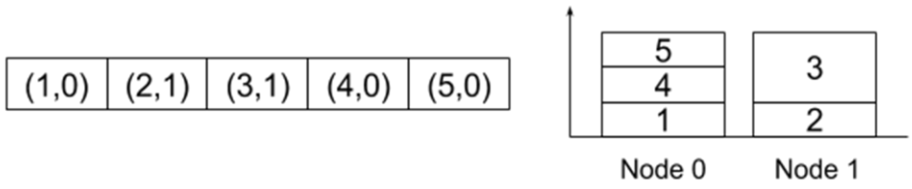

Рис. 10. Представление расписания в виде последовательности назначений

2.3.3 Воспроизведение фактического состояния плана

Для увеличения скорости работы алгоритма при перестроении расписания, то есть когда когда какая-то часть заданий уже начала свое выполнение, используется воспроизведение фактического состояния плана в виде последовательности назначений. Для тех заданий, который уже закончили свою работу, указывается как время начала выполнения, так и время завершения их работы. Для заданий, которые выполняются в настоящее время, указывается время начала их работы и время их завершения, полученное в результате анализа статистических данных длительностей выполнения данного сервиса. Таким образом, при перестроении расписания часть плана воспроизводится на основе текущего состояния среды, то есть планировщик не тратит время на перебор вариантов построения уже фактически выполняющегося плана. Воспроизведение части плана происходит на основании данных о уже выполняющихся заданиях, то есть заданиях, переназначение которых на другие вычислительные узлы невозможно (назовем такие задания фиксированными).

Процесс поиска наилучшего плана графически представлен на Рис. 11. На представленном графе поиска плана вершины $V_{i,j}$ представляют собой назначения задания t_i на вычислительный узел n_j. Вершина V_0 не является назначением какого-

либо задания на узел, она соединяет начальные варианты назначений для более удобной работы с графом. Нахождение плана происходит путем прохода графа в глубину, то есть сначала происходит назначение задания t_1 на вычислительный узел n_1 ($V_{1,1}$), затем назначение задания t_2 на вычислительный узел n_1 ($V_{2,1}$) и т.д. Найденным планом будет называться подграф, представляющий собой цепочку назначений от задания V_0 до задания $V_{M,n}$, где M–количество заданий, $n \in n_0 \dots n_N$, где N–количество вычислительных узлов.

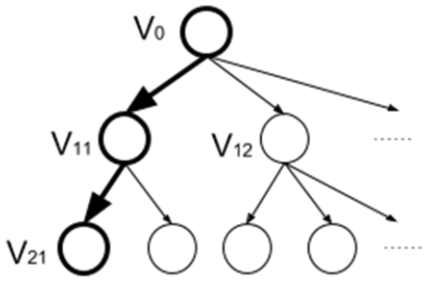

Рис. 11. Процесс поиска наилучшего плана

При перестроении плана возникает ситуация, когда некоторые из заданий уже приступили к выполнению, или уже выполнились. В таком случае процесс перестроения плана можно ускорить, учитывая фиксированные задания. При фиксации заданий уменьшается пространство поиска возможных планов. На Рис. 12 представлен вариант графа поиска плана, когда задания V_0 и $V_{1,1}$ зафиксированы (V_0 зафиксирован всегда, $V_{1,1}$ уже начал своё выполнение). Таким образом, при переборе вариантов построения плана учитывается только та часть графа, которая остается при переборе вариантов после задания $V_{1,1}$ (обозначена пунктиром).

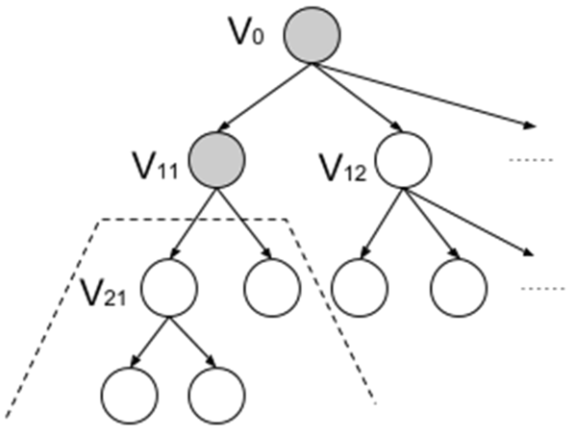

Рис. 12. Граф поиска плана с фиксированными заданиями

Предположение: время, которое будет потрачено на поиск плана с фиксированными заданиями, отражающими реальное состояние вычислительной среды, будет меньше или равно времени, потраченном на поиск плана без учета текущего состояния вычислительной среды.

Доказательство: для доказательства приведённого выше утверждения введена функция $L(V)$, вычисляющая количество вершин графа, которые необходимо будет построить при поиске плана, начиная от текущего задания v. Создание вершины графа занимает некоторую дискретную величину времени t, таким образом, количество времени, потраченное на построение (и на перебор) всех возможных ветвей графа T будет вычисляться как $L(V_0) \cdot t = T$ (перебор всех вариантов построения начинается от начальной вершины V_0). В то же время, величина T определяется как сумма значений $L(V)$ для всех дочерних вершин графа для вершины V_0, то есть $L(V_0) = \sum_{i=0..T} \sum_{j=0..N} L(V_{i,j})$.

Рассмотрим случай, когда одно из заданий начало своё выполнение, например, задание $V_{1,1}$. В таком случае, $L(V_0)$ можно выразить как

$$L(V_0) = \sum_{i=0..T} L(V_{1,j}) + \sum_{i=1..T} \sum_{j=0..N} L(V_{i,j}) = L(V_{1,1}) + \sum_{j=1..N} L(V_{1,j}) + \sum_{i=1..T} \sum_{j=0..N} L(V_{i,j}) = L(V_{1,1}) + 0 + 0 = L(V_{1,1})$$

, причем $\sum_{j=1..N} L(V_{1,j}) = 0$ и $\sum_{i=1..T} \sum_{j=0..N} L(V_{i,j}) = 0$ в силу того, что все варианты назначения задания V_1 на любой другой вычислительный узел, отличный от N_1, а также назначения любых других заданий на любые другие вычислительные узлы сразу после V_0 невозможны, так как назначение $V_{1,1}$ уже зафиксировано. Таким образом, при сравнении $L(V_0)$ без фиксированных (обозначим $L'(V_0)$) и с фиксированными назначениями (обозначим $L''(V_0)$) получается соотношение $L'(V_0) \geq L''(V_0)$, так как $L(V_{1,1}) + \sum_{j=1..N} L(V_{1,j}) + \sum_{i=1..T} \sum_{j=0..N} L(V_{i,j}) \geq L(V_{1,1}) \Rightarrow \sum_{j=1..N} L(V_{1,j}) + \sum_{i=1..T} \sum_{j=0..N} L(V_{i,j}) \geq 0$, что есть истинное высказывание.

Таким образом, воспроизведение подграфа графа поиска плана на основании фактического состояния вычислительной среды в данный момент времени ускоряет процесс нахождения плана, так как пространство перебора вариантов плана сокращается.

В случае перестроения расписания на основе уже частично выполняющегося расписания $P(t,n)$ воспроизведение уже выполняющейся части расписания происходит практически мгновенно, так как порядок следования пар (t,n) задается жестко, следовательно, алгоритм не тратит время на перебор возможных вариантов, которые бы соответствовали уже выполняющейся или выполнившейся части расписания. Как только уже выполняющаяся часть расписания построена, алгоритм продолжает достраивать расписание в стандартном режиме.

При перестроении расписания время выполнения уже выполненных заданий имеет фактическое значение, полученное в процессе выполнения композиции сервисов, таким образом, перестраиваемое расписание более точно.

2.3.4 Динамическое изменение расписания

Учитывая специфику среды распределенных сервисов, алгоритм построения расписаний выполнения заданий должен перестраивать ранее созданное и частично выполняющееся расписание при наступлении следующих событий:

1. Происходит добавление или удаление вычислительного узла – во время выполнения композиции вычислительные узлы могут выходить из строя и возвращаться в рабочее состояние. В случае выхода узла из строя все задания, выполняющиеся на нём, считаются еще не запущенными и снова ставятся в очередь на выполнение, в то же время все уже выполненные на отключившемся узле задания при планировании остаются на нём (при условии, что результаты выполнения заданий остаются доступными, например, файловые результаты можно загрузить по URL). В целях сокращения времени построения расписания при пересчете расписания значение aggregatedCost для вершин графа зависимости сервисов по данным не пересчитывается, так как меняется только состав узлов, на которые необходимо назначить задания. На Рис. 13 показана ситуация, когда узел с идентификатором 0 отказывает во время выполнения на нём задания 2. При перестройке плана задания 2 и 4 переносятся на другой узел, задание 1, так как оно успело завершиться, остается на узле 0 (светло-серым обозначены выполняющиеся в данный момент узлы, серым уже выполнившиеся);

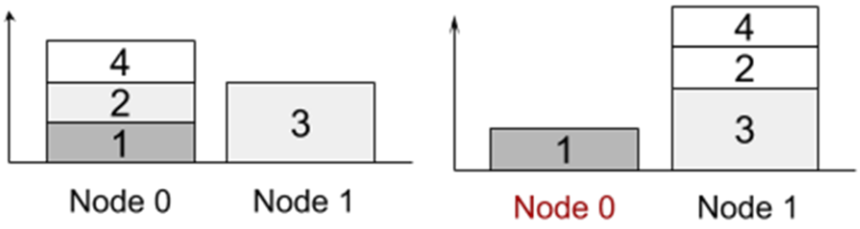

Рис. 13. Отказ узла во время выполнения композиции

2. Происходит добавление задания в DAG – во время выполнения композиции добавляются новые задания по мере интерпретации сценария композиции. Перестроение всего расписания происходит, когда добавляемое задание изменяет прежде рассчитанное время выполнения всего сценария. В случае, когда добавленный сервис при назначении на самый свободный узел не меняет общее время выполнения сценария, перестроение плана не происходит и производится запуск задания на выполнение. На Рис. 14 в первом случае задание 5 при назначении на самый свободный вычислительный узел не изменяет время выполнения расписания, перестроение не происходит. Во втором случае на рис. N после добавления задания 5 время выполнения расписания увеличивается, следовательно, необходимо выполнить перерасчёт;

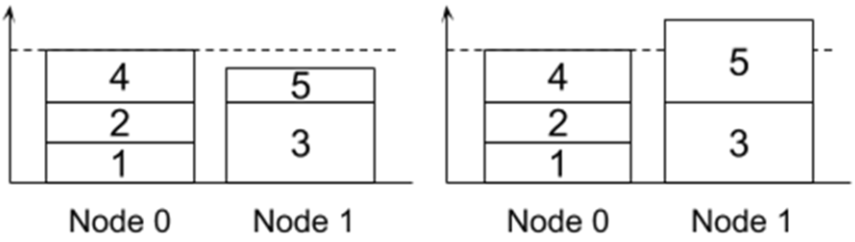

Рис. 14. Добавление задания в план

3. Завершённое задание выполнялось меньше ожидаемого времени – если разница между фактическим и ожидаемым временем меньше значение L, то перестроение не производится, так как перестроение потребует больше времени, нежели получившийся по времени выполнения данного сервиса выигрыш. На Рис. 15 сервис 3 завершил своё выполнение, но раньше, чем предполагалось – пунктирная линия показывает время фактического завершения. Разница между фактическим и ожидаемым временем завершения меньше L, следовательно, перестройка расписания не производится (светло-серым обозначены выполняющиеся в данный момент узлы);

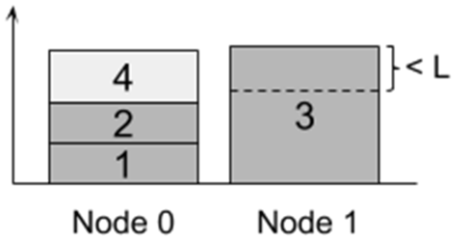

Рис. 15. Добавление задания в план

4. Выполняющееся задание не закончилось в ожидаемый срок – происходит перестроение расписания с предположением, что время рассматриваемого задания становится $t = (1 + d) * c(t) + L$, где d – некоторая константа от 0 до 1. На отображена ситуация, когда задание 3 выполняется дольше, чем планировалось. Левая схема показывает изначальное расписание, правая схема показывает перестроенное расписание с добавленным для задания 3 ожидаемым временем выполнения;

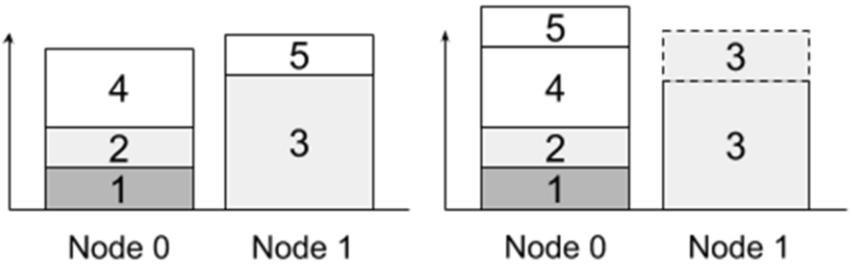

Рис. 16. Задание выполняется дольше, чем ожидалось

5. Выполняющееся задание завершилось с ошибкой – если в текущем расписании на вычислительном узле после рассматриваемого задания назначены какие-либо другие задания, которые уже начали выполнение, то данное задание помечается как невыполненное. Копия задания добавляется в список заданий, ожидающих планирование, но с изменением в составе узлов, на которых оно может выполняться – узел, на котором оно завершилось с ошибкой, не

учитывается при планировании рассматриваемого задания. На Рис. 17 показано изменение структуры расписания в виде последовательности назначений – задание 3 на узле 0 завершилось с ошибкой, поэтому происходит перестройка расписания с учетом необходимости выполнения копии задания 3 на узле, отличном от узла с идентификатором 0;

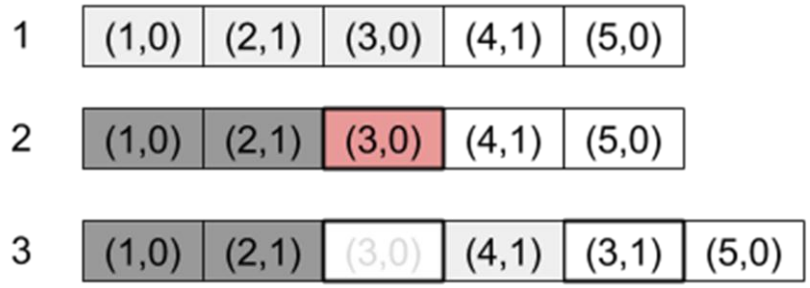

Рис. 17. Завершение задания с ошибкой

2.4 Выводы

В данной главе было предложено использование одного из распространенных процедурных языков программирования для задания композиций сервисов. Рассмотрены как преимущества такого подхода, так и аспекты построения графа зависимостей сервисов по данным (DAG) при использовании основных структур управления – последовательностей, ветвлений и повторов.

В главе был предложен оригинальный метод планирования и выполнения композиций сервисов. Основными его преимуществами является его устойчивость к событиям, происходящим в вычислительной среде, способность учитывать текущее состояние среды при перепланировании в целях составления более точных расписаний за меньшее время. Событиями вычислительной среды, обрабатываемыми при планировании выполнения композиций, являются отключения или включения вычислительных узлов, изменения времени выполнения

сервисов по сравнению с ожидаемым, изменение структуры графа зависимостей вызовов сервисов по данным (DAG), ошибки в выполнении сервисов.

Подход, предполагающий задание композиций сервисов с помощью одного из распространенных языков программирования, позволяет пользователям избежать планирования и параллельного выполнения сервисов. Разработанный алгоритм составления расписаний обеспечивает приемлемое время выполнения сценариев в сочетании с устойчивостью к изменениям в вычислительной среде, характерным для гетерогенных распределенных вычислений.

ГЛАВА 3. СИСТЕМА ДИНАМИЧЕСКОГО ВЫПОЛНЕНИЯ КОМПОЗИЦИЙ СЕРВИСОВ В ГЕТЕРОГЕННОЙ СРЕДЕ

Данная глава рассматривает вопросы программной реализации предложенного метода задания композиций в виде сценариев на языке программирования JavaScript и метода планирования и выполнения композиций сервисов в гетерогенной среде. Реализация разработанных методов требует разработки большого количества вспомогательных модулей, которые также будут рассмотрены в данной главе.

Перед тем, как осуществлять планирование выполнения заданий в композициях сервисов, необходимо разработать и реализовать метод задания композиций, разработать инструментарий, необходимый для хранения и обработки информации о сервисах, а также развернуть виртуальную инфраструктуру, необходимую для развертки и апробации системы динамического выполнения композиций сервисов, обеспечить передачу данных между вызываемыми сервисами.

Общая схема системы динамического выполнения композиций сервисов приведена на Рис. 18. Система состоит из четырёх основных частей

1. Модуль выполнения сценариев – построение DAG в процессе выполнения сценариев, контроль вычислительной среды, планирование выполнения сервисов, распараллеливание обработки больших массивов данных;
2. Каталог сервисов и сценариев – регистрация сервисов в системе, создание сценариев, запуск и хранение результатов выполнения сервисов и сценариев;
3. Система хранения данных (СХД) Геопортала – передача и хранение данных, передаваемых и производимых в результате работы сервисов;
4. Облачная инфраструктура – сервисы, развернутые как на мощностях локального центра обработки данных (набор виртуальных машин,

специализирующихся на выполнении вычислительных сервисов), так и развернутые на вычислительных узлах других организаций.

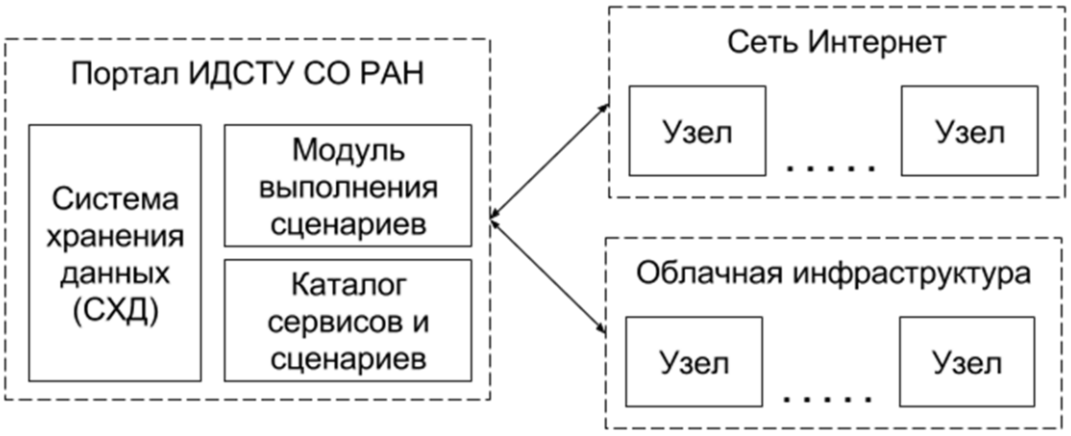

Рис. 18. Схема системы динамического выполнения композиций сервисов

3.1 Модуль выполнения сценариев

3.1.1 Задание композиций сервисовв виде JavaScript сценариев

Для работы с композициями, задаваемыми в виде сценариев на языке программирования JavaScript, в качестве интерпретатора с открытым исходным кодом был выбран JavaScript движок Google V8 разработки компании Google [57]. Данный программный продукт используется как в популярном браузере Chrome (а также большом количестве других браузерах, построенных на основе открытого браузера Chromium), так и в популярном JavaScript окружении NodeJS. Интерпретатор Google V8 на данный момент является самым производительным интерпретатором JavaScript в силу того, что он компилирует JavaScript напрямую в машинный код. Интерпретатор Google V8 написан на языке C++, что делает возможным встраивание в интерпретатор программных продуктов, также написанных на C++. Например, для удобства работы с геоинформационными данными в JavaScript окружение была внедрена библиотека для работы с матрицами.

Задание композиций сервисов пользователем осуществляется с помощью специальной формы, упрощающей и проверяющей корректность ввода данных.

Пример такой формы приведен на Рис. 19. Для создания композиции сервисов пользователю Геопортала необходимо определить название композиции, определить входные и выходные параметры, а также соответствующие для них элементы управления (используется интернет-система ввода и вывода информации «Фарамант» [27]). После определения параметров пользователь может приступить к написанию кода сценария в специальном поле с подсветкой синтаксиса JavaScript. Для удобства пользователю предоставляется список всех зарегистрированных в системе сервисов с входными и выходными параметрами. После сохранения композиция сохраняется в реляционной базе данных. Форма задания композиций сервисов в виде сценариев на языке программирования JavaScript реализована в рамках системы управления контентом (Content Management System, CMS) Calypso, на которой развернуты сервисы Геопортала ИДСТУ СО РАН. CMS Calypso это программный продукт с исходным кодом, написанный на NodeJS и работающий в связке с СУБД MonogDB и PostgreSQL.

Для каждого зарегистрированного в системе сервиса существует уникальная функция, с помощью которой пользователь может осуществить вызов сервиса. Каждая такая функция внутри себя вызывает функцию callService, которая содержит все параметры сервиса (описание входных и выходных параметров, соответствующие элементы управления для параметров, данные поддерживающих сервис вычислительных узлов). При вызове функция callService вызывает C++ функцию callServiceProxy, о которой будет рассказано далее.

Например, функция road_to_grid, использованная в примере сценария выше, задается следующим образом.

```
function road_to_grid(input, mapping) {
     callService([{
          method: 'roadToGrid,
          status: 'true',
          host: 'webexecutor.example.com',
```

```
            port: '8800',
            path: '/cgi-bin/wps/zoo_loader.cgi?',
    }], {
            file: 'file_select',
            value: 'edit'
    }, input, mapping);
}
```

Универсальная функция вызова сервисов позволяет централизовать обработку вызовов сервисов, что необходимо при установлении зависимостей между сервисами, рассмотренного в следующем пункте. При вызове callService() происходит помещение вызова сервиса (задания) в составляемый DAG композиции, что проиллюстрировано на Рис. 20.

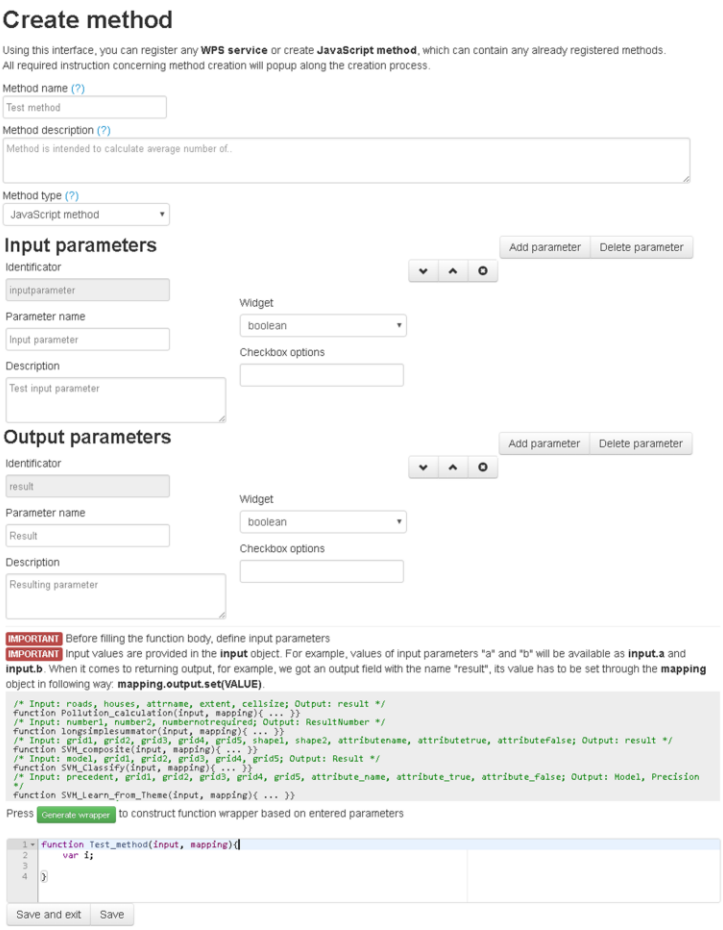

Рис. 19. Форма задания JavaScript сценария

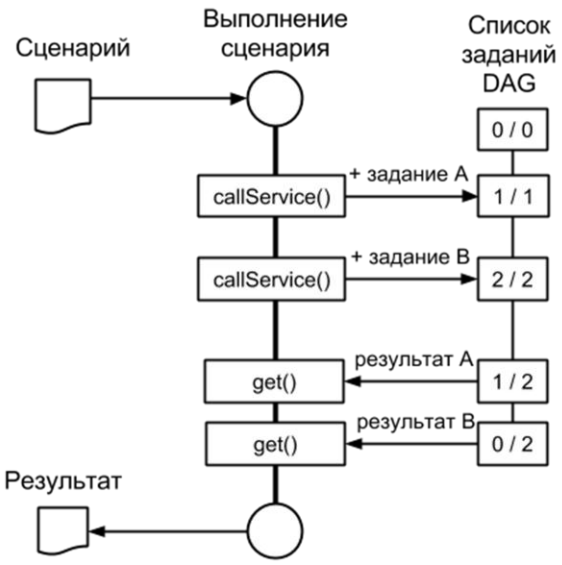

Рис. 20. Помещение вызова сервиса в список заданий DAG

3.1.2 Вложенность сценариев

Часто возникает ситуация, когда необходимо внутри одного сценария вызвать другой сценарий распределённых сервисов. Так как композиции сервисов, задаваемых в виде сценариев на языке JavaScript, представляют собой программы, их можно свободно вызывать в теле других сценариев. Передача параметров в сценарий, а также получение результатов работы вложенного сценария осуществляется аналогично передаче и получению данных в вызываемые сервисы посредством специального типа объектов *`ValueStore`*.

Например, существует сценарий `test_outer` (входной параметр *`input1`*, выходной параметр *`result1`*) и сценарий `test_inner` (входной параметр *`input2`*, выходной параметр *`result2`*). Пример вызова сценария `test_inner` внутри сценария `test_outer` приведен ниже.

```
function test_outer(input, mapping) {

    var innerResult = new ValueStore();
```

```
    test_inner({input2: input.param}, {result2: innerResult })
    mapping.result1.set(innerResult.get());
}
```

Стоит отметить, что в целях обеспечения замкнутости все входные параметры, передаваемые в функции-обертки или сценарии, имеют тип *ValueStore*. В противном случае при вызове функций-оберток или сценариев внутри других сценариев в качестве входных параметров могли бы передаваться как объекты *ValueStore*, так и переменные стандартных типов данных, что делало бы необходимым наличие кода, работающего как с *ValueStore*, так и с обычными типами данных. Обязательное использование *ValueStore* при передаче входных параметров унифицирует работу с параметрами внутри сценария и уменьшает шанс потенциальной ошибки в обрабатывающем коде.

3.1.3 Установление зависимостей по данным между вызовами сервисов

Как было упомянуто в предыдущей главе, функции-обертки сервисов внутри себя вызывают функцию *callService*, которая получает на вход непосредственные значения входных параметров, описание самого метода и вычислительного узла (узлов), на котором он может быть запущен. Для того, чтобы вызванная функция-обертка могла сразу завершить свою работу и выполнение сценария могло перейти к следующей команде, *callService* возвращает пустое значение. Особенность *callService* заключается в том, что она вызывает соответствующую ей функцию в C++, которая регистрирует вызов сервиса и помещает его в список заданий DAG.

Списком заданий DAG называет структура, реализованная в виде связного списка, который содержит в себе все вызовы сервисов, зарегистрированные с помощью вызова *callService* внутри функций-оберток сервисов. Каждый элемент списка содержит в себе описание вычислительных узлов, на которых может

выполняться сервис, его описание и описание его входных и выходных параметров, а также статус задания. Данный список используется для отложенного вызова сервисов, причем решение о запуске принимается планировщиком, о котором будет рассказано позже.

Таким образом, при вызове функций-оберток сервисов происходит регистрация задания в списке заданий DAG, и функция-обертка завершает своё выполнение. Так как функции-обертки завершают свою работу практически мгновенно, блокирования выполнения сценария в результате ожидания результата работы функции-обертки не происходит. Непосредственный запуск сервиса происходит позже на основании решения планировщика, механизм работы которого будет рассказан позднее.

Как было рассмотрено ранее, наполнение DAG выполняемого сценария происходит по мере вызова соответствующих функций-оберток. Однако, при наполнении DAG необходимо учитывать не только узлы DAG (задания), но и его ребра (зависимости между вызовами сервисов по данным). Для этих целей реализован специальный тип объектов *`ValueStore`*, который служит для определения зависимостей между сервисами и для хранения и работы с результатами выполнения сервисов, которые помещаются в объект по мере готовности заданий.

Описание класса в виде UML-диаграммы представлено на Рис. 21.

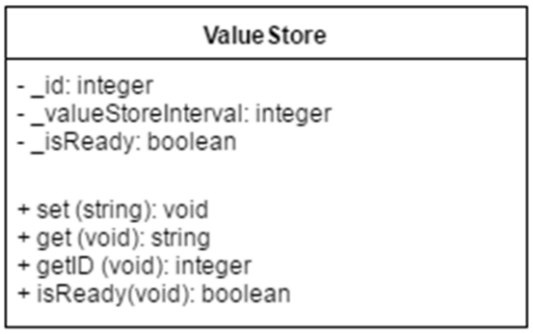

Рис. 21. Класс ValueStore

Существует механизм определения зависимости между заданиями, который выполняется при регистрации задания в глобальной очереди. Для того, чтобы определить, от каких заданий зависит регистрируемое задание, для каждого входного параметра вызывается метод *getID*, который возвращает идентификатор объекта (как было сказано ранее, все входные параметры функций-оберток и сценариев являются объектами класса *ValueStore* и имеют уникальные идентификаторы). Зная идентификатор объекта, производится проверка всех заданий, находящихся в очереди на предмет наличия среди них выходных параметров типа *ValueStore* с таким же идентификатором. Если такое задание находится, то зависимость найдена.

Рассмотрим приведенный выше сценарий.

```
function test_scenario(input) {
   var vs1 = new ValueStore();
   var vs2 = new ValueStore();
   road_to_grid({file: input.road, value: input.roadValue},
{res: vs1});
   point_to_grid({file: input.point, value: input.pointValue},
{res: vs2});
    raster_sum({file1: vs1, file2: vs2});
}
```

В сценарии участвует два объекта *ValueStore*, *vs1* и *vs2*. По мере выполнения сценария происходят следующие действия, критичные для определения зависимостей между сервисами:

1. Начинает выполняться *road_to_grid*. Среди входных параметров нет таких объектов *ValueStore*, которые бы были выходными параметрами вызовов других сервисов, значит, этот вызов сервиса является в графе начальным (корневым). Результат работы сервиса через какое-то время будет сохранен в объект *vs1*;
2. Начинает выполняться *point_to_grid*. Среди входных параметров нет таких объектов *ValueStore*, которые бы были выходными параметрами вызовов других сервисов, значит, этот вызов сервиса является в графе начальным (корневым). Результат работы сервиса через какое-то время будет сохранен в объект *vs2*;
3. Начинает выполняться *raster_sum*. Среди входных параметров есть два объекта *vs1* и *vs2*, которые являются контейнерами для результатов выполнения *road_to_grid* и *point_to_grid*, значит, *raster_sum* будет готов выполняться только тогда, когда оба контейнера будут заполнены.

Таким образом, анализируя передачу объектов класса *ValueStore* между вызовами сервисов можно определить зависимости между этими вызовами, то есть по мере выполнения сценария, приведенного выше, строится приведенный на рис. N граф зависимостей вызовов сервисов по данным (DAG). Зависимость между двумя сервисами можно определить как наличие некоего объекта *ValueStore*, который одновременно является входным и выходным параметром у двух вызовов сервисов. Идентификация объекта происходит на основании внутреннего поля id класса *ValueStore*.

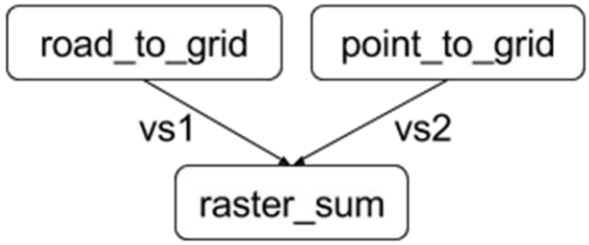

Рис. N. Граф зависимостей вызовов сервисов по данным

3.1.4 Неблокирующий вызов функций-оберток

Как было сказано ранее, выполнение функций-оберток не блокирует выполнение сценария сервисов. В таком случае, получение результатов работы сервисов как результат выполнения JavaScript функции невозможно, так как на момент завершения функции результат еще не готов. Для получения и обработки результатов выполнения сервисов также служат объекта класса *ValueStore*.

Представим, что существует некий сервис *road_to_grid*, получающий на вход некоторый файл, задающий в векторном виде дорожную сеть на определенной местности, а также среднюю величину выбросов от автомобилей на километр дороги. Результатом работы сервиса будет некая регулярная сетка, каждая ячейка которого содержит суммарное загрязнение от дорог в данной ячейке, рассчитанное на основании входных параметров. Абстрагируясь от способа запуска и контроля выполнения вызова сервиса (это будет рассмотрено позже), асинхронно вызовем сервис *road_to_grid* в примерном сценарии, приведенном ниже. Для наглядности введем аналогичный сервис *point_to_grid*, отличающийся от *road_to_grid* тем, что расчет ведется не относительно линейных объектов (дорог), а относительно точечных (например, тепло электростанции). Также введем сервис raster_sum, объединяющий результаты работы сервисов *road_to_grid* и *point_to_grid*.

```
function test_scenario(input) {
    road_to_grid({file: input.road, value: input.roadValue});
```

```
    point_to_grid({file: input.point, value: input.pointValue});
    raster_sum();
}
```

Так как сервисы *road_to_grid* и *point_to_grid* вызываются асинхронно, выполнение сценария не останавливается и не ждет их выполнения, а идет дальше. Но сервис *raster_sum* требует на вход результаты работы предыдущих двух сервисов, так им образом, необходимо осуществлять получение результаты работы сервисов в асинхронном режиме. Для этих целей используются объекты *ValueStore*. Для того, чтобы результат работы какого-либо сервиса был помещен в созданный объект *ValueStore*, необходимо передать данный объект с соответствующим ключом во втором параметре вызываемого сервиса.

Таким образом, приведенный выше сценарий записывается в следующем виде

```
function test_scenario(input) {
    var vs1 = new ValueStore();
    var vs2 = new ValueStore();
    road_to_grid({file: input.road, value: input.roadValue},
{res: vs1});
    point_to_grid({file: input.point, value: input.pointValue},
{res: vs2});
    raster_sum({file1: vs1, file2: vs2});
}
```

Как видно из примера, сервисы *road_to_grid* и *point_to_grid* при вызове получают дополнительный второй входной параметр, указывающий в какой объект *ValueStore* поместить результат работы. В сервисе *raster_sum*, которому требуются результаты работы предыдущих двух сервисов, получает на вход созданные ранее контейнеры *ValueStore*, и как только оба из них будут заполнены, сервис *raster_sum* начнет выполняться.

Стоит отметить, что публичный метод get класса *ValueStore* является блокирующим, то есть во время его вызова выполнение сценария останавливается, пока контейнер не получит данные. Для корректного использования объектов *ValueStore* в целях использования всех возможностей задания сценариев с помощью JavaScript существуют следующие правила:

1. Вызывать метод *get* для получения содержимого контейнера *ValueStore* только когда это необходимо для управления потоком выполнения сценария, как в приведенном ниже примере – в зависимости от результата работы сервиса *check_vector_type* вызывается либо сервис *point_to_grid*, либо *line_to_grid*

```
function test_scenario(input) {
   var res = new ValueStore(), calcRes = new ValueStore();
  check_vector_type({ ... }, {result: res});
  if (res.get() === “LINE”) {
    line_to_grid({ ... }, {result: calcRes});
  } else if (res.get() === “POINT”) {
    point_to_grid({ ... }, { result: calcRes});
  }
}
```

2. При передаче объекта *ValueStore* в качестве входного параметра для какого-либо сервиса передача должна осуществляться без вызова блокирующего метода *get()*. В приведённом ниже примере вызову сервиса *point_to_grid* передаётся не сам объект *res*, а результат работы его метода *get()*, то есть выполнение сценария остановится до тех пор, пока методом *get()* не будет возвращено значение.

```
function test_scenario(input) {
var vectorResult = new ValueStore();
```

```
        generate_vector({ ... }, {result: vectorResult});
        point_to_grid({ vector: vectorResult .get() }, { ... });
    }
```

Стоит отметить, что один объект *ValueStore* может передаваться любому количеству других сервисов, но помещать в него данные может только один вызов сервиса.

3.1.5 Работа с DAG

DAG заполняется по мере выполнения функций-оберток сервисов внутри сценария. DAG реализован в виде связного списка структур, для каждой из которых определены ссылки на зависящие и зависимые элементы DAG. Каждый элемент списка хранит в себе идентификатор и статус, входные параметры вызываемого сервиса и результаты его работы, описание элементов управления для входных и выходных параметров, данные о характеристиках вычислительных узлов, на которых сервис может выполняться, другую сервисную информацию. Статус элемента списка может принимать четыре значения – «ожидает выполнения», «выполняется», «успешно завершился», «завершился с ошибкой». Входные и выходные параметры хранятся с помощью специальной структуры, содержащей в себе идентификатор входного параметра, значение параметра, а также идентификатор элемента управления, соответствующего данному параметру. Характеристики вычислительных узлов, на которых может выполняться сервис, хранятся в виде массива структур, каждый элемент которой содержит IP адрес вычислительного узла, порт и путь до запрашиваемого веб-сервиса. Дополнительно каждый вычислительный узел характеризуется характеристиками сетевого канала, подключающего узел к сети Интернет, объёмом оперативной памяти, установленной на вычислительном узле и характеристиками центрального процессора. Также в каждом элементе списка планировщик хранит информацию о предполагаемом

времени и назначенном узле для запуска данного сервиса, при условии, что было произведено планирование, и рассматриваемый сервис был назначен на какой-либо вычислительный узел.

Узлы DAG постоянно обрабатываются, специальный обработчик проходит по всем элементам списка через определённый отрезок времени. Постоянный контроль списка необходим по следующим причинам:

1. Некоторые стандарты интерфейсов веб-сервисов, например, стандарт Web Processing Service (WPS) имеют поддержку неограниченного времени выполнения сервисов. Таким образом, для некоторых сервисов, для которых заранее известно, что они могут выполняться неограниченное время, выполняется проверка их статуса. Если сервис выполнился, то происходит получение и обработка результата его работы, его статус становится «выполняется». Если сервис завершился с ошибкой, то его статус становится «завершился с ошибкой»;
2. Непосредственные запросы на выполнение распределённых сервисов реализованы таким образом, что сами запросы происходят асинхронно, то есть элементы очереди обновляются, как только возвращается соответствующий ответ от удалённого сервера. Таким образом, постоянная проверка очереди позволяет своевременно обрабатывать результаты выполнения запросов к удалённым серверам;
3. Обработка узлов DAG производится при наступлении событий распределенной гетерогенной среды. Перепланирование осуществляется только в случае определенных событий, описанных в п. 2.3.4. Так как в один момент времени происходит только один процесс перепланирования, причем время, тратящееся на перепланирование, ограничено заранее заданным значением, не превышающим период обработки глобальной очереди заданий,

составляемые расписания учитывают текущее состояние среды и не производится перестроений расписаний, результаты которых не учитываются.

На Рис. 22 представлен механизм взаимодействия обработчика со списком заданий DAG. Обработчик реализован в виде отдельного потока выполнения программы, который запускается в рамках диспетчера выполнения сценариев распределённых веб-сервисов. Поток завершает свою работу при условии, что сценарий завершил своё выполнения, и что не осталось ни одного сервиса, который бы выполнялся в данный момент времени. Осуществление запросов к распределённым веб-сервисам производится с помощью библиотеки с открытым исходным кодом cURL, реализованной под языка программирования C++. Выбор данной библиотеки был сделан на основе того, что cURL предоставляет возможность совершения асинхронных запросов к удалённым сервисам, т. е. диспетчер выполнения сценариев не ожидает выполнения каждого запроса (обычно максимальное время выполнения HTTP запроса ограничено 60 секундами). Для каждого асинхронного запроса определяется некая функция, которая вызывается по мере завершения запроса или в случае его ошибки. При успешном завершении запроса результат запроса обрабатывается и помещается в соответствующий элемент списка заданий, в случае возникшей ошибки соответствующий элемент списка также обновляется в соответствии со спецификой возникшей ошибки.

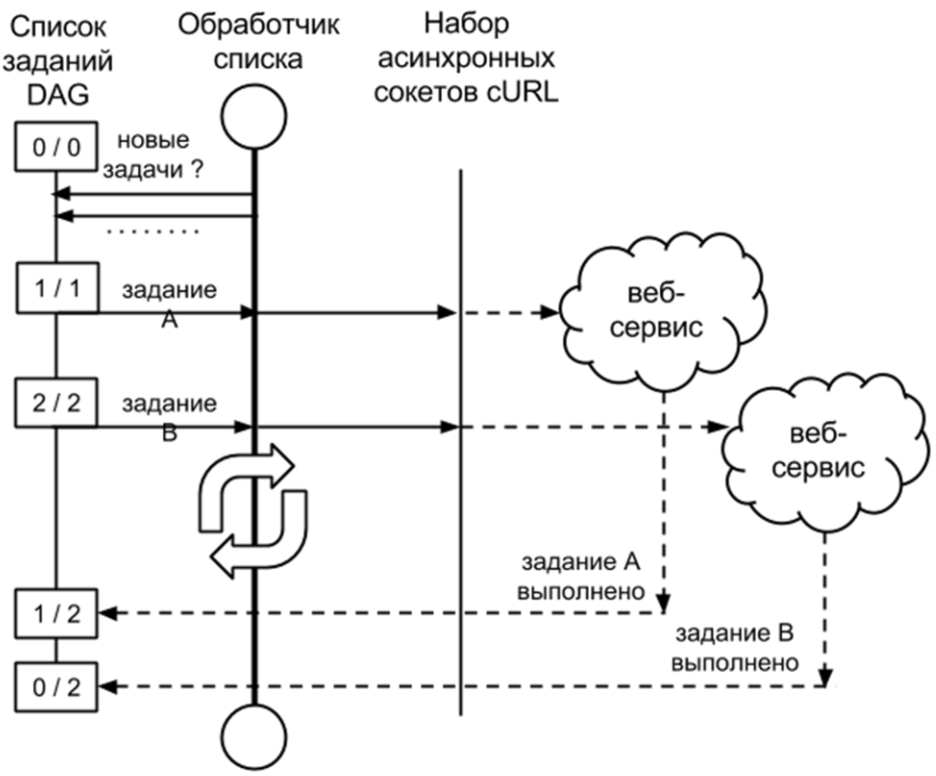

Рис. 22. Схема периодической обработки списка заданий DAG

3.1.6 Проверка выполнимости сценариев сервисов

В рамках сервис-ориентированного подхода сервисы могут находиться в любой точке сети Интернет, причем постоянная доступность сервисов не гарантируется. Поэтому при работе со сценариями может возникать проблема, что сценарии, включающие в себя вызовы большого количества сервисов и других сценариев, не могут завершить выполнение в силу недоступности каких-либо вычислительных узлов, реализующих определенные сервисы. При этом выполнение такой композиции может привести к значительным затратам вычислительных ресурсов и потери времени пользователя на подготовку данных и запуск выполнения композиции. Поэтому является актуальной задача проверки выполнимости композиции сервисов в рамках текущего состояния вычислительной среды. Решение данной задачи является актуальной как для администраторов системы, так и для пользователей. Администраторы могут определить вычислительные узлы,

необходимые для работы композиций сервисов, и обеспечить их работу. Пользователи нуждаются в актуальном каталоге рабочих сценариев как перед их непосредственным выполнением, так и в случае редактирования какого-либо сервиса, впоследствии вызываемого в сценариях – необходимо проверять, повлияет ли изменение настроек сервиса на выполнимость зависимых сценариев (проверка работоспособности вводимых пользователем вычислительных узлов). Таким образом, возникает задача проверки выполнимости композиции сервисов в рамках текущей распределенной гетерогенной среды перед непосредственным запуском композиции.

Для решения задачи проверки выполнимости композиции сервисов в рамках текущей распределенной гетерогенной среды предлагается подход, предполагающий анализ JavaScript кода сценариев. Общий алгоритм проверки выполнимости представлен ниже:

1. Извлечение абстрактного синтаксического дерева (Abstract syntax tree, AST) из JavaScript сценария. На Рис. 23 изображен пример сценария и соответствующего ему AST. Извлечение AST возможно для любого JavaScript сценария, в том числе и для случаев вложенных сценариев и рекурсивных вызовов;

```
function serviceA() {
  return 1;
}
function serviceB() {
  return 2;
}
function composition() {
  var aResult = serviceA();
  if (aResult > 0) {
    serviceB();
  }
}

composition();
```

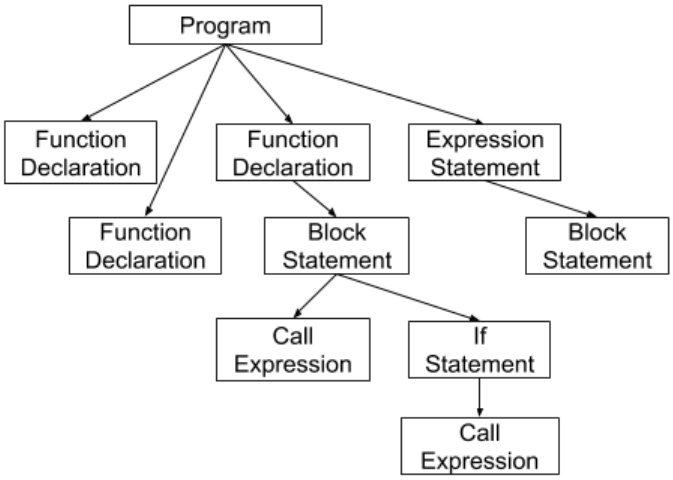

Рис. 23. Сценарий и соответствующее ему AST

2. На основании полученного AST строится список сервисов, которые вызываются внутри сценария (учитываются как управляющие конструкции языка, так и вложенные вызовы других сценариев);
3. Для каждого из полученных сервисов из каталога сервисов Геопортала получается список вычислительных узлов (для каждого узла задан сетевой адрес и адрес службы сервисов). Далее каждый из узлов проверяется на доступность путем вызова службы сервисов (например, в случае службы WPS сервисов происходит вызов метода `GetCapabilities`). Стоит отметить, что недоступность одного или нескольких вычислительных узлов не приводит к невыполнимости сценария, так как некоторые сервисы могут выполняться на нескольких вычислительных узлах, то есть для выполнимости сценария необходимо и достаточно следующее правило – для каждого вызываемого сервиса должен быть доступен хотя бы один вычислительный узел. На Рис. 24 изображен процесс проверки доступности вычислительных узлов на основании построенного AST – при прохождении дерева AST производится проверка узлов типа `CallExpression`, соответствующих вызовам функций (если функции соответствуют сервисам);

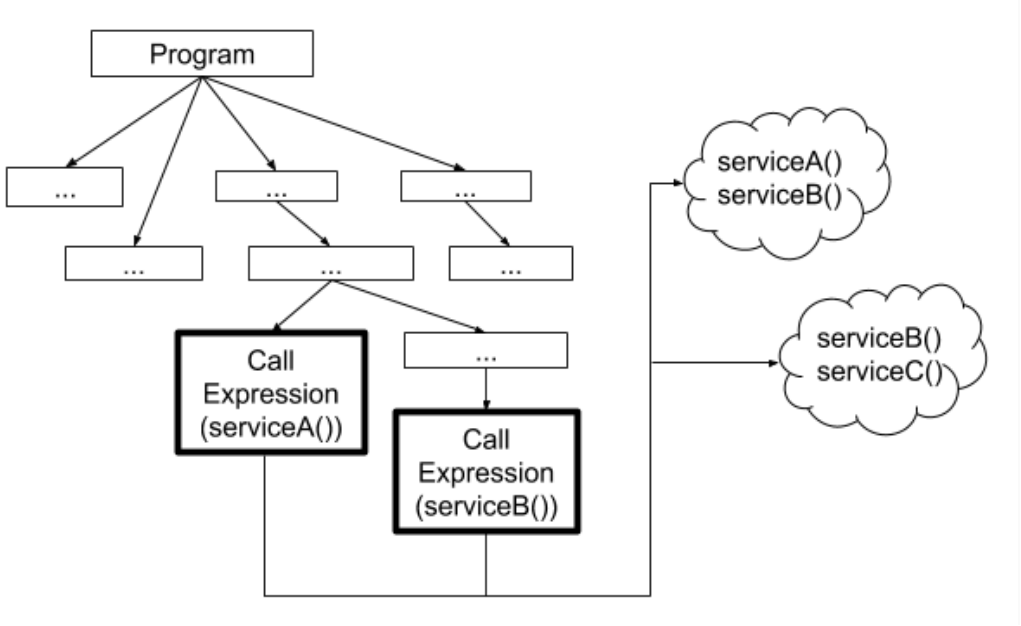

Рис. 24. Проверка доступности вычислительных узлов

4. В случае невозможности выполнения сценария пользователю возвращается сообщение об ошибке и предоставляется список сервисов, для которых не обнаружено рабочих вычислительных узлов, в ином случае производится вывод о готовности среды для выполнения композиции.

Описанный подход к проверке выполнимости композиции сервисов в рамках текущей распределенной гетерогенной среды позволяет пользователям более эффективно работать со сценариями, а администраторам системы отслеживать выполнимость созданных пользователями сценариев.

3.1.7 Контроль среды и перепланирование

Как уже было сказано, при определённых ситуациях, происходящих в среде распределённых сервисов, происходит перепланирование распределения вызова сервисов (см. п. 2.3.4).

Для контроля состояния среды на протяжении всего времени выполнения сценария ведётся список вычислительных узлов, которые поддерживают

выполняющиеся сервисы. Каждый элемент списка узлов содержит информацию о занятости узла и его статусе («активен» или «неактивен»). Данная информация используется при построении нового расписания.

Обновление статуса узлов происходит при обнаружении ошибки при соединении с вычислительным узлом, то есть когда не удалось выполнить запрос на выполнение какого-либо сервиса. Вычислительный узел, в случае если он отмечен как неактивный, считается готовым к работе через некоторую константу T, определённую в настройках системы.

3.1.8 Диспетчер выполнения сценариев

Модуль выполнения сценариев реализован в виде отдельного приложения (диспетчера выполнения композиций сервисов), работающего под управлением Геопортала ИДСТУ СО РАН. Геопортал берет на себя функции хранения информации об участвующих в композиции сервисах – их типах, сетевом расположении, характеристиках вычислительных узлов, на которых они могут располагаться. Сервисы регистрируются на Геопортале пользователями. Геопортал предоставляет удобный интерфейс для ввода параметров запускаемых сценариев, среди них могут быть как текстовые данные, так и файлы из СХД Геопортала, результаты выгрузки данных из реляционных таблиц, хранимых на Геопортале, и другие типы данных. При запуске диспетчера Геопортал анализирует, какие из зарегистрированных сервисов участвуют в композиции, и включает их в код композиции в виде оберток на языке JavaScript, таким образом, любой сервис в сценарии может быть вызван всего одной функцией. Готовая для запуска композиция, с объявленными участвующими сервисами, а также самим кодом композиции, передается в диспетчер.

Принятый на вход диспетчером скрипт компилируется с помощью встроенного JavaScript движка Google V8 и запускается на выполнение. По мере

выполнения скрипта вызываются функции-обертки соответствующих сервисов, соответствующие вызовы сервисов помещаются в список заданий DAG диспетчера. Стоит отметить, что вызов сервиса уникален, в то время как вызовов самого сервиса в одном сценарии может быть любое количество, то есть вызов сервиса определяется как сочетание входных параметров, введенных пользователем или сгенерированных сценарием, и описания самого сервиса.

С момента запуска внутри приложения диспетчера запускается отдельный поток выполнения, который с определенной периодичностью опрашивает список заданий DAG на предмет изменения состава или статуса зарегистрированных в нем сервисов. При наступлении определенных событий, рассмотренных в п.4, происходит перестроение расписания с учетом уже выполняющихся сервисов. Для перестроения расписания необходимо также знать время выполнения сервисов на определенных узлах, а также состояние вычислительных узлов, выполняющих хотя бы один сервис из участвующих в композиции.

Также диспетчер хранит информацию о состоянии вычислительных узлов, на которых выполняются сервисы, с помощью модуля контроля состояния среды распределённых сервисов. При каждом вызове сервиса или проверке его состояния в случае ошибки во время передачи данных модуль контроля обрабатывает ситуацию и при следующем планировании составляет корректный список вычислительных узлов, учитывающие возникшие в процессе выполнения композиции сбои в работе среды распределённых сервисов.

Опрос состояния выполняемых длительных сервисов осуществляется с определенной периодичностью в отдельных нитях исполнения. В случае завершения или ошибки выполняемого сервиса происходит обновление соответствующего элемента в глобальной очереди заданий.

Модуль статистики служит для хранения данных о времени выполнения сервисов на определенных узлах. Модуль отслеживает вызовы сервисов по двум

параметрам: сетевой адрес сервиса и идентификатор сервиса. Отслеживание вызовов сервисов только по идентификаторам (обычно представляет собой текстовую строку с названием сервиса) не ведется в силу того, что на разных вычислительных узлах сервис может иметь разную скорость выполнения. Для каждой такой записи хранится история вызовов с указанием времени выполнения.

Перед тем, как составить расписание с вызовом определенного сервиса, нужно получить его ожидаемое время выполнения. Для получения ожидаемого времени модуль статистики сверяется, есть ли для такого сочетания сетевого адреса и названия сервиса записи. Если записи есть, то ожидаемое время выполнения определяется как среднее время последних тридцати процентов записей. Если же записей нет, то берется среднее от среднего времени выполнения других сервисов, выполняемых на этом вычислительном узле. Если для данного вычислительного узла нет записей, то тогда делается предположение относительно его длительности на основании заранее заданных значений, зависящих от типа сервиса (является ли это длительным сервисом или нет).

3.1.9 Автоматическое распараллеливание данных в рамках модели MapReduce

При выполнении распределенных сервисов часто возникает ситуация, когда сервисы выполняются очень большое время в силу большого объема обрабатываемых данных. Нередко данные, передаваемые сервисам, возможно разделить на некоторое количество частей и передать на обработку некоторому количеству копий требуемого сервиса. Такой подход к обработке данных описывается в рамках модели MapReduce [58], где map это операция разделения данных, а reduce – операция сборки результатов работы копий сервиса.

Учитывая особенности распределённой среды веб-сервисов, а также специфику форматов данных обрабатываемых данных в среде геоинформационных сервисов, использование модели MapReduce повысит скорость выполнения

распределенных сервисов. Например, при обработке растровых данных исходное изображение можно разделить на более мелкие части и по отдельности обработать без ущерба результату.

Для автоматизации и упрощения выполнения сервисов в рамках программной модели MapReduce были введены спецификации – текстовые описания операций map и reduce, определяющих минимальный и максимальный размер ячейки, на которые делится исходный массив данных, размер перекрытия ячейки, и другие параметры [59,60]. При регистрации сервиса пользователь выбирает какую-либо из предустановленных спецификаций для конкретного сервиса, либо определяет свою собственную.

Спецификации могут определять жестко заданный размер ячейки массива данных, но чаще всего используется диапазон возможных значений размера ячейки. Механизм выполнения распределенных сервисов в рамках программной модели MapReduce интегрирован с диспетчером выполнения сценариев и планировщиком, что позволяет выполнять операции map и reduce над данными с учетом состояния распределенной среды – фактической и запланированной загруженности и доступности вычислительных узлов. Например, при заданном диапазоне возможных размеров ячейки на основании данных, полученных от планировщика, диспетчер подбирает размер ячейки таким образом, чтобы вычислительные узлы, которые могут выполнять требуемый сервис, были равномерно загружены, тем самым скорость выполнения сервиса была бы увеличена.

Говоря о непосредственной реализации операций map и reduce, на данный момент реализованы операции для растровых данных. Обработчик операции Map включает реализованные функции чтения спецификаций, на их основе формируются параметры для распределения растровых данных между вычислительными узлами. Для разделения данных формируются параметры запуска утилиты

GDALTRANSLATE, предназначенной для конвертации растров. Обработчик операции Reduce включает реализованные функции чтения спецификаций, реализованные обработчики сбора данных. Обработчики данных реализует в себе стандартные функции обработки конфликтных ситуаций возникающих в процессе сбора данных. Конфликтные ситуации возникают, например, при сборе частей мозаики растра в одно целое. К таким ситуациям можно отнести поступление повторяющихся или неоднозначных данных. В этом случае обработчик применяет к ним операцию, указанную в спецификации. В текущей версии доступны следующие операции: *max* – установить максимальное значение из двух перекрывающихся пикселей, *min* – установить минимальное значение из двух перекрывающихся пикселей, *avg* – вычислить среднее значение из двух перекрывающихся пикселей.

Для использования механизма увеличения скорости выполнения распределенных сервисов в рамках модели MapReduce достаточно один раз указать параметры разбиения и склейки при регистрации сервиса на Геопортале, в дальнейшем диспетчер будет самостоятельно принимать решение по способу разбиения массива входных данных сервиса на основании предоставленной спецификации и текущего состояния распределенной среды. Также стоит отметить, что использование программной модели MapReduce при выполнении распределенных сервисов в рамках Геопортала не требует изменения самих вычислительных веб-сервисов. Иллюстрация обработки массива данных в рамках модели MapReduce изображена на Рис. 25.

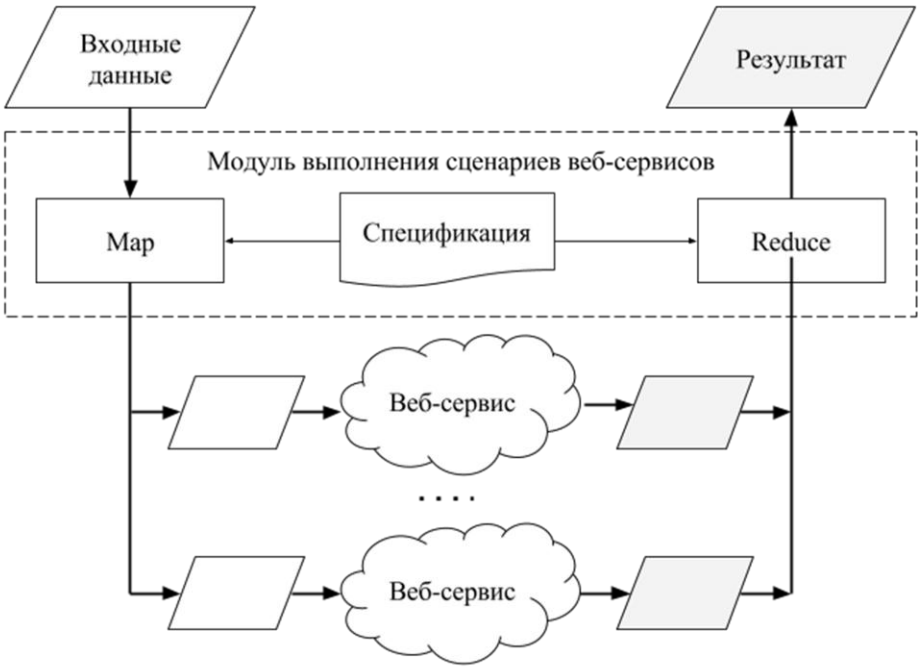

Рис. 25. Выполнение распределённых сервисов в рамках программной модели MapReduce

3.2 Каталог сервисов и сценариев

Составление композиций сервисов (разработка сценариев) и дальнейшее выполнение таких композиций происходит, ориентируясь на информацию об зарегистрированных в системе сервисах. Информацию о сервисах предоставляет каталог сервисов. Помимо хранения информации о сервисах, каталог предоставляет интерфейс для редактирования композиций сервисов, задаваемых в виде сценариев на языке программирования JavaScript, осуществляет публикацию композиций сервисов. Другой важной функцией каталога является хранение информации о состоявшихся запусках сервисов и сценариев.

3.2.1 Регистрация сервисов

Для удобства и корректности заполнения информации о регистрируемых сервисах была разработана специальная форма ввода данных, представленная на Рис. 26. Каждый сервис в системе задается такими параметрами, как имя, тип сервиса (WPS, REST и т.д.), список вычислительных узлов, на которых данный сервис может выполняться. Также пользователь определяет, может ли сервис

выполняться длительное время, то есть больше 60 секунд. Если сервис помечен как длительный, то при работе с ним система будет обращаться с ним как с длительным, если это позволяет стандарт интерфейса сервиса. Например, для стандарта WPS возможно длительное выполнение сервиса, в таком случае при вызове сервиса необходимо добавлять в запрос определенные параметры – в ответ на запрос о запуске WPS-сервиса возвращается URL XML-файла, в котором по мере выполнения сервиса будет отображаться процент выполнения сервиса, а по завершению WPS-сервиса – результат работы.

По мере ввода параметров вычислительных узлов с данным сервисом (вычислительный узел идентифицируется по своему сетевому адресу) в зависимости от типа сервиса система автоматически опрашивает первый из введенных вычислительных узлов на предмет, какие сервисы данный узел предоставляет (получение информации о поддерживаемых сервисах осуществляется с помощью специальных команд соответствующих стандартов, например, GetCapabilities у стандарта WPS). Далее пользователь выбирает один из предоставленных сервисов, и система загружает описание параметров сервиса и отображает как входные, так и выходные параметры. Для каждого входного параметра пользователь выбирает элемент управления (система элементов управления является частью Геопортала ИДСТУ СО РАН), с помощью которого будут вводиться параметры для данного сервиса.

При нажатии на кнопку сохранения вся информация, введенная на интерфейсе, передается на сервер. На сервере происходит создание функции-обертки для регистрируемого сервиса, которая будет использоваться впоследствии для вызова сервиса в композициях. Каждая функция обертки содержит в себе вызов общей для сервисов функции `callService()`, в которую передаются параметры, непосредственно введенные пользователем или пришедшие внутри сценария, а также сервисная информация, хранимая в базе данных. Например, для сервиса

test_service, развернутого на вычислительном узле с сетевым адресом 192.168.10.10 функция-обертка будет выглядеть таким образом:

```
test_service(input, mapping, specification) {
    callService(input, mapping, [{“host”: “192.168.10.10”,...
}]);
}
```

`callService` является единой точкой регистрации вызова сервиса внутри диспетчера выполнения композиций сервисов в гетерогенной среде. После создания тела функции-обертки вся информация о зарегистрированном сервисе сохраняется в реляционной базе данных.

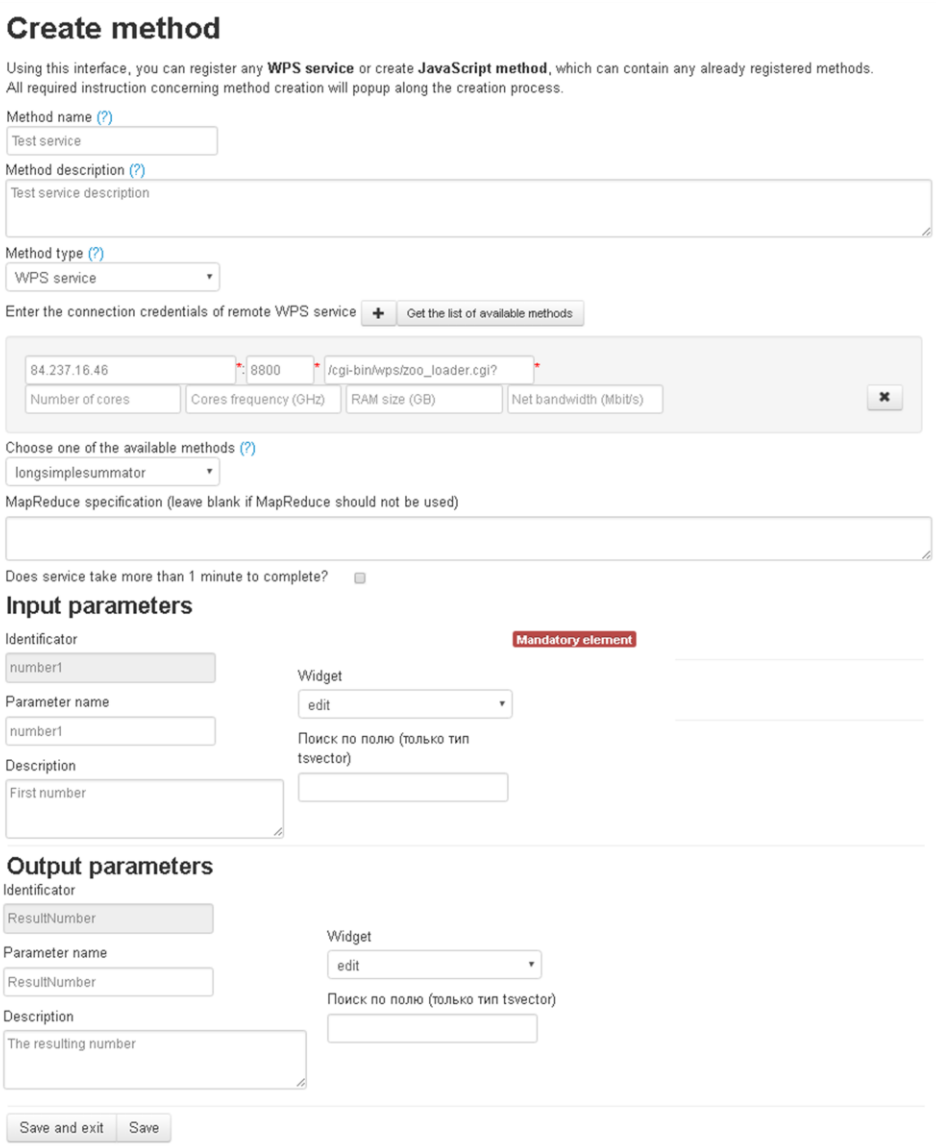

Рис. 26. Форма регистрации сервиса

Программно каталог реализован на основе CMS Calypso, на которой развернуты сервисы Геопортала ИДСТУ СО РАН. CMS Calypso это программный продукт с исходным кодом, написанный на NodeJS и работающий в связке с СУБД MonogDB и PostgreSQL.

На данный момент пользователям системы динамического выполнения композиций сервисов в гетерогенной среде доступны следующие веб-сервисы,

опубликованные с помощью стандарта WPS и зарегистрированные в каталоге сервисов:

1) *summator* и *longsummator* – сервисы, использующиеся для тестирования корректности работы WPS службы. Оба сервиса выполняют сложение чисел, однако последний сервис имеет искусственно увеличенное время выполнения (для тестирования длительного выполнения сервисов, опубликованных в WPS службе);
2) *vector2grid* – принимает на вход точечные векторные данные в формате shp и производит конвертацию переданных данных в растровый формат GeoTIFF. Размер ячейки результирующего файла, координаты области конвертации также передаются в качестве входных параметров сервиса;
3) *road2grid* – принимает на вход линейные векторные данные в формате shp и производит конвертацию переданных данных в растровый формат GeoTIFF. Размер ячейки результирующего файла, координаты области конвертации также передаются в качестве входных параметров сервиса;
4) *svm_learn* и *svm_classify* – осуществляют классификацию растительности нанаборах растровых файлов используя прецеденты, заданные в векторном формате. *Svm_learn* в результате своей работы производит файл с моделью, позволяющей классифицировать определенный тип растительности. *Svm_classify* в результате своей работы производит растровый файл, который отображает местоположение требуемых видов растительности, учитывая вероятность их нахождения в каждой точке;
5) *raster_addition*, *raster_subtraction*, *raster_multiplication* и *raster_division* – выполняют операции сложения и вычитания двух растров, умножения и деления значений в точках растровых файлов. Все четыре сервиса в результате своей работы производят растровый файл;

6) *elevation_slope_aspect* – в результате своей работы производит три растровых файла, соответственно содержащие данные о высоте, о наклоне и экспозиции области, ограниченной одним из входных параметров.
7) *newrastr* – создает растровый файл по заданным пользователем размерам и параметрам ячеек;
8) *road_availability* – производит расчет временной доступности точечных объектов на основании данных о дорожной сети и естественных преградах с учетом движения пешком;
9) *shp_to_geojson_converter*, *geojson_to_shp_converter* – конвертирует геоинформационные данные из векторного формата Shapefileв формат GeoJSONи обратно;
10) *bufferize_vector* – получает на вход набор точечных объектов в формате Shapefile и возвращает набор точечных объектов с построенными вокруг них буферными зонами определенного радиуса;
11) *tracelines* – принимает на вход векторный файл с заданными линейными объектами и возвращает растровый файл с определенным пользователем значением в точках, где проходят линейные объекты. Также возвращается растровый файл, содержащий атрибуты линейных объектов.

3.2.2 Запуск и контроль выполнения сервисов и сценариев

Интерфейс запуска и контроля выполнения сервисов и сценариев сервисов реализован в интеграции с Геопорталом ИДСТУ СО РАН и тесно связан с интернет-системой ввода и редактирования пространственных данных «Фарамант», которая упрощает и проверяет вводимые пользователями данные, в том числе геоинформационные данные. Пример интерфейса запуска сценария приведен на Рис. 27 для сценария распределенного суммирования чисел.

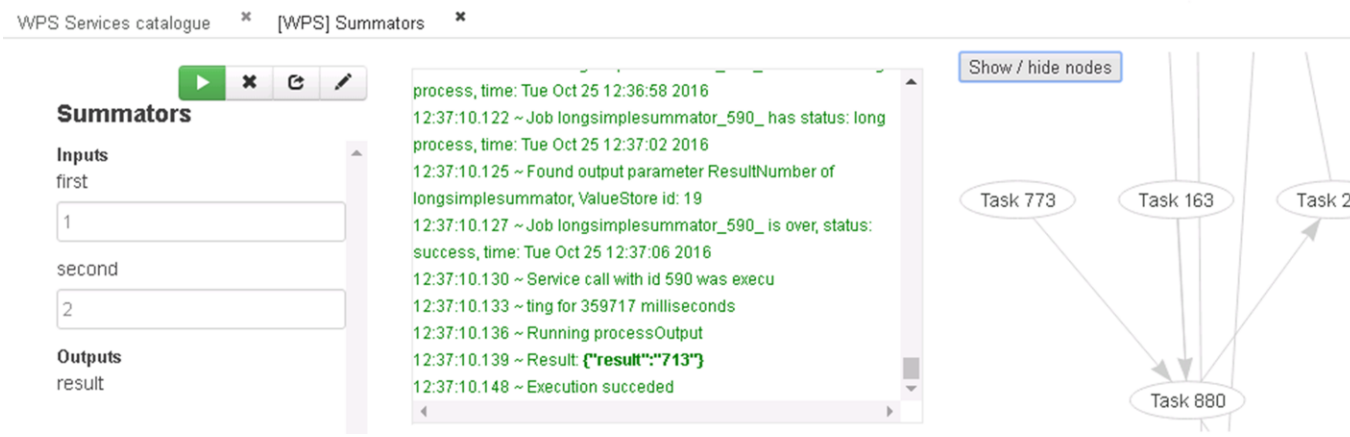

Рис. 27. Интерфейс запуска и контроля выполнения сервисов и сценариев сервисов

Интерфейс запуска сценариев сервисов разбит на три части. В левой части находится форма заполнения параметров сценария. Заполнение каждого параметра осуществляется с помощью использования специальных виджетов, определенных на этапе регистрации сервисов – например, для площадного объекта карта предоставляет инструменты рисования таких объектов, для даты предоставляется всплывающий календарь. Также в данной части интерфейса находятся элементы управления, ведущие на страницу редактирования сценария, открытия в отдельном окне, запуска и удаления.

После запуска сценария в средней части интерфейса начинает отображаться консоль выполнения сценария. В консоль в реальном времени выводятся сервисные сообщения, сообщающие о начале и завершении работы отдельных сервисов, времени выполнения сервисов, обработке промежуточных данных, изменении характеристик вычислительной среды. В случае критических ошибок, прекращающих выполнение сценария, пользователь будет также оповещен о причине остановки сценария.

Правая часть интерфейса выполнения сценариев служит для графического отображения состояния среды распределенных сервисов, то есть в ней отображаются участвующие вычислительные узлы, для каждого из которых указывается список

выполняемых на них сервисов, а также их статус (активен узел или неактивен). Также в правой части интерфейса строится текущий направленный ациклический граф зависимости веб-сервисов по данным, в котором для каждого сервиса отображается его статус выполнения.

Результат выполнения сценария также выводится в консоль. Если в результате выполнения сценария или сервиса получаются файлы в одном из поддерживаемых форматов геоинформационных данных (SHP, GeoTIFF и т.д.), то по мере их загрузки в локальную систему хранения данных производится их отображение на интерактивной веб-карте. Пример визуализации результатов выполнения изображен на Рис. 28 (на рисунке приведена визуализация результата выполнения сценария временной доступности, подробнее рассмотренного в п. 4.3). Интерактивная веб-карта позволяет настраивать различные параметры отображения данных и предоставляет несколько вариантов картографической подложки.

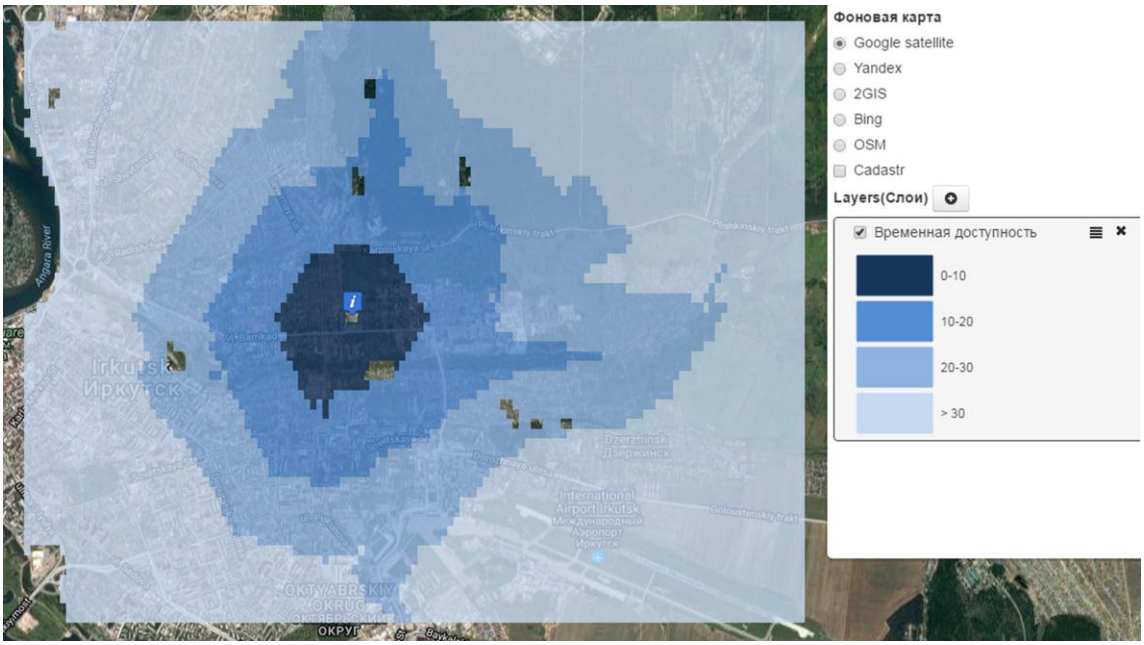

Рис. 28. Отображение результатов выполнения сценария или сервиса на интерактивной карте

Передача текущего состояния вычислительной среды осуществляется с помощью постоянного опроса Геопортала клиентом (веб-браузером). В ответ на запрос состояния сервер возвращает текущую консоль, массив описания состояний вычислительных узлов и выполняемых сервисов.

3.2.3 Публикация сценариев сервисов в виде WPS-сервисов

В каталоге сервисов предусмотрена возможность публикации созданных композиций сервисов в виде WPS-сервисов, то есть создаваемые композиции становятся автоматически доступными в виде WPS-сервисов, которые можно вызывать как внутри других сценариев, так и извне, как самостоятельный WPS-сервис.

При выборе стандарта протокола выбран именно WPS, так как он позволяет публиковать информацию обо всех имеющихся сценариях, созданных в системе, а также выполнять сценарии в виде сервисов, причем выполнение сценариев может длиться произвольное время.

3.2.4 Хранение результатов выполнения сервисов и сценариев

Нередко возникает ситуация, когда необходимо просмотреть результаты уже выполнившихся сценариев или вызовов сервисов. Для этих целей было реализовано хранения результатов выполнения сценариев и вызовов сервисов в специальном модуле хранения в рамках каталога сервисов и сценариев.

Модуль хранения истории вычислений для каждого вызова сценария или сервиса сохраняет его входные параметры, введенные пользователем, результаты выполнения, а также консольный вывод.

Доступ к истории выполненных вычислений производится посредством веб-интерфейса стандартными средствами Геопортала – в виде реляционной таблицы, пример которой приведен на Рис. 29.

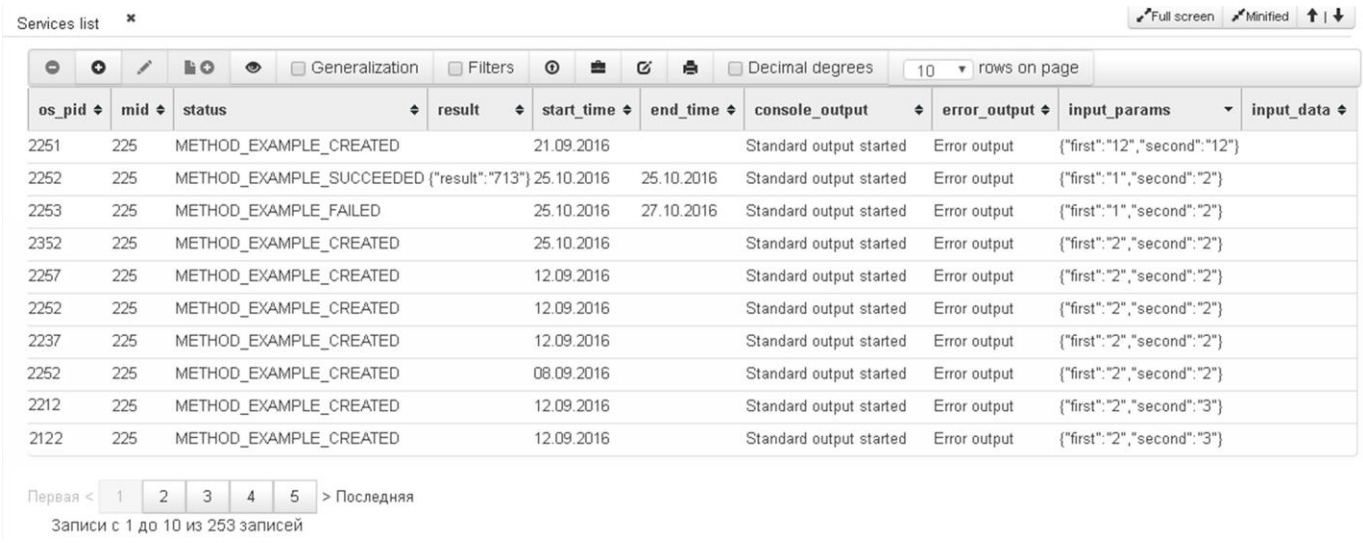

Рис. 29. Интерфейс просмотра истории выполнения сервисов

3.3 Организация доступа к файлам

Одна из важных функций среды — обеспечение передачи и получения данных веб-сервисами через Интернет. Следовательно, необходимо использование систем хранения данных, обеспечивающих регламентированный доступ через Интернет не только на чтение, но и на запись. В качестве входных данных чаще всего используют файлы и реляционные таблицы. Для веб-сервисов применяются и могут потребоваться следующие протоколы и стандарты доступа к данным: HTTP, WFS, WFS-T, WCS, WMS. Учитывая возможно большой объем данных, использование стандартов WFS и WFS-T может оказаться не эффективным из-за текстового представления данных. Стандарты WCS, WMS не позволяют производить изменение данных. Поэтому необходимо в первую очередь поддержать протокол HTTP и стандарт Simple Features Specification For SQL. В рамках среды разработаны:

1) файловая система хранения данных (СХД), в которой пользователю предоставляется директория, и доступ к файлам регламентируется в соответствие с запуском веб-сервисов;

2) СУБД PostgreSQL (с модулем расширения PostGIS) для хранения реляционных данных, в котором для каждого пользователя создается схема данных.

Для работы в браузере с этими компонентами предоставляется файловый менеджер и подсистема редактирования и отображения реляционных данных.

Если передаваемый параметр является файлом, расположенным в директории пользователя в СХД, то необходима передача данного файла веб-сервису. В соответствие со стандартом веб-сервису передаются URL адрес файла, используя который, веб-сервис должен по протоколу HTTP скачать файл. Файл должен находиться в открытом доступе. Так как данные пользователей могут содержать информацию ограниченного доступа, то любой файл, хранимый в СХД, по умолчанию не может быть свободно доступным. Таким образом, необходимо одновременно защитить и предоставить веб-сервисам доступ к хранимым в СХД файлам. Механизм доступа и контроля обращений к файлам построен следующим образом – каждому экземпляру выполнения веб-сервиса присваивается уникальный идентификатор. Если выполняемый веб-сервис должен получить в качестве параметров файлы, то для каждого из файлов создается уникальная ссылка, привязываемая именно к выполняемому экземпляру. Таким образом, WPS-сервис может свободно скачать требуемый файл из СХД по сгенерированной ссылке определенное число раз, причем все ссылки, созданные для определенного экземпляра, уничтожаются, как только метод завершит свою работу. Если файл является результатом работы веб-сервиса, то подсистема выполнения сценариев загружает в СХД в соответствующую пользовательскую директорию по протоколу HTTP. URL–адреса файлов берутся из описаний результатов работы веб-сервисов.

3.4 Облачная инфраструктура

Создаваемая система динамического выполнения композиций сервисов в гетерогенной среде в первую очередь направлена на использование специалистами-предметниками, нуждающимися в публикации и использовании сервисов в рамках междисциплинарных исследований. Таким образом, необходимы средства, которые

бы упрощали процесс развертывания вычислительных узлов, предназначенных для разработки и публикации вычислительных модулей в виде веб-сервисов.

В данных целях был разработан специальный модуль для Портала ИСДТУ СО РАН, который позволяет создавать виртуальные машины пользователям системы динамического выполнения композиций сервисов в гетерогенной среде с помощью веб-интерфейса. Создаваемые виртуальные машины имеют предустановленный набор программного обеспечения, помогающего при публикации собственного вычислительного сервиса.

Для регистрации виртуальной машины пользователю достаточно заполнить веб-форму, в которой необходимо указать название создаваемой виртуальной машины, а также тип операционной системы, которая будет предустановлена на машине. На данный момент пользователю предоставляется на выбор несколько типов операционных систем семейств Windows Server, Debian и CentOS. Скриншот формы представлен на Рис. 30, на Рис. 31 изображен интерфейс отображения имеющихся у пользователя виртуальных машин.

Рис. 30. Интерфейс создания виртуальной машины

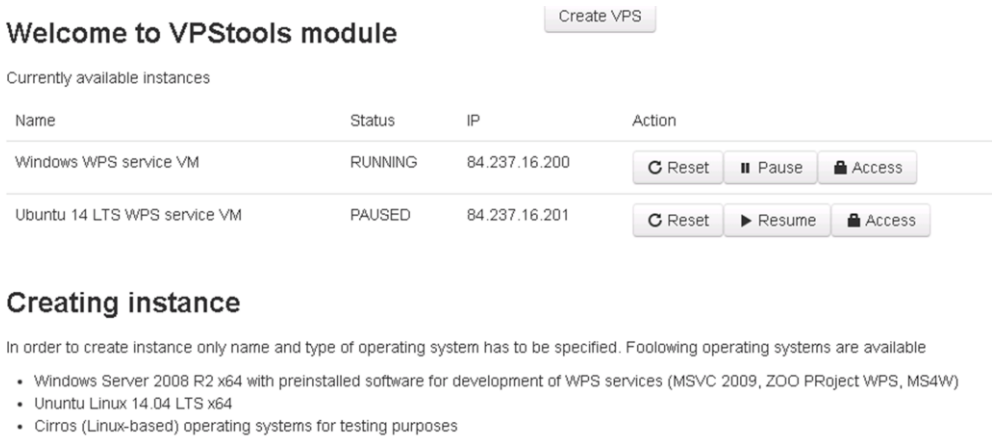

Рис. 31. Интерфейс просмотра имеющихся виртуальных машин

После отправки данных формы на сервер пользователь ждет некоторое время. Время создания виртуальной машины зависит от размера образа виртуальной машины, максимальное время, наблюдаемое при создании машины из образа весом в 50GB, равнялось приблизительно 1 минуте. После создания и включения виртуальной машины пользователю сообщаются данные для входа на виртуальную машину. Пользователь получает полный контроль над виртуальной машиной. При зависании или потери связи с виртуальной машиной пользователь может перезагрузить виртуальную машину посредством веб-интерфейса.

На всех виртуальных машинах установлены следующие средства разработки и публикации веб-сервисов (в зависимости от операционной системы):

- компиляторы C/C++ с открытым кодом

- текстовые редакторы с подсветкой синтаксиса с открытым кодом

- веб-сервер с открытым исходным кодом

Все виртуальные машины также оснащены службой WPS-сервисов ZOO Project – реализацией стандарта WPS консорциума OGC с открытым исходным кодом [61]. ZOO Project был выбран в силу своей кроссплатформенности, а также

поддержки сервисов, написанных на различных языках, что делает интеграцию уже реализованных вычислительных модулей в виде WPS-сервиса проще. В стандартной установке ZOO Project поддерживает вычислительные модули, реализованные в виде DLL-библиотек, приложений Java, а также скриптов на языках PHP, JavaScript и Python. Также ZOO Project был расширен для запуска выполняемых файлов (exe файлы для ОС Windows и sh для систем, базирующихся на Linux).

Таким образом, специалисты-предметники имеют инструмент, с помощью которого они могут создать отдельную виртуальную машину под свой вычислительный сервис с предустановленными средствами разработки и публикации вычислительных модулей в виде WPS-сервисов.

Возможность публикации программных модулей пользователей в виде сервисов с помощью облачной виртуальной инфраструктуры реализована на мощностях Телекоммуникационного центра коллективного пользования «Интегрированная информационно-вычислительная сеть Иркутского научно-образовательного комплекса» (ИИВС ИРНОК). Под пользовательскую виртуализацию был выделен сервер HP Proliant 460cGen8, подключенный к системе хранения данных. В качестве технологии виртуализации используется программное обеспечение компании VMware, обеспечивающее процесс выделения вычислительных, дисковых ресурсов, создание локальных подсетей и организация доступа к сети Интернет, квотирование нагрузки и контроль состояния оборудования при создании и управлении виртуальными машинами. Поверх инфраструктуры VMware развернута система с открытым кодом OpenStack [62]. OpenStack позволяет связывать большое количество разнообразных систем виртуализации в один логический кластер с попутным выделением ресурсов, миграцией, настройкой создаваемых виртуальных машин, но основная используемая возможность OpenStack – предоставление программного API для задач, связанных с

работой с облачной инфраструктурой. На Рис. 32 показана схема облачной инфраструктуры, иллюстрирующая весь процесс взаимодействия – Геопортал (в частности, модуль, позволяющий пользователям создавать свои виртуальные машины) работает только с API OpenStack. В дальнейшем использование OpenStack благоприятно скажется на возможностях масштабирования системы, например, при подключении систем виртуализации, базирующихся на других технологиях виртуализации (XEN, KVM и т.д.).

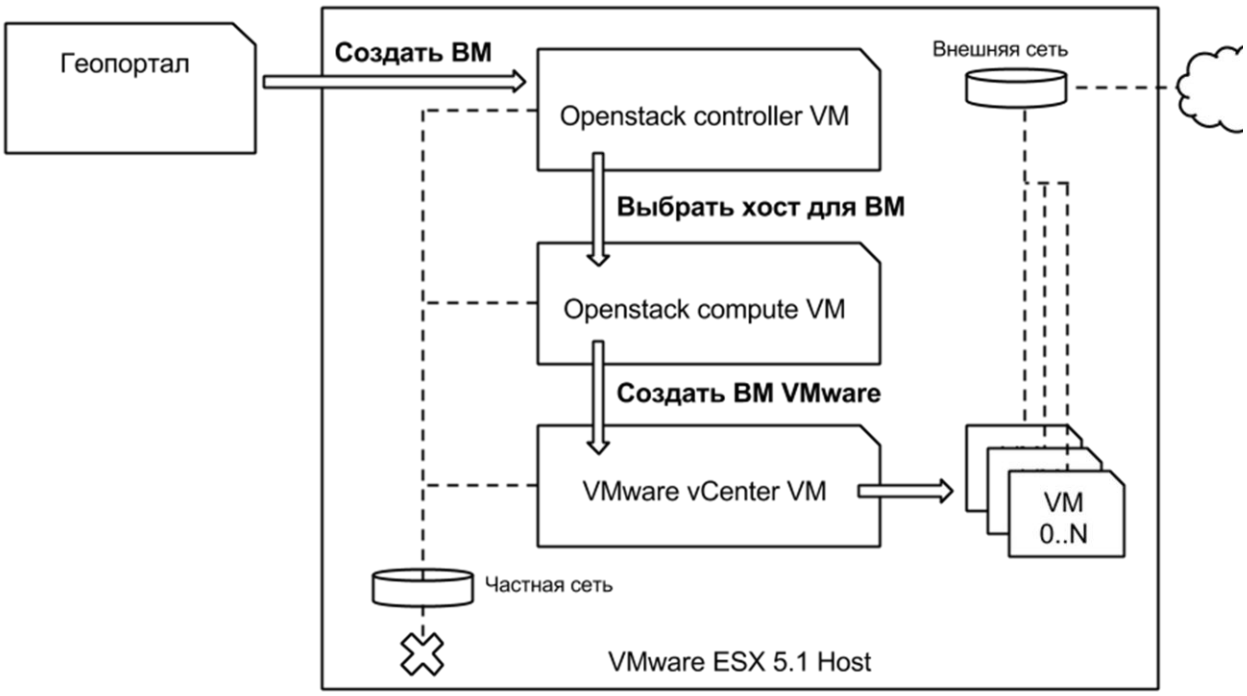

Рис. 32. Взаимодействие системы динамического выполнения композиций сервисов в гетерогенной среде и облачной виртуальной инфраструктуры

3.5 Выводы

В данной главе были описаны основные аспекты реализации системы выполнения композиций сервисов в гетерогенной среде. Рассмотрены особенности реализации метода планирования выполнения композиций сервисов в гетерогенной среде, а также метода задания композиций сервисов в виде программ на языке программирования JavaScript. Был рассмотрен весь процесс работы с композициями

сервисов, начиная от регистрации участвующих в композициях сервисов и задания самих композиций до их выполнения, планирования и хранения результатов выполнения композиций сервисов. Были рассмотрены основные инструменты для работы в системе динамического выполнения композиций сервисов в гетерогенной среде, возможности создания вычислительных узлов на основе существующих технологий виртуализации, были рассмотрены основные аспекты передачи данных между сервисами. Данная глава в полной мере раскрывает все основные момент реализации разработанного программного инструментального средства для организации взаимодействия программ и программных систем в гетерогенной среде.

ГЛАВА 4. АПРОБАЦИЯ

В данной главе рассказывается об апробации системы динамического выполнения композиций сервисов в динамической среде при решении реальных задач, возникающих в процессе проведения междисциплинарных исследований. На примере представленных задач показывается поведение системы в различных ситуациях, иллюстрирующих возможности планирования в условиях гетерогенности среды, устойчивость к изменениям в вычислительной среде, а также основные механизмы работы системы динамического выполнения композиций сервисов в гетерогенной среде.

4.1 Классификация типов растительности методом опорных векторов

В целях апробации устойчивости работы композиции сервисов к происходящим изменениям в гетерогенной вычислительной среде (отключение узлов, переменное время выполнения сервисов, завершение выполнения заданий с ошибкой) была выбрана задача поиска наиболее подходящих данных для классификации типов растительности методом опорных векторов [29].

Постановка задачи: существует некий набор векторных данных в виде файлов в геоинформационном векторном формате SHP, называемых прецедентами, содержащих местоположения представителей следующих типов растительности – степь, сосновый лес, болото, пихтовый лес, берёзовый лес, лиственничный лес (всего 6 типов). Местоположение представителей задается точками с координатами, в которых гарантированно находится некий образец типа растительности. Обычно наборы данных точек получаются в результате работы полевых экспедиций и непосредственных измерений. Данный набор предоставлен специалистами предметной области. Также даны 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 бы обеспечивали наибольшую точность определения типа растительности для каждой пары типов.

Для классификации типов растительности применяется метод опорных векторов *SVM (Support Vector Machine). На основе данного метода реализован* WPS-сервис *SVM_Learn*, развернутый на 4 вычислительных узлах в пределах локальной облачной инфраструктуры. Сервис работает под управлением WPS-службы ZOO Project, виртуальные машины работают под управлением операционной системы Windows Server 2008. *SVM_Learn* принимает на вход набор растров, SHP файл с прецедентами, название класса и значения класса для разделения прецедентов. Результатов работы сервиса является файл классификатора, а также значение точности найденной модели классификации.

Задача осложняется тем, что при поиске растров, дающих наиболее точную модель для рассматриваемых файлов прецедентов, необходимо осуществить перебор всех возможных комбинаций растровых файлов, что требует вычислительных и временных ресурсов.

Решение задачи: для перебора возможных комбинаций растровых файлов, предоставленных для анализа специалистами предметной области, необходимо реализовать композицию сервисов в виде сценария на языке JavaScript. Заданная композиция сервисов должна перебирать все возможные комбинации растров для каждого сочетания имеющихся прецедентов. 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.

В процессе выполнения композиции сервисов вычислительные узлы будут менять свою доступность, выполнение отдельных сервисов будет иметь искусственно уменьшенное или увеличенное время выполнения, процесс выполнения не должен прерываться, система динамического выполнения композиций сервисов в динамической среде должна адаптироваться к изменениям среды и не прекращать выполнение сценария.

Для начала необходимо задать непосредственный сценарий, который будет перебирать комбинации растров для определенных прецедентов. Как уже было сказано в п. N, существует специальный веб-интерфейс, упрощающий и автоматизирующий процесс ввода сценариев. На Рис. 33 представлен скриншот формы создания сценария на Портале ИДСТУ СО РАН. Для упрощения и корректировки ввода кода сценария пользователю предоставляется редактор с подсветкой синтаксиса и список сервисов, уже зарегистрированных на Портале. Список сервисов представлен в виде JavaScript-функций с описанием параметров. Для удобства задания сценария композиции сервисов на основе описания входных и выходных параметров сценария в автоматическом режиме составляется функция-обёртка, пользователю остаётся определить саму логику составляемого сценария.

IMPORTANT Before filling the function body, define input parameters

IMPORTANT Input values are provided in the **input** object. For example, values of input parameters "a" and "b" will be available as **input.a** and **input.b**. When it comes to returning output, for example, we got an output field with the name "result", its value has to be set through the **mapping** object in following way: **mapping.output.set(VALUE)**.

```
/* Input: roads, houses, attrname, extent, cellsize; Output: result */
function Pollution_calculation(input, mapping){ ... }}
/* Input: number1, number2, numbernotrequired; Output: ResultNumber */
function longsimplesummator(input, mapping){ ... }}
/* Input: grid1, grid2, grid3, grid4, grid5, shape1, shape2, attributename, attributetrue, attributefalse; Output: result */
function SVM_composite(input, mapping){ ... }}
/* Input: model, grid1, grid2, grid3, grid4, grid5; Output: Result */
function SVM_Classify(input, mapping){ ... }}
/* Input: Sourse, Mode, AttrName, EXTENT, CELLSIZE, CELLSIZE2; Output: Resulf */
function theme2grid(input, mapping){ ... }}
/* Input: number1, number2, numbernotrequired; Output: ResultNumber */
function simplesummator(input, mapping){ ... }}
/* Input: msg */
function print() { [native code] }
```

Press Generate wrapper to construct function wrapper based on entered parameters

```
function SVM_composite_full_sort(input, mapping){
    var classes = [
        {id: 3, name: 'Step'},
        {id: 4, name: 'Sosnovyj les'},
        {id: 5, name: 'Boloto'},
        {id: 9, name: 'Pihtovyj les'},
        {id: 10, name: 'Berezovyj les'},
        {id: 12, name: 'Listvennichnyj les'},
        {id: 14, name: 'Elovyj les'}
    ];

    var commonParameters = {
        precedent:      input.precedents,
        attribute_name: input.attributename
    };

    var combinationLimit = 3;
    var numberOfGrids    = 2;

    var getCombinations = function(chars) {
        var result = [];
        var f = function(prefix, chars) {
            for (var i = 0; i < chars.length; i++) {
                result.push(prefix + chars[i]);
                f(prefix + chars[i], chars.slice(i + 1));
            }
        };

        f('', chars);
        return result;
    };

    var numberOfValueStores = 0;
    var valueStores         = [];

    var testCombination = function(aIndex, bIndex) {
        print('Learning from ' + classes[aIndex].name + ' and ' + classes[bIndex].name);

        commonParameters.attribute_true  = classes[aIndex].id;
        commonParameters.attribute_false = classes[bIndex].id;

        var combinationsSourceArray = [];
        for (var t = 1; t <= numberOfGrids; t++) {
            combinationsSourceArray.push('' + t);
        }

        var combinations = getCombinations(combinationsSourceArray);
        for (var c = 0; c < combinations.length; c++) {
            print('Trying the conbination of ' + combinations[c] + ' grids');

            var splitted = combinations[c].split('');

            var localCommonParameters = commonParameters;
            for (var s = 0; s < splitted.length; s++) {
                localCommonParameters['grid' + (s + 1)] = input['grid' + splitted[s]];
            }

        var localPrecisionValueStore = new ValueStore();
            var localModelValueStore     = new ValueStore();

            valueStores[numberOfValueStores]             = {};
            valueStores[numberOfValueStores].precision   = localPrecisionValueStore;
            valueStores[numberOfValueStores].model       = localModelValueStore;
            valueStores[numberOfValueStores].grids       = splitted;
            valueStores[numberOfValueStores].description = classes[aIndex].name + ' and ' + classes[bIndex].name;

            SVM_Learn_from_shp(localCommonParameters, {Model: localModelValueStore, Precision: valueStores[numberOfValueStor

            numberOfValueStores++;
        }

        print('Total calls for ' + classes[aIndex].name + ' and ' + classes[bIndex].name + ': ' + numberOfValueStores);
    };


    for (var i = 0; i < combinationLimit; i++) {
        for (var j = i; j < combinationLimit; j++) {
            if (i !== j) {
                testCombination(i, j);
            }
        }
    }

    for (var v = 0; v < numberOfValueStores; v++) {
        print(valueStores[v].description + ' precision value: ' + valueStores[v].precision.get());
    }

    mapping.result.set(1);
}
```

Save and exit | Save

Рис. 33. Форма создания сценария

У сценария имеются следующие входные параметры:

1. Прецеденты в векторном формате – для упрощения ввода прецедентов для данного параметра используется элемент управления theme_select, который позволяет преобразовывать выбранную таблицу из реляционной базы данных Портала в файл в формате SHP (векторный формат пространственных данных). После преобразования данных таблицы в SHP файл выполняемому сценарию передается URL ссылка на полученный файл. Таким образом, данные, которые можно просматривать и редактировать в интерактивном режиме посредством инструментов, предоставляемых Порталом ИДСТУ СО РАН, можно передать в выполняемые сервисы или сценарии сервисов с помощью указанного элемента управления;

2. Название атрибута – указывается название атрибута реляционной таблицы (впоследствии сгенерированного SHP-файла), на основе которого будет производиться классификация. Для данного параметра в качестве элемента управления выбрано текстовое поле;

3. Растровые файлы, необходимые для анализа – производится выборка файлов из локальной СХД с помощью элемента управления file. После выбора файла генерируется ссылка, позволяющая скачать файл, и данная ссылка передается в сценарий или сервис. В рассматриваемом примере на вход сценария подается пять различных растровых файлов.

Код JavaScript сценария представлен ниже (в сокращённом виде). Переменная `classes` (сноска 1 в приведённом ниже коде сценария) представляет собой массив, в котором содержатся перебираемые классы с идентификаторами (идентификатор будет использоваться внутри самого метода SVM для определения принадлежности прецедента к классу). После объявления классов в сценарии идет непосредственный

перебор комбинаций с формированием определенных параметров для каждого запуска сервиса, производимых в вычислительном сервисе *SVM_Learn_from_shp* (сноска 2 в приведённом ниже коде сценария). *SVM_Learn_from_shp* представляет собой функцию-обертку для сервиса SVM. Далее (сноска 3 в приведённом ниже коде сценария) происходит выбор самых точных моделей классификации для каждой комбинации, что и является целью выполнения композиции сервисов.

```
function SVM_composite_full_sort(input, mapping){
     var classes = [{id: 1, name: 'Step'}, .. ](1)

     /* ... */
     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 ' + ...);
     }
}
```

В самом начале выполнения композиции сервисов пользователю необходимо ввести параметры выполнения композиции. Портал предоставляет необходимый

интерфейс для ввода параметров сервисов или композиции сервисов, позволяя вводить как строковые параметры, так и выбирать файлы, хранящиеся в специализированной СХД, или делать выборку данных из реляционных таблиц. В рассматриваемом случае выбирается реляционная таблица, из которой будут браться прецеденты, а также указывается классифицирующий столбец таблицы и растры, над которыми будет производиться анализ.

По мере наполнения глобальной очереди вызовов сервисов и наступления, указанных выше событий происходит перестроение расписания с учетом текущего состояния вычислительной среды. Рассмотрим событие отключения половины вычислительных узлов в процессе выполнения.

На рис. N приведено фактическое расписание выполняющихся заданий без выключения вычислительных узлов в процессе выполнения. Рис. 34 является скриншотом фрагмента веб-интерфейса запуска сценариев. Всего сценарий выполнялся в течение 158 секунд.

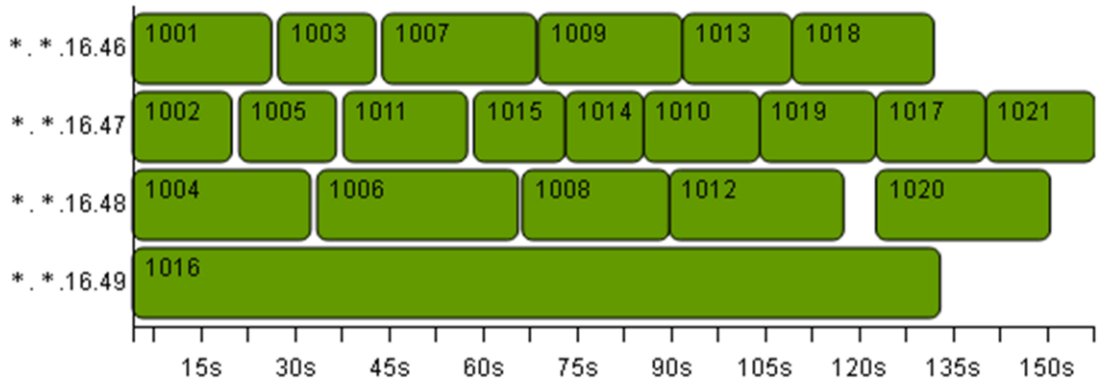

Рис. 34. Расписание без отключения узлов

По мере выполнения сценария происходит автоматическое построение направленного ациклического графа задачи, приведенного на Рис. 35.

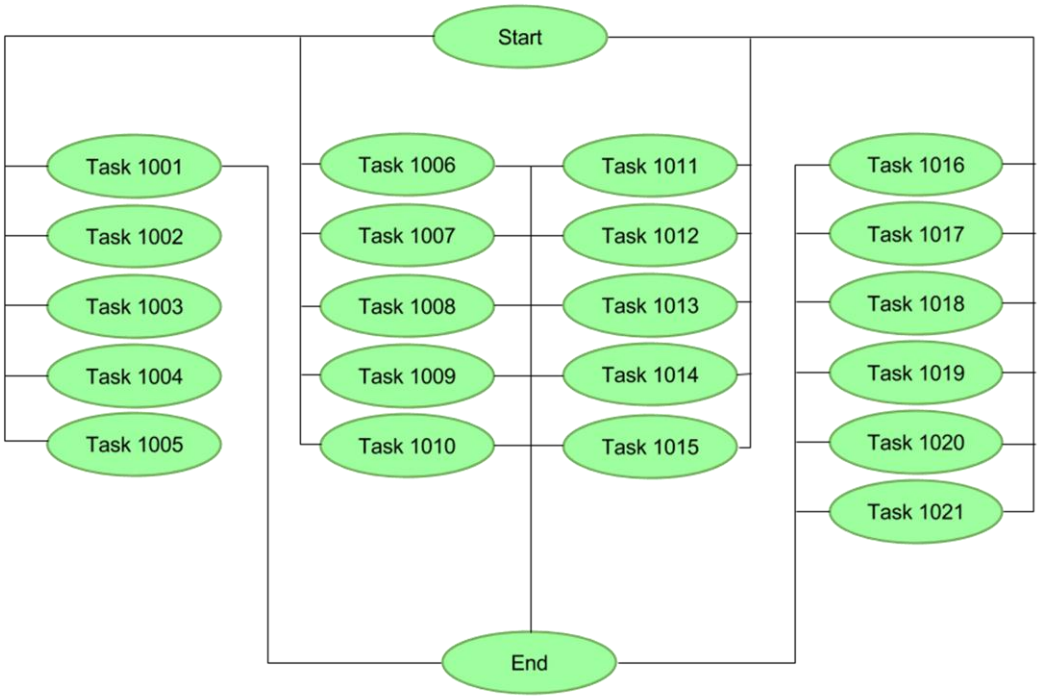

Рис. 35. Направленный ациклический граф (DAG) композиции

На Рис. 36 приведено фактическое расписание выполнения в случае с отключением двух узлов в процессе выполнения сценария. На 21 и 38 секундах выполнения были отключены вычислительные узлы *.*.16.46 и *.*.16.47 (был выключен веб-сервер Apache, то есть запросы к удаленному серверу со стороны диспетчера завершались по таймауту подключения), соответственно, диспетчер обработал данное изменение в вычислительной среде и перенаправил назначенные на эти узлы задания на другие вычислители. Стоит отметить, что через определенное время диспетчер пробует повторно назначить на неактивные узлы задания, и в случае удачи данные узлы снова учитываются при составлении расписания выполнения композиции сервисов. В случае с отключением части вычислительных узлов сценарий выполнялся в течение 174 секунд.

Рис. 36. Расписание при отключении узлов

На Рис. 37 приведено фактическое расписание выполнения в случае с ошибкой выполнения сервиса в процессе выполнения сценария. Для симуляции ошибки сервиса на вычислительном узле *.*.16.46 сервис *SVM_Learn* был модифицирован для аварийного завершения при любом запросе. Таким образом, в ответ на запрос на выполнение сервиса *SVM_Learn* на вычислительном узле *.*.16.46 диспетчеру возвращается XML-документ, содержащий сообщение об ошибки. В таком случае диспетчер перестраивает план с учетом того, что завершившийся с ошибкой сервис больше не будет назначаться на определенный вычислительный узел. На Рис. 37 видно, что на 45 секунде диспетчер принял решение не выполнять сервис *SVM_Learn* на вычислительном узле *.*.16.46 из-за произошедшей ошибки сервиса.

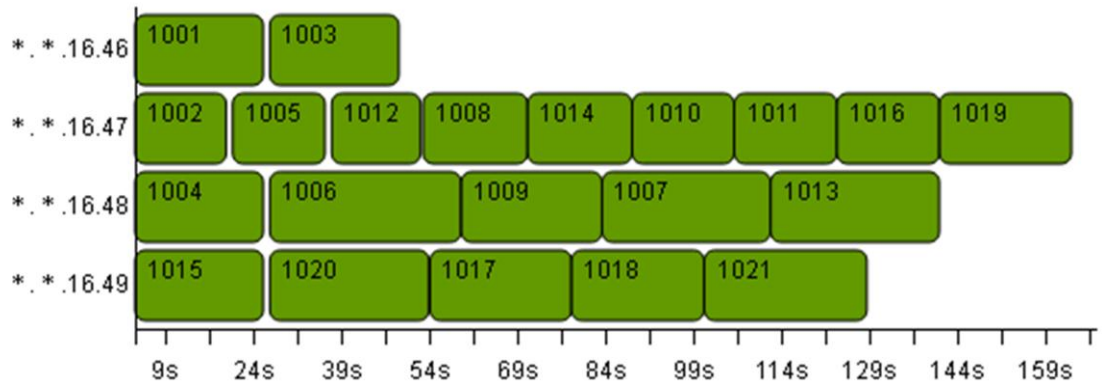

Рис. 37. Расписание при ошибке сервиса

По завершению выполнения композиции сервисов в обоих случаях в пользовательскую консоль выводится результат выполнения композиции сервисов. Результаты работы сервисов (файлы моделей классификации, которые могут быть впоследствии использованы) автоматически загружены на СХД Портала. Результатом выполнения композиции является список растровых файлов для каждого сочетания типов растительности, который дает наиболее точную оценку получившегося классификатора. Пример консольного вывода результата выполнения приведен ниже:

```
Skippeddecisions: 0, takendecisions: 0
Service composition is completed
Best precision for Step and Sosnovyj les is 0.83, grids: N.tif, E.tif
Best precision for Step and Boloto is 0.20, grids: E.tif, S.tif, A.tif
...
Best precision for Listvennichnyj les and Elovyjlesis 0.67, grids: N.tif, E.tif, A.tif
Done
```

Рассмотренный пример демонстрирует преимущество разработанной системы, заключающееся в автоматической обработке событий, происходящих в распределенной гетерогенной среде (изменение ожидаемого времени выполнения еще не запущенных сервисов, изменение доступности вычислительных узлов или ошибка при выполнении одного из вызываемых вычислительных сервисов), без участия пользователя. В сценарии композиции определяется только логика взаимодействия сервисов, построение направленного ациклического графа и планирование выполнения сервисов происходит автоматически.

4.2 Расчет степени загрязнения

Предложенная система планирования выполнения композиции сервисов апробирована на задаче расчета загрязнения четырьмя загрязняющими веществами (сажа, парниковые газы SO2, NO3 и CO) для определенного региона. Сценарий учитывает загрязнения от точечных (домов, юрт) и линейных объектов (дороги) при известном среднем загрязнении за определенный период от точечного объекта и одного километра дороги соответственно. Данный сценарий был применения для расчета загрязнения города Улан-Батор, Монголия.

Постановка задачи: существуют статистические данные, содержащие значения выбрасываемых загрязняющих веществ для точечных объектов, а также значения средних выбросов веществ на 1 километр дороги. Все дороги разделены на несколько классов, для каждого из которого определен собственный коэффициент, учитывающий интенсивность среднего выброса загрязняющих веществ – например, коэффициент многополосной дороги выше коэффициента однополосной грунтовой дороги. Необходимо рассчитать суммарную величину выбросов в каждой ячейке по каждому загрязняющему веществу.

Решение задачи: для решения задачи необходимо составить сценарий на языке JavaScript, вычисляющий значения выбросов загрязняющих веществ в ячейках

выбранной области. В сценарии используется 3 различных сервиса – *vectorToGrid*, *roadToGrid*, *raster_sum*, в вычислениях участвует 4 вычислительных узла, причем *vectorToGrid* выполним только на первых двух. Сервис *vectorToGrid* используется для расчета выделяемых загрязнений от точечных источников (котельных, ТЭЦ и т.д.) в ячейках регулярной сетки. Сервис *roadToGrid* используется для расчета выделяемых загрязнений от автотранспорта в ячейках регулярной сетки. Сервис *raster_sum* используется для суммирования выделяемых загрязнений от точечных источников и автотранспорта.

Сценарий имеет следующие входные параметры:

1) Векторный файл в формате SHP, содержащий точечные объекты (юрт, частных домов, ТЭЦ) –производится выборка файлов из локальной СХД с помощью элемента управления file. После выбора файла генерируется ссылка, позволяющая скачать файл, и данная ссылка передается в сценарий или сервис;

2) Векторный файл в формате SHP, содержащий линейные объекты (дороги) - производится выборка файлов из локальной СХД с помощью элемента управления file. После выбора файла генерируется ссылка, позволяющая скачать файл, и данная ссылка передается в сценарий или сервис;

3) Размер ячейки создаваемых растровых файлов;

4) Область, для которой необходимо произвести расчет;

5) Средние значения выбросов для точечных и линейных объектов для четырех загрязняющих веществ.

Код JavaScript сценария представлен ниже. Сервис *vectorToGrid*(сноска 1 в приведённом ниже коде сценария) в результате своей работы производит растровый файл, показывающий среднее загрязнение от определенного газа в ячейках

регулярной сетки для точечных объектов. Сервис *roadToGrid* (сноска 2 в приведённом ниже коде сценария) производит расчеты, аналогичные расчетам сервиса *vectorToGrid*, но для линейных объектов. Сервис *raster_sum* (сноска 3 в приведённом ниже коде сценария) суммирует результаты работы сервисов *vector2grid* и road2grid, таким образом, получая совокупное загрязнение от определенного газа на местности. Целью выполнения сценария является набор растровых файлов, каждая ячейка которых содержит значение загрязнения местности по всем интересующим загрязняющим веществам, а также значение совокупного загрязнения (всего 5 растровых файлов).

```
function test_scenario(input, resultStore) {
  for (var i = 0; i <4; i++) {
    var result1 = new ValueStore();
    var result2 = new ValueStore();
    vectorToGrid({in1: input.a[i]}, {result: result1});(1)
    roadToGrid({in1: input.b[i]}, {result: result2});(2)
    raster_sum({in1: result1, in2: result2},(3)
          {result: resultStore.my_scenario_result[i]});
  }

  var sum1 = new ValueStore(), sum2 = new ValueStore();
  raster_sum({file1:results[0],file2:results[1]}, {Output: sum1});
  raster_sum({file1:results[2],file2:results[3]}, {Output: sum2});
  raster_sum({file1: sum1, file2: sum2}, {Output: results[4]});
}
```

На Рис. 38 показано составленное для сценария расписание. Сервисы, выполняющиеся около 16 секунд, занимаются непосредственно расчетом выбросов

загрязняющих веществ (сервисы *vectorToGrid* и *roadToGrid* с идентификаторами 1001, 1004 … 1011), в то время как сервисы, занимающиеся только суммированием растровых файлов (сервис *raster_sum*), выполняются около 3 секунд. Автоматически составленный направленный ациклический граф композиции изображен на Рис. 39.

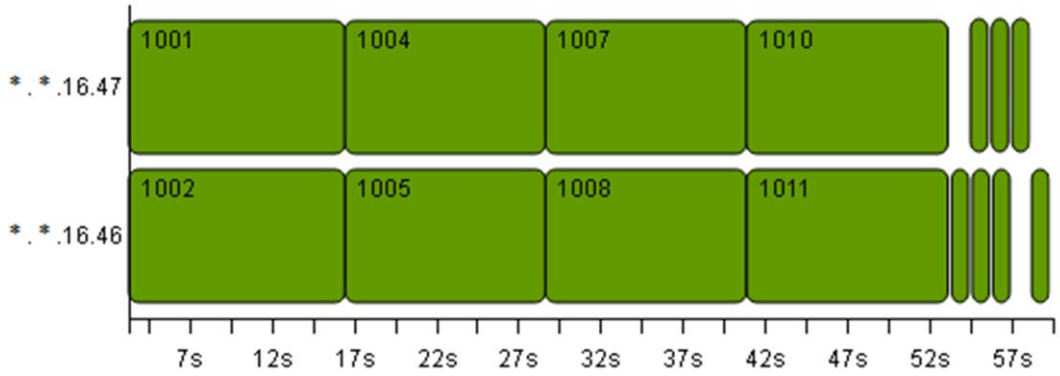

Рис. 38. Составленное расписание

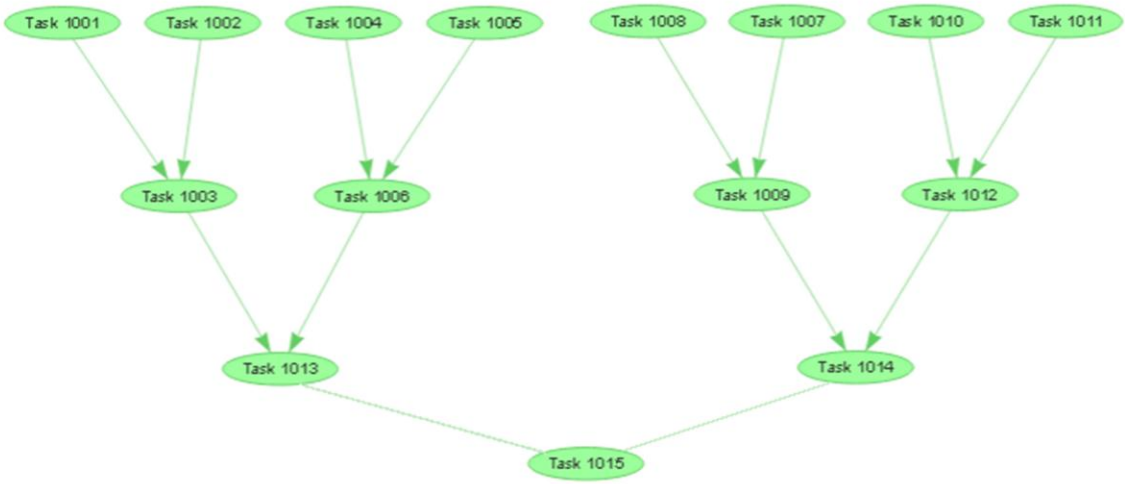

Рис. 39. Направленный ациклический граф (DAG) композиции

Результатом выполнения композиции, в соответствии с постановкой задачи, является набор растровых файлов со значениями загрязняющих веществ. На Рис. 40 изображено наложение результата работы композиции на интерактивную карту

(интерактивная карта и средство отображения растровых файлов предоставлено Порталом ИДСТУ СО РАН). На основании значений, содержащихся в ячейках растрового файла, производится окрашивание растра в зависимости от концентрации загрязняющих веществ.

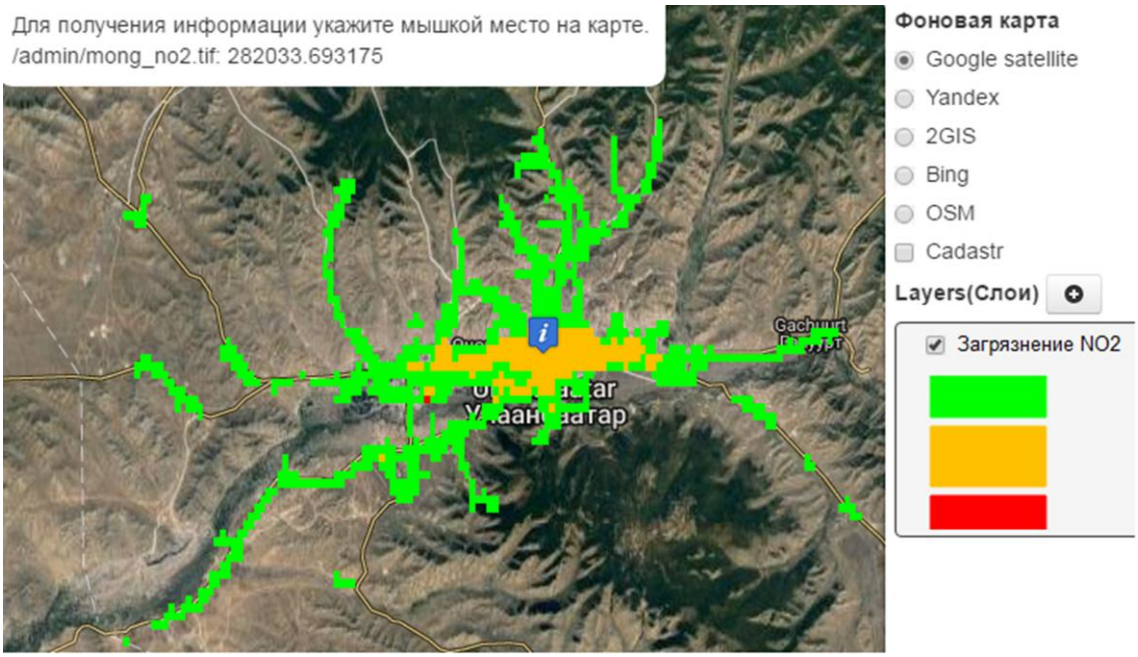

Рис. 40. Визуализация результата выполнения композиции

Рассмотренный пример демонстрирует преимущество разработанной системы, заключающееся в автоматическом преобразовании композиции сервисов с имеющимися между ними зависимостями в направленный ациклический граф с последующим параллельным выполнением сервисов данной композиции (например, итерации циклов распараллеливаются автоматически, без каких-либо настроек со стороны разработчика сценария). При выполнении данной композиции диспетчер выполняет планирование и контроль выполнения сервисов в распределенной вычислительной среде.

4.3 Определение доступности образовательных организаций

В целях демонстрации возможности работы с промежуточными результатами работы вычислительных сервисов в рамках выполняемых сценариев, приводится задача расчета временной доступности образовательных учреждений для определенного района города с учетом дорожной сети и естественных преград (например, водные объекты). Данная задача возникает при планировании развития городской инфраструктуры [24].

Постановка задачи: существует набор точечных объектов, соответствующих расположению объектов образовательной инфраструктуры. Необходимо произвести расчет временной 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 данной задачи является композиция сервисов, которая осуществляет разбиение начального набора точечных объектов инфраструктуры и расчет временной доступности для каждого объекта. Разбиение объектов должно осуществляться внутри сценария композиции с помощь стандартных средств, предоставляемых языком программирования JavaScript.

В сценарии используются сервисы *shp_to_geojson_converter*, *geojson_to_shp_converter*, *bufferize_vector*, *road_analysis*. 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 из себя буферные зоны, построенные вокруг входных точечных объектов с определённым радиусом в метрах. Сервис road_analysis осуществляет расчет времени, которое необходимо чтобы добраться до каждого объекта образовательной инфраструктуры.

Сценарий имеет следующие входные параметры:

1. Файл с набором точечных объектов в векторном формате SHP. Каждый точечный объект – объект образовательной инфраструктуры;
2. Файл с набором границ объектов, представляющих естественные преграды для расчета (водные объекты) в векторном формате SHP;
3. Файл, содержащий структуру дорожной сети в векторном формате SHP;
4. 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 приведённом ниже коде сценария), то есть результат выполнения сервиса становится доступных внутри сценария в виде стандартного объекта JavaScript. Далее происходит выполнение последовательности оставшихся сервисов для каждого точечного объекта, полученного в результате 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, необходимого для того, чтобы добраться до каждого из объектов образовательной инфраструктуры. На Рис. 41 показана визуализация результата выполнения сценария для отдельной школы, расположенной в центре отображаемого растрового файла и отмеченной специальным маркером. На легенде справа можно наблюдать оценки времени, необходимого для достижения образовательного объекта. Интервалы времени окрашиваются в соответствующий цвет.

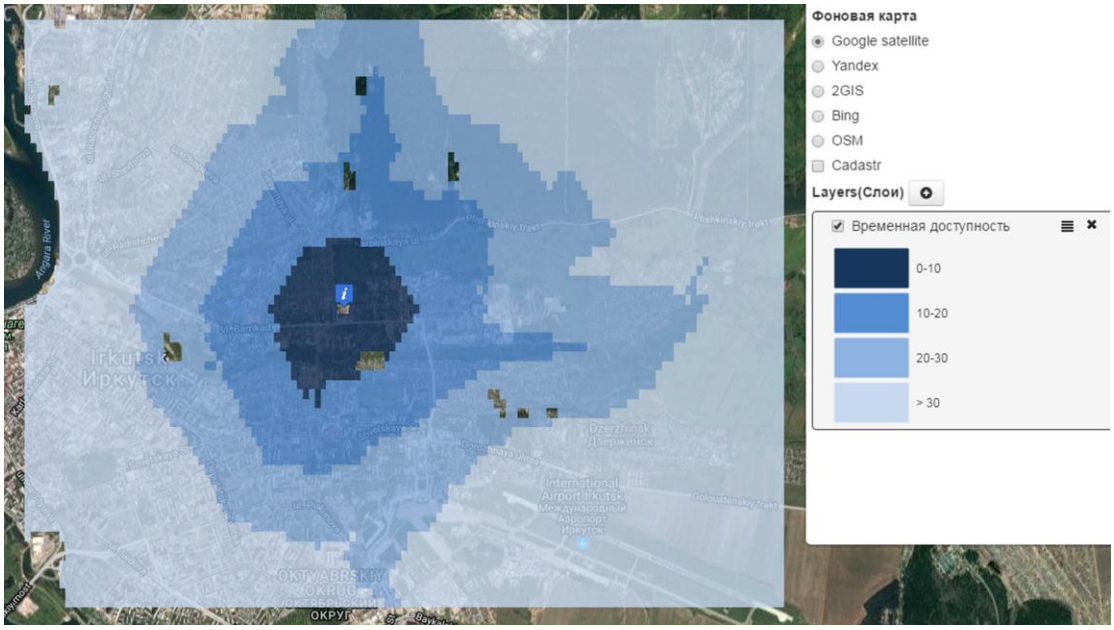

Рис. 41. Визуализация временной доступности отдельного объекта образовательной инфраструктуры

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). Данный подход позволяет использовать существующие программные библиотеки для выбранного языка написания сценариев.

4.4 Выводы

В данной главе были приведены три примера работы реализованной системы динамического выполнения композиций сервисов в гетерогенной среде, решающих задачи, возникшие при проведении междисциплинарных исследований. С помощью приведенных примеров были продемонстрированы основные возможности системы:

1. Задание сценариев с помощью процедурного языка программирования JavaScript позволило скрыть механизмы планирования и распараллеливания выполнения сервисов композиции от пользователя. Предложенный способ задания

композиций сервисов позволяет работать с результатами выполнения сервисов в теле сценария как с данными стандартных типов выбранного языка программирования, а также использовать для обработки большое количество уже существующих библиотек;

2. Планирование в условиях распределённой гетерогенной среды. Было продемонстрировано поведение системы в условиях отключения и включения вычислительных узлов, изменения времени выполнения отдельных сервисов. Был показан механизм перепланирования выполнения композиции сервисов;

3. Создание и выполнение композиций распределённых сервисов. Был показан интерфейс системы, упрощающий как задание композиций в виде сценариев на языке программирования JavaScript, так и предоставляющий средства для визуализации результатов выполнения сервисов и сценариев.

ЗАКЛЮЧЕНИЕ

Выполненная работа посвящена созданию системы динамического выполнения композиций сервисов в гетерогенной среде. В результате проведенного исследования были решены следующие задачи:

1. Изучены существующие средства организации и выполнения композиций распределенных сервисов, проанализированы методы адаптации процесса выполнения композиций к изменениям в гетерогенной среде распределенных сервисов;
2. Построена концептуальная модель динамического выполнения композиций в гетерогенной среде, определяющая основные процессы и взаимодействия между участниками и компонентами среды;
3. Разработан метод задания композиций распределенных сервисов в гетерогенной среде в виде программ на языке JavaScript, что позволило автоматизировать и скрыть механизмы планирования и распараллеливания выполнения сервисов от пользователя;
4. Разработаны методы планирования выполнения композиций сервисов в гетерогенной изменяющейся среде, устойчивые к происходящим в ней изменениям (отключение вычислительных узлов, изменение времени выполнения отдельных сервисов, ошибки при выполнении сервисов);
5. Реализовано инструментальное средство динамического выполнения композиций в гетерогенной среде, создана технология применения разработанного инструментального средства для организации междисциплинарных исследований;
6. Выполнена апробация разработанного программного средства и в целях демонстрации основных возможностей системы приведены три практические

задачи, возникшие в процессе проведения различных междисциплинарных исследований.

Таким образом, все задачи, поставленные перед исследованием, были успешно выполнены. Полученные в рамках диссертационной работы результаты успешно применяются при проведении различных междисциплинарных исследований, требующих использования вычислительных модулей в виде сервисов в сервис-ориентированной среде. Получены и выносятся на защиту следующие результаты:

1. Предложен метод разработки композиций сервисов в виде сценариев на языке программирования JavaScript. Метод был успешно реализован, поставленные перед методом задачи решены – автоматическое построение графа зависимостей сервисов по данным, реализация алгоритма диспетчеризации и минимизации времени выполнения сценария внутри интерпретатора сценариев, возможность обработки промежуточных результатов выполнения сервисов с помощью средств выбранного языка программирования, распространенность и простота самого языка, на котором реализуются сценарии композиций сервисов;
2. Реализовано планирование и выполнение композиций сервисов, заданных на процедурном языке программирования, в гетерогенной вычислительной изменяющейся среде. Выполнение композиций устойчиво к событиям, происходящих в вычислительной среде (изменение доступности вычислительных узлов, изменение времени выполнения отдельных сервисов, ошибки выполнения сервисов). Для автоматического распараллеливания обработки данных реализована программная модель MapReduce, а также реализована поддержка работы с табличными данными;
3. Реализована интернет-система динамического выполнения композиций сервисов в гетерогенной среде, позволяющая как создавать, так и выполнять

композиции сервисов для решения задач, возникающих в процессе проведения междисциплинарных исследований. Разработанной программное средство было апробировано в процессе исследований различной направленности.

Разработанная программная система ориентирована на работу в сервис-ориентированной среде распределённых сервисов. Дальнейшим развитием данной системы является увеличение спектра типов регистрируемых в системе сервисов (поддержка WSDL, REST), внедрение на основе технологий виртуализации масштабирования по требованию, реализация диспетчера выполнения сценариев в виде распределенного сервиса с RESTAPI, а также разработка и внедрение новых, более совершенных, подходов к планированию выполнения композиций сервисов в распределённой сервис-ориентированной среде.

СПИСОК ЛИТЕРАТУРЫ

1. Welke R., Hirschheim R., Schwarz A. Service-oriented architecture maturity // Computer (Long. Beach. Calif). 2011. Vol. 44, № 2. P. 61–67.

2. Hoffmann J., Weber I. Web Service Composition // Encyclopedia of Social Network Analysis and Mining. Springer-Verlag, 2014.

3. Тельнов Ю.Ф. Композиция сервисов и объектов знаний для формирования образовательных программ // Прикладная информатика. 2014. Vol. 1 (49). P. 75–81.

4. Сухорослов О.В. Реализация и композиция проблемно-ориентированных сервисов в среде Mathcloud // Вестник южно-уральского государственного университета. 2011. № 17. P. 101–112.

5. Foster I. What is the Grid ? A Three Point Checklist // GRID today. 2002. Vol. 1. P. 32–36.

6. Papazoglou M. Web Services: Principles and Technology // Technology. 2007. P. 784.

7. Pautasso C. RESTful Web service composition with BPEL for REST // Data Knowl. Eng. 2009. Vol. 68, № 9. P. 851–866.

8. Turner M., Budgen D., Brereton P. Turning software into a service // Computer (Long. Beach. Calif). 2003. Vol. 36, № 10. P. 38–44.

9. Budgen D., Brereton P., Turner M. Codifying a service architectural style // Computer Software and Applications Conference, 2004. COMPSAC 2004. Proceedings of the 28th Annual International. 2004. P. 16–22 vol.1.

10. Peltz C. Web services orchestration and choreography // Computer (Long. Beach.

Calif). 2003. Vol. 36, № 10. P. 46–52.

11. Ripon S., Uddin M.S., Barua A. Web Service Composition -- BPEL vs cCSP Process Algebra // 2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT). 2012. P. 150–155.

12. Горбунов-Посадов, М.М. Карпов В.Я., Корягин Д.А. Пакет Сафра: программное обеспечение вычислительного эксперимента // Алгоритмы и алгоритмические языки. Пакеты прикладных программ. Вычислительный эксперимент. Москва: Наука, 1983. 12-50 р.

13. Горбунов-Посадов М.М., Корягин Д.А., Мартынюк В.В. Системное обеспечение пакетов прикладных программ. Москва: Наука, 1990. 208 р.

14. Бычков И.В. et al. Сервис-ориентированное управление распределенными вычислениями на основе мультиагентных технологий // Известия Южного федерального университета. Технические науки. 2014. № 12. P. 17–27.

15. Boukhanovsky A. V., Krzhizhanovskaya V. V., Bubak M. Urgent computing for decision support in critical situations // Future Generation Computer Systems. 2018. Vol. 79. P. 111–113.

16. Smirnov P.A., Kovalchuk S. V., Boukhanovsky A. V. Knowledge-based support for complex systems exploration in distributed problem solving environments // Communications in Computer and Information Science. 2013. Vol. 394. P. 147–161.

17. Nasonov D., Butakov N. Hybrid scheduling algorithm in early warning systems // Procedia Computer Science. 2014. Vol. 29. P. 1677–1687.

18. Nasonov D. et al. Hybrid evolutionary workflow scheduling algorithm for dynamic heterogeneous distributed computational environment // J. Appl. Log. 2017. Vol. 24. P. 50–61.

19. Topcuoglu H., Hariri S., Wu M. Performance-effective and low-complexity task scheduling for heterogeneous computing // Parallel Distrib. Syst. …. 2002. Vol. 13, № 3. P. 260–274.

20. Topcuoglu H., Hariri S. Task scheduling algorithms for heterogeneous processors // Proceedings. Eighth Heterog. Comput. Work. 1999. P. 3–14.

21. Gupta S., Kumar V., Agarwal G. Task Scheduling in Multiprocessor System Using Genetic Algorithm // 2010 Second International Conference on Machine Learning and Computing. 2010. P. 267–271.

22. Thai L., Varghese B., Barker A. Task Scheduling on the Cloud with Hard Constraints // Proceedings - 2015 IEEE World Congress on Services, SERVICES 2015. 2015. P. 95–102.

23. Фёдоров Р.К. et al. Система планирования и выполнения композиций веб-сервисов в гетерогенной динамической среде // Вычислительные технологии. 2016. Vol. 21, № 6. P. 18–35.

24. Фёдоров Р.К., Шумилов А.С. Сценарий расчета временной доступности объектов образования // Вестник Бурятского государственного университета. 2017. Vol. 2. P. 20–32.

25. Фёдоров Р.К., Шумилов А.С., Авраменко Ю.В. Обработка векторных данных с помощью спецификаций в соответствии с моделью MapReduce // Вестник Бурятского государственного университета. 2017. Vol. 2. P. 12–19.

26. Фёдоров Р.К. et al. Компоненты среды WPS-сервисов обработки геоданных // Вестник Новосибирского государственного университета. Серия: информационные технологии. 2014. Vol. 12, № 3. P. 16–24.

27. Верхозина А.В. et al. Интернет-система ввода и редактирования

пространственных данных «Фарамант» // Вестник компьютерных и информационных технологий. 2015. № 9. P. 21–25.

28. Фёдоров Р.К., Шумилов А.С. Создание и публикация WPS-сервисов на основе облачной инфраструктуры // Вестник БГУ. 2015. № 4. P. 29–35.

29. Верхозина А.В. et al. Информационно-аналитическая система по фиторазнообразию Байкальской Сибири // Известия Иркутского государственного университета. Серия «Биология. Экология». 2016. Vol. 9, № 3. P. 11–28.

30. Bih J. Service oriented architecture (SOA) a new paradigm to implement dynamic e-business solutions // Ubiquity. 2006. Vol. 2006, № August. P. 1–1.

31. Foster I., Kesselman C., Tuecke S. The anatomy of the grid: Enabling scalable virtual organizations // Int. J. High Perform. Comput. Appl. 2001. Vol. 15, № 3. P. 200–222.

32. Радченко Г.И. Распределенные вычислительные системы. Челябинск: Фотохудожник, 2012. 184 p.

33. Набатов Д.Г. Проблемы межведомственного электронного взаимодействия // Труды Института государства и права Российской академии наук. 2013. № 2. P. 230–239.

34. Lin B. et al. Comparison between JSON and XML in Applications Based on AJAX // Proceedings - 2012 International Conference on Computer Science and Service System, CSSS 2012. 2012. P. 1174–1177.

35. Curbera F. et al. Unraveling the Web services Web: An introduction to SOAP, WSDL, and UDDI // IEEE Internet Comput. 2002. Vol. 6, № 2. P. 86–93.

36. Lawrence K. et al. Web Services Security: SOAP Message Security 1.1 (WS-

Security 2004) // Security. 2006. Vol. 2003, № February. P. 76.

37. Webber J. REST in practice // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2010. Vol. 6285 LNCS. P. 7.

38. Schut P. OpenGIS ® Web Processing Service // Open Geospatial Consort. 2007. № June. P. 1–3.

39. Garofalakis J. et al. Contemporary Web Service Discovery Mechanisms // J. Web Eng. 2006. Vol. 5, № 3. P. 265–290.

40. Nixon T. et al. Web Services Dynamic Discovery (WS- Discovery), version 1.1 [Electronic resource] // OASIS. 2009. P. 1–50. URL: http://docs.oasis-open.org/ws-dd/discovery/1.1/os/wsdd-discovery-1.1-spec-os.pdf.

41. Brooks F. No Silver Bullet: Essence and Accident of Software Engineering // IEEE Softw. 1987. Vol. 20. P. 12.

42. Ouyang C. et al. Translating bpmn to bpel // BPM Cent. Rep. BPM-06-02, BPMcenter. org. 2006. P. 1–22.

43. Volkov S.Y., Sukhoroslov O.V. Running Parameter Sweep applications on Everest cloud platform // Comput. Res. Model. 2015. Vol. 7, № 3. P. 601–606.

44. Xie G. et al. A High-Performance DAG Task Scheduling Algorithm for Heterogeneous Networked Embedded Systems // 2014 IEEE 28th Int. Conf. Adv. Inf. Netw. Appl. 2014. P. 1011–1016.

45. Kwok Y.-K., Ahmad I. Static scheduling algorithms for allocating directed task graphs to multiprocessors // ACM Comput. Surv. 1999. Vol. 31, № 4. P. 406–471.

46. Munir E.. et al. SDBATS: A Novel Algorithm for Task Scheduling in Heterogeneous

Computing Systems // Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International. 2013. P. 43–53.

47. Arabnejad H., Barbosa J.G. List scheduling algorithm for heterogeneous systems by an optimistic cost table // IEEE Trans. Parallel Distrib. Syst. 2014. Vol. 25, № 3. P. 682–694.

48. Wang G., Guo H., Wang Y. A novel heterogeneous scheduling algorithm with improved task priority // Proceedings - 2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security and 2015 IEEE 12th International Conference on Embedded Software and Systems, H. 2015. P. 1826–1831.

49. Canon L.-C. et al. Comparative Evaluation Of The Robustness Of DAG Scheduling Heuristics // Grid Comput. 2008. P. 73–84.

50. Dorigo M., Stützle T. Ant Colony Optimization // Intelligence Magazine IEEE. 2004. 319 p.

51. Bertsimas D., Tsitsiklis J. Simulated Annealing // Stat. Sci. 1993. Vol. 8, № 1. P. 10–15.

52. Kennedy J., Eberhart R. Particle swarm optimization // Neural Networks, 1995. Proceedings., IEEE Int. Conf. 1995. Vol. 4. P. 1942–1948 vol.4.

53. Claessen K. et al. SAT-solving in practice // Proceedings - 9th International Workshop on Discrete Event Systems, WODES' 08. 2008. P. 61–67.

54. Kozen D., Tseng W.L.D. The Böhm-Jacopini theorem is false, propositionally // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). 2008. Vol. 5133 LNCS. P. 177–192.

55. Dijkstra E.W. Letters to the editor: go to statement considered harmful // Commun. ACM. 1968. Vol. 11, № 3. P. 147–148.

56. mills chris david. JavaScript | MDN [Electronic resource] // Mozilla developer. 2017. P. 2. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

57. Chrome V8 [Electronic resource]. URL: https://developers.google.com/v8/.

58. Dean J., Ghemawat S. MapReduce // Commun. ACM. 2010. Vol. 53, № 1. P. 72.

59. Авраменко Ю.В., Шумилов А.С. Спецификации методов разбиения и сборки растровых изображений в рамках программной модели MapReduce // География и природные ресурсы. 2016. № 6. P. 156–159.

60. Шумилов А.С., Авраменко Ю.В. Спецификация распараллеливания обработки векторных данных в модели MapReduce // Информационные и математические технологии в науке и управлении. 2017. Vol. 4. P. 71–79.

61. Fenoy G., Bozon N., Raghavan V. ZOO-Project: The open WPS platform // Appl. Geomatics. 2013. Vol. 5, № 1. P. 19–24.

62. Breu F., Guggenbichler S., Wollmann J. OpenStack Cloud Computing Cookbook // Vasa. 2008.

ПРИЛОЖЕНИЕ 1. СВИДЕТЕЛЬСТВА О ГОСУДАРСТВЕННОЙ РЕГИСТРАЦИИ ПРОГРАММЫ ДЛЯ ЭВМ

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2014610274

Интернет-система ввода и редактирования пространственных данных «Фарамант»

Правообладатель: *Федеральное государственное бюджетное учреждение науки Институт динамики систем и теории управления Сибирского отделения Российской академии наук (ИДСТУ СО РАН) (RU)*

Авторы: *Фёдоров Роман Константинович (RU), Ружников Геннадий Михайлович (RU), Шумилов Александр Сергеевич (RU), Ветров Александр Анатольевич (RU), Михайлов Андрей Анатольевич (RU)*

Заявка № 2013660173
Дата поступления 06 ноября 2013 г.
Дата государственной регистрации
в Реестре программ для ЭВМ *09 января 2014 г.*

Руководитель Федеральной службы по интеллектуальной собственности

Б.П. Симонов

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2016663724

Система планирования и выполнения композиций веб-сервисов в гетерогенной динамической среде

Правообладатель: ***Федеральное государственное бюджетное учреждение науки Институт динамики систем и теории управления имени В.М. МАТРОСОВА Сибирского отделения Российской академии наук (RU)***

Авторы: ***Фёдоров Роман Константинович (RU), Шумилов Александр Сергеевич (RU), Бычков Игорь Вячеславович (RU), Ружников Геннадий Михайлович (RU)***

Заявка № **2016660937**
Дата поступления **18 октября 2016 г.**
Дата государственной регистрации
в Реестре программ для ЭВМ ***15 декабря 2016 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев

РОССИЙСКАЯ ФЕДЕРАЦИЯ

СВИДЕТЕЛЬСТВО

о государственной регистрации программы для ЭВМ

№ 2017617913

Среда выполнения сервисов и их сценариев

Правообладатель: ***Федеральное государственное бюджетное учреждение науки Институт динамики систем и теории управления имени В.М. Матросова Сибирского отделения Российской академии наук (RU)***

Авторы: ***Фёдоров Роман Константинович (RU), Шумилов Александр Сергеевич (RU), Бычков Игорь Вячеславович (RU), Ружников Геннадий Михайлович (RU)***

Заявка № **2017613155**

Дата поступления **10 апреля 2017 г.**

Дата государственной регистрации в Реестре программ для ЭВМ ***17 июля 2017 г.***

Руководитель Федеральной службы по интеллектуальной собственности

Г.П. Ивлиев