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

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

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

[2]Иркутский научный центр СО РАН, Россия

[*]Контактный e-mail: alexshumilov@yahoo.com

Рассмотрено расширение сервис-ориентированной архитектуры вычислений, в которой вычислительные модули реализованы на узлах различной архитектуры и доступны через Интернет (гетерогенную вычислительную среду). Естественным развитием данной архитектуры является композиция сервисов, т. е. их объединение для решения задач, в условиях динамической вычислительной среды — добавление/выключение узлов, изменение их загруженности. Для выполнения композиций распределенных сервисов разработана и реализована система планирования и выполнения композиций веб-сервисов, позволяющая учитывать требования гетерогенной динамической среды. Система апробирована и интегрирована в Портал ИДСТУ СО РАН.

Ключевые слова: сервис-ориентированная архитектура, композиция сервисов, JavaScript, оркестрация сервисов.

Ведение

В последнее время активно развивается сервис-ориентированная архитектура вычислений (SOA — Service-Oriented Architecture) [1], которая предполагает представление программных компонентов в виде отдельных вычислительных сервисов. Основные преимущества данного подхода — простота использования сервисов (сложность реализации сервиса полностью скрывается за его интерфейсом), сравнительно легкие тестирование и отладка (так как тестирование производится для каждого сервиса отдельно), масштабируемость и возможность повторного использования. Самые распространенные стандарты интерфейсов доступа и описания сервисов — SOAP (Simple Object Access Protocol) [2], RPC (Remote Procedure Call) [3], REST (Representational State Transfer) [4].

В области обработки пространственных данных все больше вычислений производится с помощью сервисов, в том числе и удаленных, т. е. веб-сервисов, развернутых на разных вычислительных узлах. Например, активно используются сервисы геокодирования, которые позволяют по адресу получать координаты, и сервисы построения маршрутов, дающие возможность определить кратчайший путь с учетом дорожной сети, их покрытия и т. д. При использовании таких сервисов нет необходимости устанавливать программное обеспечение, вести и актуализировать базы адресов или транспортной сети.

Например, программный продукт QGIS позволяет вызывать и использовать результаты работы таких удаленных тематических сервисов. Говоря о стандартах сервисов обработки пространственных данных, необходимо упомянуть о разработанном концерном OGC (Open Geospatial Consortium) стандарте WPS (Web Processing Service) [5], который определяет механизм описания, выполнения и контроля сервисов, а также формат передачи пространственных данных. Длительность выполнения сервисов, базирующихся на WPS, не ограничена. Стандарт основан на формате данных XML, в качестве среды передачи обычно используется протокол HTTP.

Основные преимущества SOA наиболее выгодно проявляются при использовании композиций сервисов, где решение сложной задачи обеспечивается взаимодействием сервисов и обменом данными между ними. Существует несколько подходов к заданию композиций сервисов, которые сводятся к определению графа зависимостей вызовов сервисов.

При выполнении композиций распределенных сервисов возникает ряд особенностей. В частности, каждый сервис композиции может выполняться на различных вычислительных узлах с разной производительностью и пропускной способностью каналов связи. Количество выделяемых узлов может динамически меняться. Время выполнения сервисов может отличаться от ожидаемого, также сервисы могут прекращать работу вследствие сбоев (получения некорректных данных, программной ошибки и т. д.). В зависимости от решаемой задачи набор выполняемых сервисов композиции может динамически изменяться. Поэтому при выполнении композиций распределенных сервисов актуальна задача их планирования с учетом гетерогенной динамически изменяющейся среды. Планирование может производиться с целью минимизации или времени их выполнения, или используемых ресурсов. Для решения данной задачи разработана и реализована система выполнения композиций веб-сервисов, позволяющая динамически подстраиваться под изменяющееся состояние вычислительной среды. Система была апробирована и интегрирована в инфраструктуру Портала ИДСТУ СО РАН (http://geos.icc.ru).

1. Обзор средств планирования и выполнения композиций сервисов

Композицией называется набор сервисов с определенными между ними зависимостями, решающий какую-либо задачу или автоматизирующий какой-либо процесс [7]. Выполнение сервиса в контексте теории расписаний называется заданием (требованием). Общепризнанным стандартом является определение зависимостей между заданиями с помощью направленного ациклического графа (Directed Acyclic Graph — DAG), в котором вершинами являются сами задания, а ребра показывают зависимости между заданиями по данным [8, 9].

Проблема составления расписания на основе DAG обсуждается уже долгое время, первые работы велись еще в 70-х годах прошлого века. Данная проблема — нахождение такого расписания для набора заданий, которое бы минимизировало время выполнения всего DAG, — является NP-полной задачей, за исключением нескольких простейших случаев [10]. Таким образом, существующие методы находят только приближенные решения. Все алгоритмы по способу нахождения расписания для DAG можно разделить на две группы — эвристические и метаэвристические.

Один из наиболее популярных эвристических алгоритмов составления расписания — Heterogeneous Earliest Finish Time (HEFT) [11], который является списковым алгоритмом, т. е. он сортирует задания на основании заранее вычисленной величины и назначает их по порядку в списке на менее загруженный на момент назначения узел. Величина для каждого задания определяется суммой среднего времени его выполнения на всех вычислительных узлах и наибольшим значением эвристики для узлов, напрямую зависящих от данного задания.

Среди метаэвристических алгоритмов наибольшей популярностью пользуется семейство генетических алгоритмов. Они позволяют находить приближенные решения из широкого пространства поиска, применяя эволюционные принципы. Генетические алгоритмы обычно получают более точные решения, нежели эвристические, но проигрывают по времени выполнения [12].

Стоит отметить, что большинство алгоритмов планирования разрабатывалось с расчетом на использование в высокопроизводительных вычислениях, когда вычислительные узлы почти гарантированно способны выполнять сервисы и их характеристики, а также степень занятости заранее известны. С развитием сервис-ориентированного подхода характеристики среды, в которой работают данные алгоритмы, изменились, так как большинство сред стали гетерогенными.

Все чаще появляются гибридные алгоритмы, объединяющие в себе разные техники. Например, алгоритм Hybrid Evolutionary Workflow Scheduling Algorithm for Dynamic Heterogeneous Distributed Computational Environment [12] сочетает в себе HEFT и генетический алгоритм. Данный алгоритм способен адаптироваться к изменениям вычислительной среды, что особенно важно в распределенных гетерогенных вычислительных средах, когда работа отдельных вычислительных узлов и каналов передачи данных не гарантируется, а также когда время выполнения сервисов и передачи данных между ними изменяется по мере работы сервисов. Этот 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.

Говоря о подходах к заданию композиций сервисов, необходимо рассмотреть наиболее известные два способа их задания — BPEL (Business Process Execution Language) [13] и XPDL (XML Process Definition Language) [14]. Оба способа дескриптивно задают граф зависимостей заданий DAG. Несмотря на то, что стандарт XPDL был принят раньше, чаще всего в работе используется стандарт BPEL, так как BPEL более приспособлен для описания взаимодействия сервисов. В свою очередь, он позволяет использовать сервисы, не имеющие веб-интерфейс, а также дает возможность определять графическое представление заданного взаимодействия.

Один из наиболее популярных программных пакетов, позволяющих задавать композиции сервисов с помощью формата BPEL, — Oracle BPEL Process Manager [14]. Данный программный продукт предоставляет удобный графический интерфейс для создания композиций сервисов и средство отладки композиций, поддерживает большое количество форматов интерфейса веб-сервисов (WSDL, WPS и т. д.). BPEL Process Manager не осуществляет планирование выполнения сервисов в композиции, что критично при сложных композициях сервисов с большим количеством участвующих вычислительных узлов.

Для задания композиций сервисов также используется программный продукт Taverna [16], позволяющий с помощью графического интерфейса определять последовательности выполнения заданий. В отличие от императивного BPEL с явным заданием процесса выполнения, Taverna использует функциональную модель с управлением данными. Taverna распространена в научной среде и часто используется в астрономии, биоинформатике (например, для определения генов, ответственных за какое-либо заболевание [17]). Однако Taverna не осуществляет планирование выполнения композиций в сложных средах.

Таким образом, можно отметить, что существуют подходы к заданию композиции сервисов, где граф зависимостей заданий DAG полностью определяется до момента выполнения, и разработаны алгоритмы составления расписания для таких DAG в целях обеспечения приближенного к оптимальному времени выполнения. Однако отсутствуют алгоритмы и подходы, позволяющие динамически в процессе выполнения определять граф зависимостей заданий DAG и осуществлять планирование его выполнения с учетом особенностей гетерогенных динамических сред распределенных вычислений.

2. Описание гетерогенной динамической среды

Структура гетерогенной динамической среды, в рамках которой производится выполнение композиции сервисов, приведена на рис. 1. Каталог сервисов позволяет регистрировать и запускать сервисы на Портале, а также хранить информацию о них (данные о вычислительных узлах, поддерживающих зарегистрированный сервис, описания входных и результирующих параметров сервиса). Каталог хранит информацию о сервисах, реализованных как на удаленных вычислительных узлах, так и на узлах, развернu-

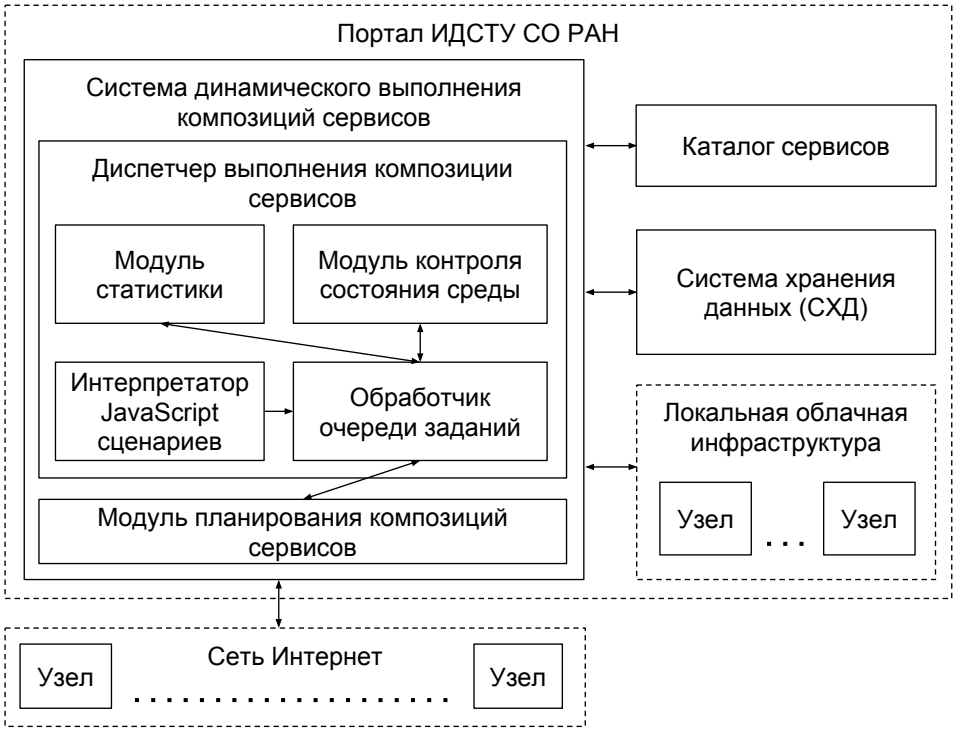

Рис. 1. Структура среды

тых в пределах локальной облачной инфраструктуры — наборе виртуальных машин, реализующих различные геоинформационные веб-сервисы. Система хранения данных (СХД) позволяет пользователям Портала загружать, хранить и использовать файлы в качестве входных параметров сервисов. Также СХД обеспечивает обмен файлами между запускаемыми сервисами. Составные части системы динамического выполнения композиций сервисов рассмотрены в разд. 6.

Выполнение композиции сервисов производится на ограниченном множестве узлов. Количество узлов облачной инфраструктуры может динамически изменяться. Таким образом, требования, предъявляемые к разрабатываемой системе, следующие:

- граф зависимостей заданий DAG композиции сервисов должен модифицироваться в процессе выполнения;
- планирование должно обеспечивать приемлемое время выполнения композиций;
- во время выполнения композиции система должна обрабатывать происходящие изменения в среде, будь то отказ или восстановление вычислительных узлов, изменения ожидаемого времени выполнения заданий, неполадки в сетевой инфраструктуре и т. д., т. е. изменять текущее расписание с учетом уже выполняющихся и выполнившихся заданий.

3. Определение композиции сервисов с помощью JavaScript

В ИДСТУ СО РАН разработан и реализован способ задания композиции сервисов с помощью языка программирования JavaScript [18]. Композиция сервисов представляет собой программный код, в котором вызов сервисов осуществляется с помощью специальных функций, закрепленных за каждым сервисом. Применение процедурного языка для определения композиции сервисов позволяет задавать сложные алгоритмы с использованием ветвлений, циклов, рекурсий и других конструкций языка программирования JavaScript.

Участвующие в сценариях сервисы должны быть предварительно зарегистрированы в каталоге Портала и могут находиться на любом вычислительном узле независимо от его месторасположения и характеристик. Таким образом, известны только сетевое расположение и описание параметров сервисов, а также характеристики самих вычислительных узлов. Пример композиции представлен ниже.

```
function test_scenario(input, resultStore) {
        var result1 = new ValueStore();
        var result2 = new ValueStore();
        vectorToGrid({in1: input.a}, {result: result1});
        roadToGrid({in1: input.b}, {result: result2});
        raster_sum({in1: result1, in2: result2},
        {result: resultStore.my_scenario_result});
}
```

Композиция вычисляет массу выделяемых загрязняющих веществ на регулярной сетке местности с помощью WPS-сервисов vectorToGrid, roadToGrid и raster_sum (подробнее о данном сценарии см. в разд. 8).

Граф зависимостей заданий формируется динамически в процессе выполнения JavaScript кода и может зависеть от начальных данных или промежуточных результатов. Для вызова сервисов используются так называемые функции-обертки (в слу-

чае приведенного выше сценария это функции vectorToGrid, roadToGrid и raster_sum), каждая из которых соответствует определенному сервису. Во время вызова функций-оберток создаются вершины графа, т. е. регистрируются задания. Для определения зависимостей между заданиями используются объекты класса ValueStore, которые передаются в качестве параметров функций сервисов. Объекты класса ValueStore необходимы для однозначной идентификации передаваемых данных при вызове функций сервисов. Вызовы специальных функций не блокируют выполнение JavaScript-кода. Если данные еще не вычислены, блокировка сценария производится только в случае вызова метода Get объекта ValueStore, с помощью которого можно получить значение получаемых данных. Представленный выше пример сценария начинает выполнение и заканчивает работу без ожидания результатов работы сервисов.

Применение процедурного языка программирования для определения DAG имеет ряд преимуществ:

- для большинства программистов это привычный язык разработки алгоритмов, так как применение сервисов не отличается от использования различных программных библиотек;
- наличие готовых алгоритмов, которые можно применить в этом подходе;
- определение зависимостей заданий происходит динамически.

4. Постановка задачи планирования

Задача поиска расписания является NP-полной, т. е. нахождение оптимального решения в некоторых случаях для такого рода задач практически невыполнимо. Таким образом, осуществляется поиск приемлемого решения, которое максимально приближено к оптимальному.

Представим задачу в виде математической модели *M = (N, T, depends, busy, executable, finishTime)*:

- множество вычислительных узлов $N = \{n_{1..k}\}$, где — n_i вычислительные узлы, k — их количество;
- множество заданий $T = \{t_i\}_{i=1..M}$, где $t_1..t_m$ — задания, m — их количество; $c_e(t_i)$ — время выполнения i-го задания на любом поддерживающем его узле, $c_t(t_i, t_j)$ — время передачи результатов работы i-го задания на другой узел для выполнения j-го задания (т. е. если t_i и t_j выполняются на одном узле, то $c_t(t_i, t_j) = 0$);
- отношение предшествования $depends(t_i, t_j)$, которое определяет, зависимо ли задание t_i от t_j;
- ациклический направленный граф зависимостей заданий по данным, вершины графа — задания, ребра — отношения зависимости между заданиями (например, t_i не может начать свое выполнение, пока не закончилось задание t_j);
- отношение $busy(n_i)$, показывающее время высвобождения узла с учетом назначенных на него заданий;
- отношение $executable(t_i, n_j)$, показывающее, может ли задание t_i выполняться на узле n_j;
- существует некоторое подмножество T_{start} множества T, содержащее начальные задания, т. е. не существует таких заданий t_i, что выполняется $depends(t_{start}, t_i)$;
- существует некоторое подмножество T_{exit} множества T, содержащее выходные задания графа, т. е. не существует таких заданий t_i, что выполняется $depends(t_i, t_{exit})$.

Определим $P(T, N)$ как некое расписание выполнения заданий, назначенных на какие-либо узлы. Расписание $P(T, N)$ является набором пар вида $(t, n)_j$, где t — задание, n — вычислительный узел, на который задание назначается, j определяет, каким по порядку из назначенных вызовов сервисов на n является t. Множеством допустимых расписаний называется набор всех расписаний для заданий T и вычислительных узлов N, для которых выполняется отношение предшествования заданий.

Определим функцию $evaluate(P) = \max_{n_i \in N}\{busy(n_i)\}$, показывающую 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.

5. Реализация и модификация алгоритма планирования

Для нахождения расписания, обеспечивающего приближенное время выполнения композиции заданий, используется списковый алгоритм составления расписания HEFT, реализующий метод поиска в глубину. Особенностями алгоритма, служащими для ускорения нахождения приемлемого расписания, являются сортировка заданий, узлов и отсечение неперспективных ветвей графа поиска решения. Рассмотрим каждую технику в отдельности на примере графа, где заданы идентификаторы задач, время выполнения заданий и стоимость передачи данных для каждого задания (рис. 2).

Определим функцию $aggregatedCost(t)$, вычисляющую минимальную оценку времени выполнения задания t и всех зависимых от нее заданий. Оценка вычисляется путем прохождения графа, начиная с выходного задания из T_{exit} до самого задания t:

$$\text{если } \; t_i \in T_{exit} \; \text{ то } \; aggregatedCost(t_i) = w_e(t_i);$$
$$\text{иначе } \; aggregatedCost(t_i) = w_e(t_i) + \max_{t_j \in D}(aggregatedCost(t_j)),$$

где w — среднее время выполнения задания на всех поддерживающих его вычислительных узлах; D — множество заданий, зависимых от t_i.

Построение расписания выполнения композиции сервисов производится за несколько шагов:

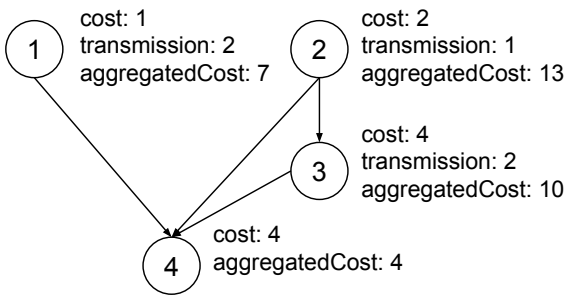

Рис. 2. Граф зависимости заданий

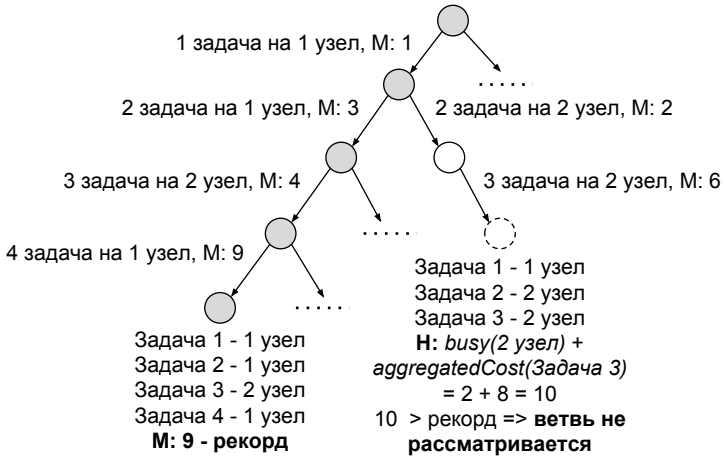

Рис. 3. Отсечение ветвей при выполнении алгоритма

- Перед началом работы алгоритма производится сортировка заданий по убыванию значения функции $aggregatedCost(t)$ происходит перед началом работы алгоритма. Таким образом, алгоритм будет поочередно выбирать задания, начиная с самого затратного по времени задания.
- Внутри самого алгоритма после каждого назначения задания на определенный узел происходит сортировка узлов по возрастанию значения их занятости. Данная сортировка позволяет сначала рассматривать случаи, когда самое затратное по времени задание назначается на самый свободный узел.
- В то время как первые две техники позволяют находить приближенное расписание как можно быстрее, техника отсечения ветвей графа дает возможность сократить пространство перебора решений. Данная техника реализуется с помощью алгоритма 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 приемлемого решения). Функция эвристики складывается из текущего значения загруженности узла и значения $aggregatedCost(t)$ для текущего задания. Пример отсечения ветви представлен на рис. 3.
- 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, каждый из которых является назначением какого-то сервиса t_i на узел n_j.

6. Алгоритмы динамического изменения расписания

Учитывая специфику среды распределенных сервисов, необходимо создать алгоритм построения расписаний выполнения заданий, который должен обрабатывать наступающие в динамической вычислительной среде следующие события:

- *Удаление или добавление вычислительного узла.* Во время выполнения композиции вычислительные узлы выходят из строя и/или возвращаются в рабочее состояние. Если выходит из строя вычислительный узел, то все задания, выполняющиеся на нем, 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 нем (при условии, что результаты выполнения заданий остаются доступными). 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. 4 показана ситуация, когда узел с идентификатором 0 отказывает во время выполнения на нем 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).
- *Добавление задания в DAG.* Во время выполнения композиции по мере интерпретации ее сценария добавляются новые задания. Перестроение всего расписания происходит, когда добавляемое задание изменяет прежде рассчитанное время выполнения всего сценария. В случае, когда добавленное задание при назначении на самый свободный узел не меняет общее время выполнения сценария, перестроение плана не происходит и производится запуск задания на выполнение. На рис. 5 в первом случае задание 5 при назначении на самый свободный вычислительный узел не изменяет время выполнения расписания, перестроение не происходит. Во втором случае после добавления задания 5 время выполнения расписания увеличивается, следовательно, необходимо выполнить перерасчет.
- *Завершенное задание выполнялось меньше ожидаемого времени.* Если разница между фактическим и ожидаемым временем меньше значения L, то перестроение не производится, так как оно потребует больше времени, нежели получившийся по времени выполнения данного задания выигрыш. На рис. 6 задание 3 завершило свое выполнение, но раньше, чем предполагалось, — штриховая линия показыва-

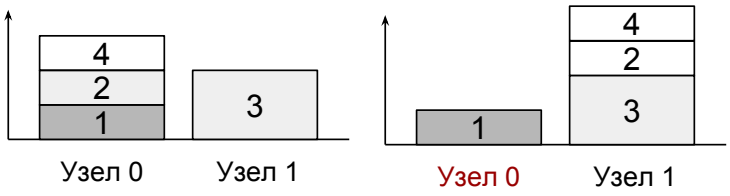

Рис. 4. Отказ узла во время выполнения композиции

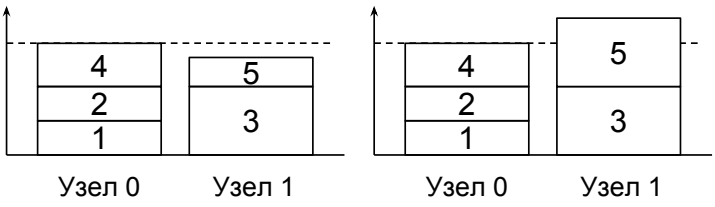

Рис. 5. Добавление задания в план

ет 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).

- *Выполняющееся задание не закончилось в ожидаемый срок.* Происходит перестроение расписания с предположением, что время рассматриваемого задания становится $t = (1 + d) \cdot c(t) + L$, где d — некоторая константа от 0 до 1. На рис. 7 отображена ситуация, когда задание 3 выполняется дольше, чем планировалось. Левая схема показывает изначальное расписание, правая схема — перестроенное расписание с добавленным для задания 3 ожидаемым временем выполнения.
- *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 очередь 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. 8 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.

В случае перестроения расписания на основе уже частично выполняющегося расписания $P(t, n)$ воспроизведение уже выполняющейся его части 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 его в стандартном режиме.

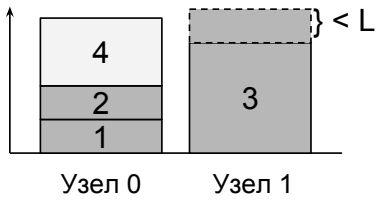

Рис. 6. Задание выполнялось меньше ожидаемого времени

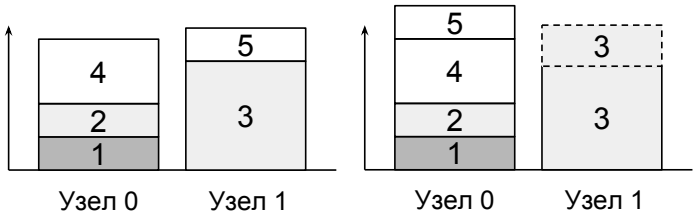

Рис. 7. Задание выполняется дольше, чем ожидалось

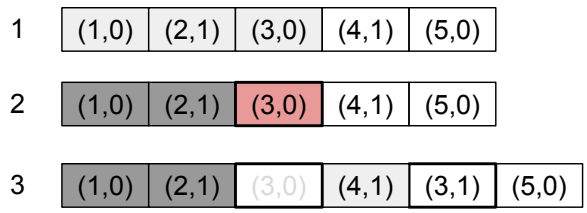

Рис. 8. Завершение задания с ошибкой

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

7. Система планирования и выполнения композиций сервисов

Система планирования и выполнения композиций сервисов реализована в виде отдельного приложения, встраиваемого в Портал. Архитектура системы динамического выполнения композиции сервисов приведена на рис. 1. Портал реализует функции хранения информации об участвующих в композиции сервисах — их типах, сетевом расположении, характеристиках вычислительных узлов, на которых они могут располагаться. Сервисы регистрируются на Портале пользователями. Портал предоставляет удобный интерфейс для ввода параметров запускаемых композиций, среди них могут быть как текстовые данные, так и файлы из СХД Портала, результаты выгрузки данных из реляционных таблиц, хранимых на Портале, и другие типы данных. При запуске диспетчера Портал 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 программный JavaScript-код компилируется с помощью библиотеки проекта Google V8 (JavaScript движок с открытым исходным кодом) и запускается на выполнение. По мере выполнения скрипта вызываются функции-обертки с различными параметрами, соответствующие задания помещаются в глобальную очередь диспетчера. С момента запуска внутри приложения диспетчера запускается отдельный поток выполнения, который с определенной периодичностью опрашивает глобальную очередь на предмет изменения состава или статуса зарегистрированных в ней заданий. При наступлении определенных событий, рассмотренных в разд. 3, происходит перестроение расписания с учетом уже выполняющихся заданий. Для перестроения расписания необходимо также знать время выполнения заданий на определенных узлах и состояние вычислительных узлов, выполняющих хотя бы один сервис из участвующих в композиции.

Диспетчер хранит информацию о состоянии вычислительных узлов, на которых выполняются сервисы, с помощью модуля контроля состояния среды распределенных 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, учитывающих возникшие в процессе выполнения композиции сбои в работе среды распределенных сервисов.

Опрос состояния длительных сервисов осуществляется с определенной периодичностью в отдельных потоках выполнения. 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: сетевому адресу сервиса и идентификатору 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. С этой целью 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 b записей, где b — задаваемая константа. 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 делается предположение относительно его длительности на основании заранее заданных значений, зависящих от типа сервиса (является ли это длительным сервисом или нет).

8. Апробация

В качестве апробации системы динамического выполнения композиции сервисов в распределенной среде приводятся две задачи.

Задача 1. В целях апробации устойчивости работы композиции сервисов к отключениям и добавлениям узлов в гетерогенной вычислительной среде решена задача поиска наиболее подходящих растров для классификации видов растительности методом опорных векторов. Существует некий набор векторных данных, содержащих прецеденты для разных видов ландшафта — степь, сосновый лес, болото, пихтовый лес, березовый лес, лиственничный лес, еловый лес, всего семь видов ландшафта. Данный набор данных предоставлен специалистами предметной области. Существует метод SVM_Learn, реализованный в виде WPS-сервиса авторами работы и развернутый на четырех серверах, работающих под управлением ОС Windows Server 2008. Данный сервис принимает на вход набор растров, SHP-файл с прецедентами, название класса и значения класса для разделения прецедентов. Результатом работы сервиса является файл классификатора, а также значение точности найденной модели классификации.

Всего для анализа имеется пять растровых файлов, предоставленных специалистами предметной области. Композиция сервисов в виде сценария на языке JavaScript должна перебирать все возможные комбинации растров для каждого сочетания имеющихся прецедентов. Для каждой комбинации прецедентов должна избираться 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 отключаться, процесс выполнения не должен останавливаться.

Сокращенный код JavaScript сценария представлен ниже. Переменная classes (сноска 1 в приведенном 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 (сноска 2 в приведенном ниже коде сценария). SVM_Learn_from_shp представляет собой функцию-обертку для сервиса SVM. Далее, начиная с (сноска 3 в приведенном 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: 3, name: 'Step'},[1]
        {id: 4, name: 'Sosnovyj les'},
        {id: 5, name: 'Boloto'}];

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

                print('Best precision is ' + ...);[3]
        }
}
```

В самом начале выполнения композиции сервисов пользователю необходимо ввести ее параметры. Портал предоставляет необходимый интерфейс для ввода параметров сервисов или композиции сервисов, позволяя как вводить 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 указываются 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. 9 приведено расписание выполняющихся заданий до отключения узлов и после (светло-серым цветом выделены уже выполняющиеся сервисы, темно-серым — выполненные).

При отказе двух узлов (выключается Apache вместе с WPS-службой на узлах Node 0 и Node 1) диспетчер обрабатывает данную ситуацию и обновляет расписание выполнения (удаляя выключенные вычислительные узлы из расписания и перераспределяя нагрузку).

По завершении выполнения композиции сервисов в пользовательскую консоль выводится результат работы. За все время выполнения композиции сервисов участвующие узлы были равномерно загружены и работающие до конца выполнения узлы Node 2 и

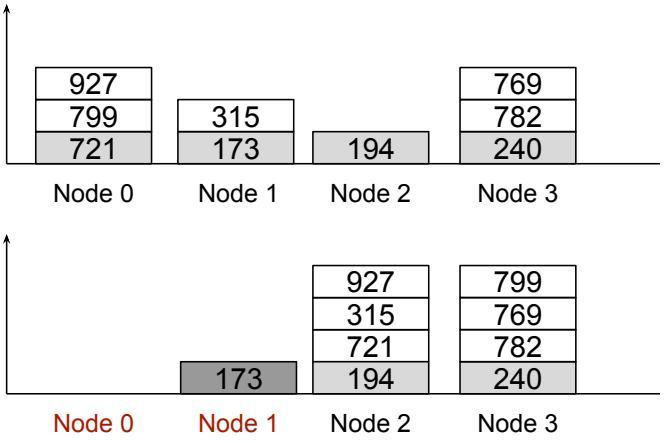

Рис. 9. Расписание до и после отключения узлов

Node 3 завершили свою работу практически одновременно с разницей в 4 с. Результаты работы сервисов (файлы моделей классификации, которые могут быть впоследствии использованы) автоматически были загружены на СХД Портала. Пример результата выполнения приведен ниже.

```
Skipped decisions: 0, taken decisions: 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 Listv. and Elovyj les is 0.67
Best precision grids: N.tif, E.tif, A.tif
```

Задача 2. В целях апробации предложенного подхода к планированию выполнения композиции с зависимостями между сервисами по данным выбрана задача расчета загрязнения тремя парниковыми газами для определенного региона.

В сценарии используются три сервиса — vectorToGrid, roadToGrid, raster_sum, в вычислениях участвуют четыре 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 — для расчета выделяемых загрязнений от автотранспорта в ячейках регулярной сетки, сервис raster_sum — для суммирования выделяемых загрязнений от точечных источников и автотранспорта.

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

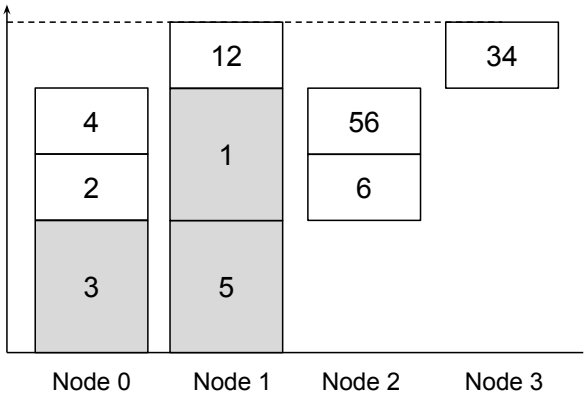

Рис. 10. Составленное расписание

На рис. 10 показано составленное для сценария расписание. Серым цветом обозначены вызовы сервиса vectorToGrid, выполняющегося дольше всех. Составленное расписание учитывает выполнимость сервиса vectorToGrid на определенных узлах, т. е. первые два узла практически освобождаются от выполнения каких-либо других сервисов, пока не завершены вызовы сервиса vectorToGrid.

Заключение

Спроектирована и разработана, а также интегрирована в существующую инфраструктуру Портала ИДСТУ СО РАН система планирования и выполнения композиций сервисов в гетерогенной динамической среде. Основными преимуществами разработанной системы являются ее способность к перестройке расписания вызова сервисов при изменениях состояния гетерогенной среды, а именно к отключению или, наоборот, включению отдельных вычислительных узлов в процессе выполнения композиции, изменению временных характеристик вызываемых сервисов. Система производит планирование с учетом динамического изменения графа зависимостей заданий, который формируется при выполнении JavaScript кода. Перестроение расписания происходит с учетом частично выполняющегося расписания, что обеспечивает его актуальность. Задание композиций сервисов с помощью JavaScript позволяет пользователю не задумываться о планировании и выполнении сервисов, синхронизации выполнения сервисов и передачи данных между ними, что упрощает процесс разработки композиций сервисов.

В результате апробаций, описанных в данной статье, а также апробаций, решающих другие задачи, отмечено, что система планирования и выполнения композиций распределенных сервисов осуществляет планирование выполнения сервисов в соответствии с возложенными на нее требованиями. Таким образом представленный подход автоматически решает множество задач асинхронного выполнения композиции сервисов, позволяя программисту работать в привычной среде.

Благодарности. Работа выполнена при финансовой поддержке РФФИ (гранты № 16-07-00411_а, 16-37-00110 мол_а, 14-07-00166_а), при поддержке Совета по грантам Президента Российской Федерации для государственной поддержки ведущих научных школ РФ (НШ-8081.2016.9).

Представленное программное обеспечение реализовано с помощью ресурсов Центра коллективного пользования Интегрированная информационно-вычислительная сеть ИНЦ ФАНО (Телекоммуникационный центр коллективного пользования “Интегрированная информационно-вычислительная сеть Иркутского научно-образовательного комплекса” (ЦКП ИИВС ИРНОК)) (http://net.icc.ru).

Список литературы / References

[1] **Emig, C., Weisser, J., Abeck, S.** Development of SOA-based software systems — an evolutionary programming approach // Advanced Intern. Conf. on Telecommunications / Internet and Web Applications and Services. Guadeloupe, French Caribbean, February 2006. P. 182–195. Doi:10.1109/AICT-ICIW.2006.84.

[2] Simple object access protocol (SOAP) 1.1. Available at: https://www.w3.org/TR/soap/ (accessed 15.6.2016).

[3] XML remote procedure call protocol (standard). Available at: http://xmlrpc.scripting.com (accessed 18.6.2016).

[4] Learn REST: Representation state transfer. Available at: http://rest.elkstein.org (accessed 17.6.2016).

[5] Web processing service. Open geospatial consortium. Available at: http://www.opengeospatial.org/standards/wps (accessed 22.4.2016).

[6] **Бычков И.В., Ружников М., Фёдоров Р.К., Шумилов А.С.** Компоненты среды WPS-сервисов обработки геоданных // Вест. НГУ. Информационные технологии. 2014. Т. 12, № 3. С. 16–24.
Bychkov, I.V., Rugnikov, G.M., Fedorov, R.K., Shumilov, A.S. Components of WPS environment for geoprocessing // NSU J. of Information Technologies. 2014. Vol. 12, No. 3. P. 16–24. (In Russ.)

[7] Learn about Service-Oriented-Architecture. Available at: http://serviceorientation.com/ (accessed 29.6.2016).

[8] **Sachdeva, S., Rana, P.P.** A review of multiprocessor Directed Acyclic Graph (DAG) scheduling algorithms // Intern. J. of Computer Sci. & Communication. 2015. Vol. 66, No. 11. P. 67–72.

[9] **Sinnen, O.** Task scheduling for parallel systems. John Wiley & Sons, 2007. P. 108.

[10] **Shirazi, B., Wang, M.** Analysis and evaluation of heuristic methods for static task scheduling // J. of Parallel and Distributed Computing. 1990. Vol. 10, iss. 3. P. 222–232.

[11] **Zhao, H., Sakellariou, R.** An experimental investigation into the rank function of the heterogeneous earliest finish time scheduling algorithm // Lecture Notes in Computer Science. 2003. Vol. 2790. P. 189–194.

[12] **Nasonov, D., Butakov, N., Balakhontseva, M., Knyazkov, K., Boukhanovsky, A.V.** Hybrid evolutionary workflow scheduling algorithm for dynamic heterogeneous distributed computational environment // Advances in Intelligent Systems and Computing. 2014. Vol. 299. P. 83–92.

[13] OASIS Web Services Business Process Execution Language (WSBPEL) TC. Available at: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel (accessed 25.6.2016).

[14] XML Process Definition Language (XPDL). Available at: http://www.xpdl.org/ (accessed 24.6.2016).

[15] Oracle BPEL Process Manager. Available at: http://www.oracle.com/technetwork/middleware/bpel/overview/index.html (accessed 24.6.2016).

[16] Taverna Workflow System. Available at: https://taverna.incubator.apache.org/ (accessed 26.6.2016).

[17] **Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Greenwood, M., Carver, T., Glover, K., Pocock, M.R.** Taverna: a tool for the composition and enactment of bioinformatics workflows // Bioinformatics. 2004. Vol. 20. P. 82–98.

[18] **Бычков И.В., Ружников Г.М., Потапов В.П., Шумилов А.С., Фёдоров Р.К.** Асинхронное выполнение WPS сервисов на гетерогенных вычислительных ресурсах облачной информационно-вычислительной среды // Тр. Всерос. конф. “Обработка пространственных данных в задачах мониторинга природных и антропогенных процессов”. Новосибирск: ИВТ СО РАН, 2015. С. 60–65.
Bychkov, I.V., Rugnikov, G.M., Potapov, V.P., Shumilov, A.S., Fedorov, R.K. Asynchronous execution of of WPS services in heterogeneous computational resources of the cloud infrastructure // Proc. of the Nat. Conf. ”Geospatial Data Processing in the Field of Natural and Anthropogenic Processes Monitoring”. Novosibirsk: ICT SB RAS, 2015. P. 60–65. (In Russ.)

Поступила в редакцию 15 августа 2016 г.

System for dynamic execution of composition services in the heterogeneous environment

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

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

[*]Corresponding author: Shumilov, Alexander S., e-mail:alexshumilov@yahoo.com

The use of service-oriented architecture is considered, where all services are available through the Internet. Service composition, i. e. combination of services for the solution of complex tasks is a natural development of such infrastructure. The number and characteristics of various computational nodes can vary during the execution of service composition because of the environment heterogeneity.

In order to execute compositions of distributed services the system for dynamic planning and execution of service compositions was developed and implemented. System adapts to the changing conditions of the distributed environment — computing nodes can fail or become available, services can have varying execution time, network can have different bandwidth.

The proposed system has two main advantages — service compositions are described as scripts in JavaScript programming language and original algorithm of distributed services execution planning.

Using Java Script allows users to easily call services and treat results of these services execution as a regular data, using all capabilities that Java Script offers. At the same time users do not deal with the actual service execution planning as it is hidden behind the Java Script part.

Developed system takes care of distributed services compositions planning using the original approach. The goal of planning is to perform composition at the least possible time, taking into account the instability of the computing environment and statistical data that was collected before.

The system was tested and integrated into the Portal of ISDCT SB RAS. Tests were performed on the real interdisciplinary tasks.

Keywords: service-oriented architecture, service composition, Java Script, service orchestration.

Acknowledgements. This research was supported by RFBR (grants No. 16-07-00411, 16-37-00110, 14-07-00166) and by the Russian President's Council grant for State support of leading scientific schools (Nsh-8081.2016.9), the software implemented by the Center for collective use of resources, integrated information and computer network ISC FASO (Telecommunications Center of collective use "Integrated information and computer network of the Irkutsk Research and Educational complex") (http://net.icc.ru).

Received 15 August 2016