Список литературы

1. Стрыгин В. В., Соболев В. А. Разделение движений методом интегральных многообразий. – Москва: Наука. 1988. - 256 с.
2. Иманалиев З.К., Аширбаев Б.Ы. Разделение движений сингулярно возмущенной управляемой системы //Исследования по интегро-дифф. уравнениям. – Бишкек: Илим, 2007.- Вып.36. - С.136 -141.
3. Икрамов Х.Д. Численное решение матричных уравнений. Москва: Наука, 1984.- 192 с.
4. Беллман Р. Введение в теорию матриц. – Москва: Наука, 1976. – 352 с.
5. Иманалиев.З.К., Аширбаев Б.Ы., Алымбаева Ж.А. Решение дискретной задачи оптимального управления с малым шагом //Междунар. научно-информац. журн. «Наука и инновации». Вып. №1, Бишкек, 2013. С. 35 – 41.

УДК 004.75

ДИНАМИЧЕСКОЕ ВЫПОЛНЕНИЕ КОМПОЗИЦИЙ СЕРВИСОВ В ГЕТЕРОГЕННОЙ СРЕДЕ[1]

***Бычков Игорь Вячеславович**, д.т.н., академик, ИДСТУ СО РАН, Россия, 664033, г.Иркутск, ул. Лермонтова 134,* e-mail: idstu@icc.ru
***Ружников Геннадий Михайлович**, д.т.н., с.н.с., ИДСТУ СО РАН, Россия,*
***Фёдоров Роман Константинович**, к.т.н., доц., ИДСТУ СО РАН, Россия,*
***Шумилов Александр Сергеевич**, ИДСТУ СО РАН, Россия*

Цель статьи – использование сервис-ориентированной архитектуры для вычисления композиции распределённых сервисов, доступных через Интернет. Количество вычислительных узлов и их характеристики может меняться в процессе выполнения композиции в силу гетерогенности среды. Разработана и реализована система динамического планирования и выполнения композиций Web-сервисов, учитывающая требования изменяющейся гетерогенной среды, которая интегрирована в существующую инфраструктуру Портала ИДСТУ СО РАН

Ключевые слова: сервис-ориентированная архитектура, Web-сервисы, композиция распределённых сервисов, гетерогенная вычислительная среда, портал, оркестрация сервисов.

DYNAMIC EXECUTION COMPOSITION OF SERVICES IN HETEROGENEOUS ENVIRONMENTS

***Bychkov Igor V.,** Doctor of Technical Sciences (Engineering), Academician, ISDCT of SB RAS, Russia, 664033, Irkutsk, Lermontov 134, e-mail: idstu@icc.ru*
***Ruzhnikov Gennadii M.,** Doctor of Technical Sciences (Engineering), Senior Researcher., ISDCT of SB RAS, Russia,*
***Fedorov Roman K.,** PhD (Engineering), Associate Professor, ISDCT of SB RAS, Russia,*
***Shumilov Alexander S.,** graduate student, ISDCT of SB RAS, Russia,*

The purpose of the article - the use of service-oriented architecture to calculate the composition of distributed services that are available over the Internet. The number of computing nodes and their characteristics may change during the execution of the composition due to the heterogeneity of the

[1] Работа выполнялась при финансовой поддержке грантов РФФИ 16-57-44034-а, 16-07-00411-а, 14-07-00166-а и проекта Интеграционной программы ИНЦ СО РАН

environment. It developed and implemented a dynamic system of planning and performing songs Web-services, taking into account the varying requirements of a heterogeneous environment that is integrated into the existing infrastructure of the Portal ISDCT SB RAS.

Keywords: service-oriented architecture, Web-services, the composition of distributed services, heterogeneous computing environment, the portal, JavaScript, orchestration services.

1. Введение

В последнее время активно развивается сервис-ориентированная архитектура вычислений (SOA – Service-Oriented Architecture), которая предполагает представление программных компонентов в виде отдельных вычислительных сервисов. Основными преимуществами данного подхода является простота использования сервисов (сложность реализации сервиса полностью скрываются за его интерфейсом), легкое тестирование и отладка, масштабируемость и возможность повторного использования. Сервисы имеют определенный интерфейс, доступ и описание которого определяется одним из стандартов: SOAP (Simple Object Access Protocol), RPC (Remote Procedure Call), REST (Representational State Transfer) [3].

При обработке пространственных данных, в силу их специфики, актуально использование уже готовых сервисов, в том числе удалённых, развернутых на разных узлах гетерогенной вычислительной среды. При использовании таких сервисов нет необходимости устанавливать новое программное обеспечение, вести и актуализировать базы адресов или сети передачи данных. Сервисы обработки пространственных данных, как правило, используют стандарт Web Processing Service (WPS), разработанный консорциумом Open Geospatial Consortium (OGC), который определяет механизм описания, выполнения и контроля сервисов, а также формат передачи пространственных данных. Длительность выполнения сервисов, базирующихся на WPS, не ограничена. Данный 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. Под композицией понимается набор сервисов с определёнными между ними зависимостями, решающих какую-либо задачу или автоматизирующих какой-либо процесс.

Существует несколько подходов задания композиции сервисов, которые сводятся к определению графа зависимости вызовов сервисов.

При выполнении композиций распределённых сервисов возникает ряд особенностей:

- каждый сервис композиции может выполняться на различных вычислительных узлах разной производительности и пропускной способности каналов связи;
- количество выделяемых вычислительных узлов может динамически меняться;
- сервисы могут иметь разную длительность работы, которая меняется в процессе их выполнения;
- прекращение работы вследствие сбоев, получение некорректных данных и т.д.

В зависимости от решаемой задачи, набор и последовательность выполняемых сервисов композиции может динамически изменяться. Поэтому, при выполнении композиций распределённых сервисов актуальна задача их планирования с учетом гетерогенной динамически изменяющейся среды, с целью минимизации времени их выполнения или используемых ресурсов. Для решения данной задачи разработана и реализована система выполнения композиций Web-сервисов, позволяющая динамически адаптироваться под изменяющееся состояние среды.

2. Обзор средств

В композиции должны присутствовать как минимум три взаимодействующих сервиса, иначе взаимодействие двух сервисов представляет собой лишь обмен данными [9].

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) [8], в котором вершинами являются сами задания, а ребра показывают зависимости между заданиями по данным (рис. 1). Задания, которые не зависят от других заданий, являются начальными (1 и 2 на рис. 1), с них начинается выполнение композиции. Задание, от которого не зависят другие задания, является выходным (4 на рис. 1). Все остальные задания являются промежуточными (3 на рис. 1). В DAG есть как минимум один входной и выходной (терминальный) узел.

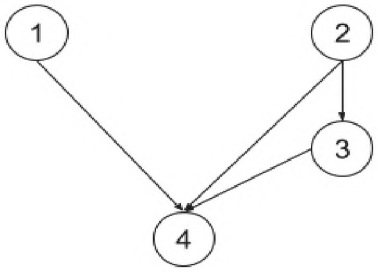

Рис. 1. Граф зависимостей заданий по данным

Проблема составления расписания на основе DAG исследуется с 70-х годов прошлого века. Отыскивание такого расписания для набора заданий, которое бы минимизировало время выполнения всего DAG – является NP-полной задачей за исключением нескольких простейших случаев [10]. Существующие методы находят только приближенные решения. Все алгоритмы нахождения расписания для DAG можно разделить на две группы по способу нахождения расписания – эвристические и метаэвристические алгоритмы.

Одним из наиболее популярных эвристических алгоритмов составления расписания является алгоритм Heterogeneous Earliest Finish Time (HEFT) [4], который является списковым алгоритмом. Среди метаэвристических алгоритмов наибольшей популярностью пользуется семейство генетических алгоритмов [4].

Стоит 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, в которой работают данные алгоритмы, изменились, так как большинство сред стали гетерогенными.

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 [4], сочетающий 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 также изменяется со временем.

Говоря о подходах к заданию композиций сервисов, необходимо рассмотреть наиболее известные два способа их задания – BPEL (Business Process Execution Language) [5] и XPDL (XML Process Definition Language). Эти способы дескриптивно задают граф зависимостей заданий DAG. Чаще всего в работе используется стандарт BPEL, так как он более полно описывает взаимодействие сервисов. В свою очередь, XPDL позволяет использовать сервисы, не имеющие веб-интерфейс, позволяет определять графическое представление заданного взаимодействия.

Одним из наиболее популярных программных пакетов, позволяющих задавать композиции сервисов с помощью формата BPEL, является Oracle BPEL Process Manager, который предоставляет удобный графический интерфейс для создания композиций сервисов и средство отладки созданных 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.

Также известен для задания композиций сервисов программный продукт Taverna, 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 [6]. Taverna так же не осуществляет планирование выполнения композиций в сложных средах.

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 приближённого к оптимальному времени выполнения. Однако отсутствуют алгоритмы, позволяющие динамически в процессе выполнения определять граф зависимостей заданий DAG, осуществлять планирование его выполнения с учетом особенностей гетерогенных сред распределенных вычислений.

2. Описание гетерогенной среды

Структура гетерогенной среды, в рамках которой производится выполнение композиции сервисов, приведена на рис. 2.

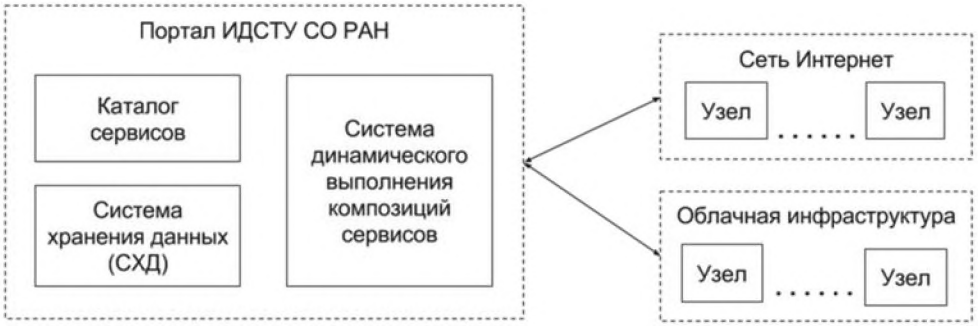

Рис. 2. Структура среды

Каталог сервисов позволяет регистрировать и запускать сервисы на Портале и хранит информацию о них (данные о вычислительных узлах, поддерживающих зарегистрированный сервис, описания входных и результирующих параметров сервиса). Каталог хранит информацию о сервисах, реализованных как на удалённых вычислительных узлах, так и на узлах, развернутых в пределах локальной облачной инфраструктуры – наборе виртуальных машин, реализующих различные геоинформационные веб-сервисы. Система хранения данных (СХД) позволяет пользователям Портала загружать, хранить и использовать в качестве входных параметров сервисов файлы, а также обеспечивает обмен файлами между запускаемыми сервисами [1].

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

1. Граф зависимостей заданий DAG композиции сервисов может модифицироваться в процессе выполнения;
2. Планирование должно обеспечивать приемлемое время выполнения композиций;
3. Во время выполнения композиции система должна обрабатывать происходящие изменения в гетерогенной среде, будь то отказ или восстановление вычислительных узлов,

изменения ожидаемого времени выполнения заданий, неполадки в сетевой инфраструктуре, и т.д., то есть 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 композиций сервисов с помощью языка программирования JavaScript. Композиция сервисов представляет собой программный код, в котором вызов сервисов осуществляется с помощью специальных функций, закреплённых 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, Порталу известно только их сетевое расположение, описание параметров и характеристики самих вычислительных узлов. Пример композиции представлен ниже (композиция вычисляет массу выделяемых загрязняющих веществ на регулярной сетке местности с помощью WPS-сервисов `vectorToGrid`, `roadToGrid` и `raster_sum`:

```
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});
}
```

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-обёртки (в случае приведённого выше сценария это функции `vectorToGrid`, `roadToGrid` и `raster_sum`), каждая из которых соответствует определённому сервису. Во время вызова функций-обёрток, создаются вершины графа, т.е. регистрируются задания. Для определения зависимостей между заданиями используются объекты класса `ValueStore`, которые передаются в качестве параметров функций сервисов. UML-диаграмма класса `ValueStore` приведена на рис. 3. Объекты класса `ValueStore` необходимы для однозначной идентификации передаваемых данных при вызове функций сервисов. Вызовы специальных функций не блокируют выполнение JavaScript кода. Блокировка сценария производится только в случае вызова метода `Get` объекта `ValueStore`, если данные еще не вычислены, с помощью которого можно получить значение получаемых данных. Представленный выше пример сценария начинает выполнение и заканчивает работу без ожидания результатов работы сервисов.

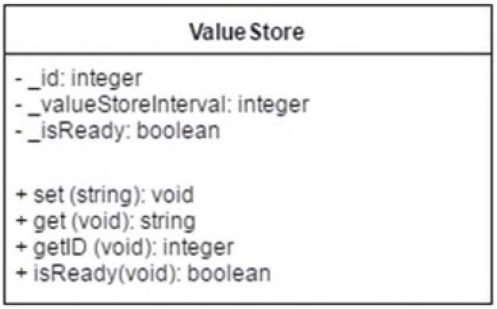

Рис. 3. Класс `ValueStore`

Применение процедурного языка программирования для определения DAG имеет ряд преимуществ:

- это привычный язык разработки алгоритмов, так как применение сервисов не отличается от использования 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, то есть нахождение оптимального решения в некоторых случаях для такого рода задач является практически невыполнимым. Таким образом, осуществляется поиск приемлемого решения, то есть решения, максимально приближенного к оптимальному решению.

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

– множество вычислительных узлов $N = \{n_i\}_{i=1..K}$, где $n_1..n_k$ – вычислительные узлы, k – количество вычислительных узлов;

– множество заданий $T = \{t_i\}_{i=1..M}$, где $t_1..t_m$ – задания, m – количество заданий. $c_e(t_i)$ – 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 (то есть 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, вершинами графа являются задания, ребра являются отношениями зависимости между заданиями (например, t_i не может начать своё выполнение пока не закончилось задание t_j);

– отношение $busy(n_i)$, показывающее время высвобождения узла с учётом 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, то есть не существует таких заданий t_i, что выполняется $depends(t_{start}, t_i)$;

– существует некоторое подмножество T^{exit} множества T, содержащее выходные задания графа, то есть не существует таких задач 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 $(t, n)_m$, где t это задание, n это вычислительный узел, на который задание назначается, m 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 не перспективных ветвей графа поиска решения. Рассмотрим каждую технику в отдельности на примере графа на рис. 4, где заданы идентификаторы задач, время выполнения заданий и стоимость передачи данных для каждого задания.

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:

если $t_i \in T^{exit}$ ***то*** $aggregatedCost(t_i) = w_e(t_i)$

иначе $aggregatedCost(t_i) = w_e(t_i) + \max_{t_j \in D}(aggregatedCost(t_j))$,

где w – среднее время выполнение задания на всех поддерживающих его вычислительных узлах, D – множество заданий, зависимых от t_i.

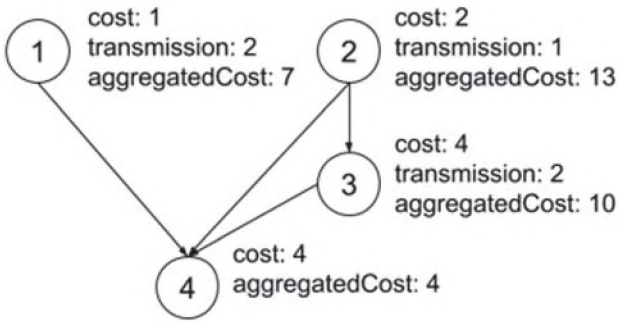

Рис. 4. Граф зависимости заданий

1. Сортировка заданий по убыванию значения функции $aggregatedCost(t)$ происходит перед началом работы алгоритма, таким образом, алгоритм будет поочередно выбирать задания для анализа, начиная с самого затратного по времени задания.

2. Внутри самого алгоритма после каждого назначения задания на определенный узел происходит сортировка узлов по возрастанию значения их занятости. На рис. 5 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. 5. Сортировка вычислительных узлов

3. В то время как первые две техники позволяют находить приближенное расписание как можно быстрее, техника пропуска ветвей графа позволяет сократить

пространство перебора решений и 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 пропуска ветви представлен на рис. 6.

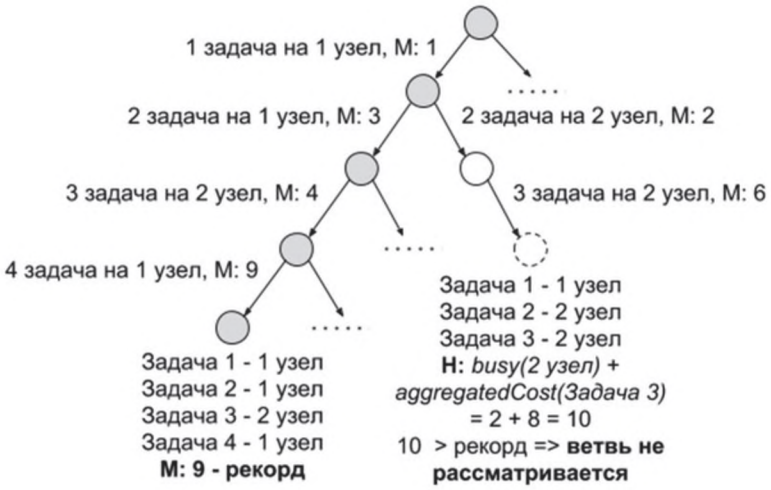

Рис. 6. Отсечение ветвей при выполнении алгоритма

4. 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. 7), каждый из которых является назначением какого-то сервиса t_i на узел n_j.

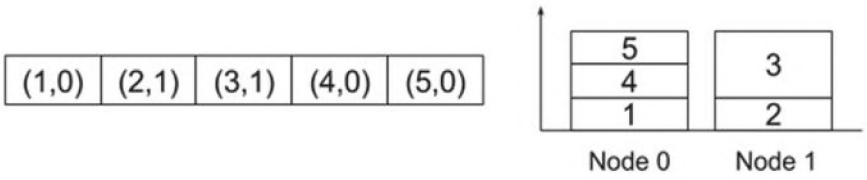

Рис. 7. Представление расписания в виде последовательности

6. Алгоритмы динамического изменения расписания

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

1. Если происходит добавление или удаление вычислительного узла – то есть 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 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). 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. 8 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);

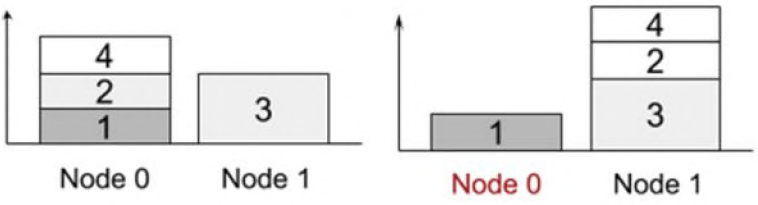

Рис. 8. Отказ узла во время выполнения композиции

2. Если происходит добавление задания в DAG – то есть 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. 9 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. 9 после добавления задания 5 время выполнения расписания увеличивается, следовательно, необходимо выполнить перерасчёт;

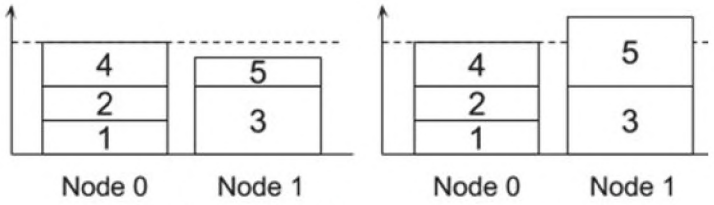

Рис. 9. Добавление задания в план

3. Если завершённое задание выполнялось меньше ожидаемого времени – то есть 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. 10 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);

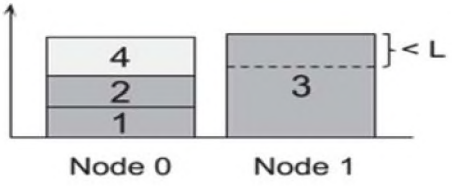

Рис. 10. Добавление задания в план

4. Если выполняющееся задание не закончилось в ожидаемый срок – то есть 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 рис. 11 отображена ситуация, когда задание 3 выполняется дольше, чем планировалось. Левая схема показывает

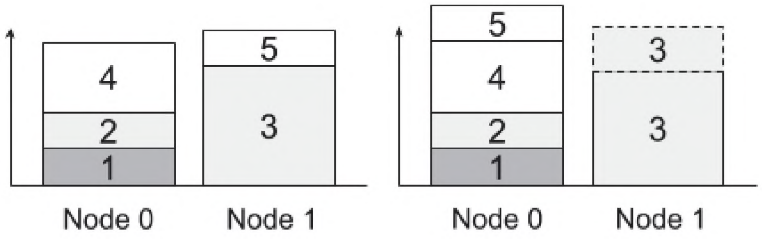

Рис. 11. Задание выполняется дольше чем ожидалось

изначальное расписание, правая схема показывает перестроенное расписание с добавленным для задания 3 ожидаемым временем выполнения;

5. Если выполняющееся задание завершилось с ошибкой – то есть 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. 12 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;

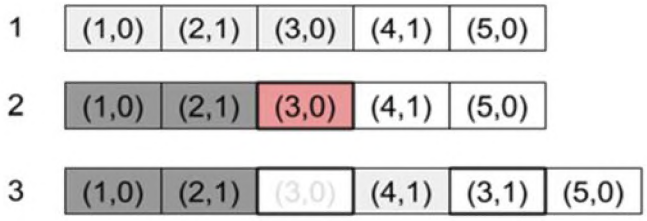

Рис. 12. Завершение задания с ошибкой

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 сервисов имеет фактическое значение, найденное в процессе выполнения композиции сервисов, таким образом, перестраиваемое расписание более точно.

7. Система динамического выполнения композиций сервисов

Система динамического выполнения композиций сервисов реализована в виде отдельного приложения (рис. 13), встраиваемого в Портал. Портал реализует функции хранения информации об участвующих в композиции сервисах – их типах, сетевом расположении, характеристиках вычислительных узлов, на которых они могут располагаться. Сервисы регистрируются пользователями на Портале, который предоставляет удобный интерфейс для ввода параметров запускаемых композиций, среди которых 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 диспетчеру.

Рис. 13. Система динамического выполнения композиций сервисов

Принятый на вход диспетчером программный код компилируется с помощью встроенного JavaScript движка Google V8 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 заданной периодичностью опрашивает глобальную очередь на предмет изменения состава или статуса зарегистрированных в ней заданий. При наступлении определенных событий, рассмотренных в п.4, происходит перестроение расписания с учетом уже выполняющихся заданий. Для этого необходимо знать время выполнения заданий 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 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, а также отслеживает вызовы сервисов по двум параметрам: сетевой адрес сервиса и идентификатор сервиса.

Перед тем, как составить расписание с вызовом определенного сервиса, нужно получить его ожидаемое время выполнения. Для чего 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 тогда делается предположение относительно его длительности на основании заранее заданных значений, зависящих от типа сервиса.

8. Заключение.

Спроектирована и разработана, а также интегрирована в существующую инфраструктуру Портала ИДСТУ СО РАН система выполнения композиций сервисов в гетерогенной среде. Основные преимущества системы: способность к перестройке расписания вызова сервисов при изменениях состояния гетерогенной среды; планирование с учетом динамического изменения графа зависимостей заданий. Задание композиций сервисов с помощью JavaScript позволяет поддерживать планирование и синхронизировать выполнение сервисов и передачу данных между ними, что упрощает процесс разработки композиций сервисов.

Список литературы

1. Компоненты среды WPS-сервисов обработки геоданных/ И.В Бычков и др. // Вестник НГУ. Серия: Информационные технологии. – 2014. - № 12. – С. 16-23.
2. Emig C., Weisser J., Abeck S.. Development of SOA-Based Software Systems - an Evolutionary Programming Approach // Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services (AICT-ICIW'06). – 2006. С. 14-15.
3. Learn REST: Representation State Transfer [Электронный ресурс] // URL:http://rest.elkstein.org(дата обращения: 17.6.2016).
4. Nasonov D., , Butakov N., Balakhontseva M., Knyazkov K. , Boukhanovsky A. Hybrid Evolutionary Workflow Scheduling Algorithm for Dynamic Heterogeneous Distributed Computational Environment // Advances in Intelligent Systems and Computing. – 2014. Vol. 299. С. 83-92.
5. OASIS Web Services Business Process Execution Language (WSBPEL) TC [Ýëåêòðîííûé ðåñóðñ] // URL: https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel (äàòà îáðàùåíèÿ: 25.6.2016).
6. Oinn T., Addis M., Ferris J., Marvin D., Senger M., Greenwood M., Carver T., Glover K., Pocock M.. Taverna: a tool for the composition and enactment of bioinformatics workflows // Bioinformatics. – 2004. Vol. 20. С. 82-98.
7. Oracle BPEL Process Manager [Ýëåêòðîííûé ðåñóðñ] // URL: http://www.oracle.com/technetwork/middleware/bpel/overview/index.html (äàòà îáðàùåíèÿ: 24.6.2016).
8. Sachdeva S., Rana P.. A Review of Multiprocessor Directed Acyclic Graph (DAG) Scheduling Algorithms // International Journal of Computer Science & Communication. – 2015. – Vol. 66, N. 11. С. 67-72.
9. Service Orientation | Learn Service-Priented-Architecture [Электронный ресурс] // URL: http://serviceorientation.com/ (дата обращения: 29.6.2016).
10. Shirazi B., Wang M. Analysis and evaluation of heuristic methods for static task scheduling // Journal of Parallel and Distributed Computing. – 2010. - Vo. 10, Is. 3. С. 222-232.

УДК 51:517.9

О ЗАДАЧЕ ДАРБУ-ГУРСА-БОДО В ПРОСТРАНСТВЕ НЕПРЕРЫВНЫХ ФУНКЦИЙ

Бердимуратов Амангельди Мухтарович, к.ф.м.н, *КНУ им. Ж.Баласагына, Кыргызстан,*
e-mail: *aman2460@mail.ru*