ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ № 3(17), 2014, с.??—??

УДК 004.75

О. В. Сухорослов

Интеграция вычислительных приложений и распределенных ресурсов на базе облачной программной платформы[1]

Аннотация. В статье рассматривается облачная программная платформа Everest, реализующая публикацию, выполнение и композицию вычислительных приложений в распределенной среде. Описаны архитектура платформы, принципы создания приложений, интеграция платформы с внешними вычислительными ресурсами, а также интерфейс прикладного программирования платформы.

Ключевые слова и фразы: распределенные вычисления, облачная платформа, вебсервисы, REST, интеграция вычислительных ресурсов, композиция приложений

Введение

Возможность беспрепятственно использовать и сочетать при решении задач существующие вычислительные приложения и ресурсы является ключевым фактором, влияющим на производительность исследований во многих областях науки. Однако научные приложения зачастую требуют особой квалификации для своей установки, конфигурации и запуска, которой не обладает большинство исследователей. То же относится к конфигурации и использованию вычислительных ресурсов, требуемых для запуска приложений. Кроме того, исследователи часто ощущают потребность в автоматизации совместного использования нескольких приложений и в эффективном запуске приложений на нескольких доступных ресурсах.

[1] Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 14-07-00309 а.

Существующие подходы и технологии решают указанные проблемы, однако - не в полной степени. Грид-технологии используют сервис-ориентированную архитектуру для реализации стандартизованного доступа к распределенным ресурсам через веб-сервисы. Данный подход значительно упрощает задачи интеграции ресурсов и запуска приложений, однако не решает остальные проблемы и является низкоуровневым для большинства исследователей. Вычислительные порталы (грид-порталы, шлюзы, хабы) реализуют удаленный доступ к приложениям и ресурсам через пользовательские веб-интерфейсы. Данный подход значительно упрощает работу с приложениями и ресурсами для простых исследователей, однако он не решает проблему композиции приложений, поскольку отсутствует программный доступ к приложениям. Существует также ряд программных инструментариев, реализующих преобразование научных приложений в веб-сервисы. Данный подход обеспечивает программный доступ к приложениям, однако существующие реализации имеют ряд недостатков, таких как отсутствие общих соглашений по интерфейсам сервисов и необходимость размещения сервисов на собственном сервере.

Платформа *Everest* [1][2] реализует новый подход к решению проблем публикации, выполнения и композиции вычислительных приложений в распределенной среде. Данный подход объединяет преимущества существующих подходов и в то же время устраняет описанные недостатки. Делается это главным образом путем использования моделей облачных вычислений и применения современных программных технологий.

В отличие от существующих решений, Everest является облачной платформой, реализующей модель облачных вычислений *Platform as a Service (PaaS)*. Это означает, что вся функциональность платформы доступна через удаленные интерфейсы: пользовательский веб-интерфейс и программный интерфейс (REST API). Один экземпляр платформы может обслуживать много пользователей, позволяя им размещать свои приложения, запускать их и делиться ими с другими пользователями без необходимости уста-

новки дополнительного ПО на компьютеры пользователей. Размещенные в Everest приложения автоматически становятся доступными через веб-интерфейс и REST API. Последний позволяет автоматизировать работу с приложениями, осуществлять их композицию в рамках расчетных схем (workflow) и работать с приложениями из внешних систем. Другой отличительной чертой Everest является поддержка запуска приложений на произвольных комбинациях внешних вычислительных ресурсов, подключенных пользователями платформы.

В статье рассматриваются архитектура платформы Everest (глава 1), принципы реализации приложений (глава 2), интеграция платформы с внешними вычислительными ресурсами (глава 3), а также интерфейс прикладного программирования платформы (глава 4). В заключении обсуждаются планы дальнейших исследований.

1. Архитектура Everest

На Рис. 1 изображена архитектура Everest. Серверная часть платформы состоит из трех уровней: программный интерфейс (REST API), приложения и вычислительное ядро. Клиентская часть платформы включает пользовательский веб-интерфейс (Web UI) и клиентские библиотеки.

REST API — программный интерфейс платформы, реализованный в виде веб-сервиса на основе стиля REST [3]. Данный интерфейс включает операции для работы со всеми основными объектами платформы, такими как приложения, задания, ресурсы. Это единая точка входа для всех клиентов платформы, включая веб-интерфейс и клиентские библиотеки.

Уровень приложений соответствует среде размещения приложений, созданных пользователями платформы. Приложения является главными объектами платформы, представляющими собой вычислительные блоки с четко описанными интерфейсами. Каждое размещенное в Everest приложение автоматически публикуется как веб-сервис через REST API. Владелец приложения может настраивать список пользователей, которым доступен запуск приложения.

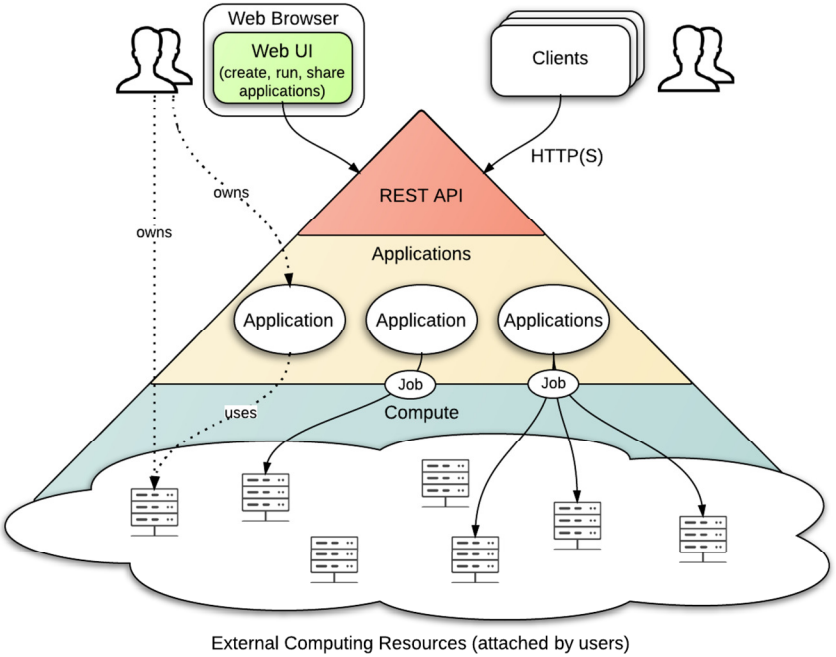

Рис. 1. Архитектура платформы Everest.

Платформа Everest не предоставляет собственных ресурсов для выполнения приложений и не связана жестко с какой-либо одной инфраструктурой. Вместо этого она позволяет пользователям подключать к себе внешние ресурсы и запускать приложения на любых сочетаниях данных ресурсов.

Вычислительное ядро управляет выполнением приложений на удаленных ресурсах. При вызове приложения через интерфейс платформы формируется *задание (job)*, состоящее из одной или нескольких *вычислительных задач (tasks)*. Ядро реализует все действия связанные с передачей данных, запуском и мониторингом задач на удаленных ресурсах. Также ядро осуществляет мониторинг состояния самих ресурсов и использует данную информацию при планировании задач.

Веб-интерфейс (Web UI) представляет собой графический интерфейс для работы пользователей с платформой. Данный интерфейс реализован в виде приложения на языке JavaScript, которое может выполняться в веб-браузере без необходимости установки дополнительного ПО. Веб-интерфейс взаимодействует с платформой через REST API, то есть использует тот же интерфейс, что и остальные клиенты платформы.

Клиентские библиотеки предназначены для упрощения программного доступа к платформе через REST API. Они позволяют пользователями легко писать программы, которые вызывают приложения и комбинируют их в произвольные сценарии. В настоящий момент реализована одна такая библиотека для языка Python.

2. Приложения

Приложения являются основой Everest — любые вычисления производятся в контексте некоторого приложения. Клиенты взаимодействуют с приложениями путем отправки запросов и получения обратно результатов расчетов.

В части их внешнего интерфейса приложения Everest следуют единой модели. А именно — приложение имеет набор *входов* (определяют запрос к приложению) и *выходов* (определяют результат расчетов для некоторого запроса). Данные приложения удобно представлять в виде “черных ящиков” с некоторыми входными и выходными портами, или в виде “функции” с некоторыми аргументами и возвращаемыми значениями.

Описанная модель позволяет реализовать унифицированный REST-интерфейс для доступа к приложениям по сети [4], что необходимо для поддержки композиции приложений. Данный интерфейс реализован как часть REST API платформы.

То, каким образом происходит обработка запросов к приложению, скрыто от пользователя. Что же происходит внутри платформы, и как реализуются приложения? Обработка каждого запроса к приложению включает следующие действия:

(1) аутентификация и авторизация клиента (кто он и имеет ли доступ к приложению);
(2) разбор и проверка входных значений (указаны ли все обязательные параметры, имеют ли они допустимые значения);
(3) создание нового задания, которое будет использоваться для отслеживания состояния запроса и получения результатов,
(4) трансляция входных значений в одну или несколько вычислительных задач, которые необходимо запустить на ресурсах в рамках данного задания;
(5) выполнение сформированных задач на заданных ресурсах;
(6) трансляция результатов задач в значения выходных параметров, возвращаемых клиенты.

Здесь шаги 1, 3 и 5 могут быть реализованы одинаковым образом для всех приложений. В то же время шаги 2, 4 и 6 зависят от конкретного приложения. Для упрощения создания приложений Everest включает универсальные реализации данных шагов (т.н. *каркасы*), которые могут быть настроены пользователем под цели конкретного приложения.

Реализованный "декларативный" подход позволяет во многих случаях избежать программирования при размещении в Everest приложений. Пользователю необходимо сформировать через веб-интерфейс *описание приложения*, которое состоит из двух основных частей:

- *публичная информация*, используемая клиентами для поиска и вызова приложения (название, аннотация, описание входов и выходов и т.п.);
- *внутренняя конфигурация*, используемая платформой для обработки запросов к приложению (настройки каркаса, файлы, вычислительные ресурсы и т.п.).

Описания входных параметров приложения используются платформой для реализации указанного выше шага 2. Что касается шагов 4 и 6, то для них в настоящий момент реализован один каркас — для приложений с интерфейсом командной строки. Данный каркас может быть использован для переноса в Everest существующих приложений. Каркас генерирует задания, состоящие из

одной задачи, в которой непосредственно происходит запуск приложения. Конфигурация каркаса включает в себя:

- шаблон команды задачи, поддерживающий вставку внутрь значений входных параметров;
- отображения входных параметров приложения во входные файлы задачи;
- отображения выходных файлов задачи в выходные параметры приложения.

В дальнейшем планируется создание дополнительных каркасов для приложений, порождающих несколько задач, в том числе композитных приложений. Поддержка заданий с несколькими задачами уже реализована в вычислительном ядре платформы, что позволило создать универсальный сервис для запуска многовариантных расчетов [5].

3. Интеграция с вычислительными ресурсами

Как уже было упомянуто, вместо собственной вычислительной инфраструктуры Everest использует для выполнения приложений внешние ресурсы, подключаемые пользователями. При этом вычислительное ядро платформы выполняет функции диспетчеризации задач на заданных пользователями наборах ресурсов.

В настоящее время основной метод подключения ресурсов к платформе основан на использовании специально разработанной программы, т.н. *агента*. Агент выполняется на стороне ресурса и играет роль посредника между платформой и ресурсом. У данного метода есть один недостаток – он требует развертывания агента на каждом подключаемом ресурсе. В сравнении с доступом к ресурсам по существующим протоколам (например, SSH), данный метод обладает следующими преимуществами:

- не требуется доступ к ресурсу извне по сети (ресурс может находиться за межсетевым экраном или во внутренней сети организации),
- возможность реализации на уровне агента контроля действий, осуществляемых платформой на ресурсе.

Агент реализован на языке Python и может быть быстро развернут на ресурсе пользователем без специальной подготовки и прав суперпользователя. Взаимодействие между агентом и платформой реализовано на базе протокола WebSocket [6], который позволяет создать двусторонний канал связи между клиентом (агент) и сервером (платформа). Данный канал используется для передачи агенту команд и приема от него различных уведомлений. Передача данных вычислительных задач между агентом и платформой осуществляется отдельно по протоколу HTTP. При подключении к платформе агент проходит процедуру аутентификации путем передачи секретного кода, выданного платформой при регистрации ресурса.

Интеграция агента с различными типами ресурсов реализована через механизм адаптеров. В настоящий момент реализованы и используются следующие адаптеры:

- *local* – запуск задач на локальной машине,
- *docker* - запуск задач на локальной машине внутри одноразовых Linux-контейнеров,
- *torque* – запуск задач на кластере под управлением TORQUE,
- *slurm* – запуск задач на кластере под управлением SLURM.

В двух последних случаях агент выполняется на запускающей машине кластера.

Для того чтобы подключить ресурс к платформе, пользователь должен зарегистрировать его через веб-интерфейс и получить секретный код ресурса. После этого необходимо развернуть агента на ресурсе, указав в конфигурации агента полученный секретный код. После успешного подключения агента к платформе он начинает передавать в Everest информацию о состоянии ресурса, которая отображается в веб-интерфейсе.

Помимо доступа к отдельным серверам и кластерам через описанного агента, была реализована интеграция платформы с европейской грид-инфраструктурой EGI. Данная интеграция выполнена на базе пользовательского интерфейса EGI (EMI User Interface). Пользователь Everest может подключить в качестве ресурса опре-

деленную виртуальную организацию EGI, передав платформе действующий прокси-сертификат.

Для того чтобы платформа могла запустить приложение, оно должно быть *связано* хотя бы с одним доступным ресурсом. В Everest реализован гибкий подход, позволяющий

- связывать ресурс с приложением *статически* (владелец приложения указывает ресурс в конфигурации приложения),
- связывать ресурс с приложением *динамически* (пользователь приложения сам выбирает ресурс перед запуском приложения),
- связывать не один, а несколько ресурсов,
- сочетать два описанных варианта (у приложения есть “свои” ресурсы, но пользователь может запустить его на “ ресурсах).

Данный подход открывает новые возможности, но и привносит с собой проблемы. Например, владелец коммерческого или требующего специальных ресурсов приложения может ограничить используемые ресурсы (статическое связывание), но при этом ему необходимо самому поддерживать данные ресурсы. С другой стороны, если владелец приложения хочет сделать свое приложение доступным широкой аудитории, не “вкладываясь” в ресурсы, он может использовать динамическое связывание (ресурс предоставляет пользователь). Однако в данном случае приложение должно быть оформлено в виде, максимально переносимом между гетерогенными ресурсами. При этом пользователи приложения должны доверять коду приложения, запускаемого на их ресурсах. Кроме того, в случае связывания приложения с несколькими ресурсами, возникает проблема эффективного планирования выполнения задач.

В рамках работ над Everest развиваются подходы к решению указанных проблем.

4. Программный доступ

Работа с приложениями через веб-интерфейс является удобным и интуитивным способом, однако у него есть ряд ограничений. Например, если требуется запустить приложение много раз с различными значениями параметров, то делать это вручную через веб-

форму неудобно. В случае если требуется получить некоторый результат путем использования нескольких приложений, то пользователю необходимо вручную копировать данные между несколькими заданиями. Наконец, веб-интерфейс не позволяет обращаться к приложениям из внешних программ.

Для поддержки решения описанных выше проблем реализован программный доступ к приложениям через REST API платформы. Поскольку данный интерфейс оформлен в виде веб-сервиса, то работать с ним можно на любом языке программирования, для которого есть реализация HTTP-клиента и формата JSON. Однако работа напрямую с веб-сервисом требует от пользователей особой квалификации, поэтому для упрощения работы с REST API была реализована клиентская библиотека *Python API*. Данная библиотека позволяет из программ на языке Python удобным образом обращаться к размещенным в Everest приложениям и комбинировать их в произвольные *расчетные схемы (workflow)*.

На Рис. 2 приведен пример программы, использующей Python API. Данная программа реализует простую расчетную схему (см. верхний правый угол рисунка), включающую обращения к четырем приложениям – *A*, *B*, *C* и *D*.

В начале программы происходит импортирование модуля *everest*, который реализует Python API, и создание новой *сессии* для работы с платформой. При создании сессии необходимо указать действующий *клиентский токен (client token)*, используемый REST API для аутентификации клиентов.

Для работы с приложениями программа создает новые объекты типа *App* для каждого из приложений, указывая уникальный идентификатор приложения. Отправка запросов к приложениям происходит путем вызова метода *run()* у соответствующих объектов. Значения входных параметров передаются в виде словаря, ключами и значениями которого являются соответственно имена и значения параметров. Метод *run()* возвращает объект типа *Job*, который может использоваться для проверки состояния задания и получения его результатов.

```
import everest

session = everest.Session(
  'https://everest.distcomp.org', token = '...'
)

appA = everest.App('52b1d2d13b...', session)
appB = everest.App('...', session)
appC = everest.App('...', session)
appD = everest.App('...', session)

jobA = appA.run({'a': '...'})
jobB = appB.run({'b': jobA.output('out1')})
jobC = appC.run({'c': jobA.output('out2')})
jobD = appD.run({'d1': jobB.output('out'), 'd2': jobC.output('out')})

print(jobD.result())

session.close()
```

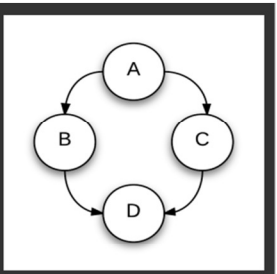

Рис. 2. Пример программы, использующей Python API.

Важно отметить, что метод *run()* не блокирует выполнение программы до окончания выполнения задания и получения его результатов. Вместо этого Python API позволяет программе продолжить свое выполнение сразу после создания запроса, выполняя все действия, связанные с его отправкой и отслеживанием задания, в фоновом потоке.

В представленном примере все задания кроме *jobA* не могут быть запущены сразу после вызова *run()*, поскольку они зависят от результатов других заданий. Программа ссылается на выходные значения (возможно еще невыполненного) задания, используя метод *output()* объекта-задания и указывая имя соответствующего выходного параметра (например, для *jobA* это – *out1* и *out2*). Данный метод также не блокирует выполнение программы до момента получения результата. Вместо этого Python API ожидает в фоновом потоке завершения задания, считывает его выходные значения и запускает зависимые задания по мере готовности их входных значений. Так в приведенном примере задания *jobB* и *jobC* будут автоматически запущены после выполнения задания *jobA*, а задание *jobD* - после выполнения заданий *jobB* и *jobC*.

Описанная неблокирующая семантика Python API, близкая по смыслу к парадигме *data flow*, позволяет

- свести программу к декларативному описанию заданий и зависимостей между ними по данным,
- избавить пользователя от необходимости самому реализовывать код, занимающийся ожиданием выполнения заданий и передачей данных между ними,
- естественным образом реализовать параллельное выполнение независимых заданий (в приведенном примере – заданий *jobB* и *jobC*).

После того, как все задания созданы (хотя, возможно, еще не запущены), программа ожидает финальный результат путем вызова метода *result()* у задания *jobD*. Данный метод блокирует выполнение программы до завершения задания и возвращает результат задания. По аналогии с запросом, результат передается в виде словаря, ключами и значениями которого являются соответственно имена и значения выходных параметров.

В заключении программа завершает сессию путем вызова метода *close()*, который останавливает фоновые потоки и обеспечивает нормальное завершение программы.

Заключение

В статье рассмотрена платформа Everest, реализующая публикацию, выполнение и композицию вычислительных приложений в распределенной среде. Особенностями данной платформы являются применение облачной модели PaaS, публикация приложений в виде REST-сервисов и поддержка гибкого связывания приложений с внешними ресурсами. Доступ к платформе открыт для всех желающих по адресу [2].

В рамках дальнейших исследований планируется расширить функциональность платформы и предложить подходы к решению следующих задач:

- эффективное планирование выполнения заданий на множестве разнотипных ресурсов,

- публикация композитных приложений,
- поддержка параллельных и распределенных приложений,
- интеграция с другими типами ресурсов (desktop grid, облака).

Список литературы

[1] Sukhoroslov O., Afanasiev A. Everest: A Cloud Platform for Computational Web Services. In Proceedings of the 4th International Conference on Cloud Computing and Services Science (CLOSER 2014). SCITEPRESS - Science and and Technology Publications, 2014, pp. 411-416.

[2] Everest. http://everest.distcomp.org/

[3] Richardson L., Ruby S. RESTful Web Services. O'Reilly, 2007.

[4] Afanasiev A., Sukhoroslov O., Voloshinov V. MathCloud: Publication and Reuse of Scientific Applications as RESTful Web Services // Lecture Notes in Computer Science Volume 7979. Springer 2013. pp. 394-408

[5] Volkov S., Sukhoroslov O. A Generic Web Service for Running Parameter Sweep Applications // Proceedings of Information Technology and Systems – 2014 (An IITP RAS Conference & School, September, 1-5, Nizhny Novgorod, Russia). ISBN 978-5-901158-25-8. pp. 285-289

[6] Fette I., Melnikov A. The WebSocket Protocol. RFC 6455, Internet Engineering Task Force, 2011.

Об авторе:

Сухорослов Олег Викторович

Кандидат технических наук, старший научный сотрудник лаборатории распределенных вычислительных систем Института проблем передачи информации РАН.

e-mail: sukhoroslov@iitp.ru

Образец ссылки на публикацию:

О. В. Сухорослов. *Интеграция вычислительных приложений и распределенных ресурсов на базе облачной программной платформы* // Программные системы: теория и приложения: электрон. научн. журн. 2014. Т. 4, № 3(17), с. ??–??.

URL: http://psta.psiras.ru/read/???

O. Sukhoroslov. *Integration of distributed computing applications and resources on the base of cloud platform.*

Abstract. Everest is a cloud platform for researchers supporting publication, execution and composition of applications running across distributed computing resources. The paper describes the platform's architecture, application development, integration with external computing resources and application composition using the platform's API.

Key Words and Phrases: distributed computing, cloud platform, web services, REST, integration of computing resources, application composition.