Принципы интеграции программных ресурсов в научных вычислениях[1]

В.В. Волошинов, С.А. Смирнов

Аннотация. Описываются принципы интеграции программных ресурсов в научных вычислениях на основе сервис-ориентированной архитектуры. Рассматриваются способы программной реализации на основе объектно-ориентированного ППО и в форме RESTful-веб-сервисов. Предлагаемые подходы демонстрируются на примерах создания сервисов оптимизации и компьютерной алгебры.

Ключевые слова: сервис-ориентированная архитектура, RESTful-веб-сервисы, распределенные вычисления, оптимизация, компьютерная алгебра.

Введение

К настоящему времени существует большой выбор технологий распределенных вычислений. К наиболее распространенным можно отнести: 1) инфраструктуру выполнения параллельных приложений для суперкомпьютеров на основе MPI (Message Passing Interface); 2) грид-системы [1], объединяющие географически и организационно распределенные ресурсы, для проведения вычислений на «виртуальном суперкомпьютере»; 3) т.н. «добровольные» (или «настольные») грид-сети (Desktop Grid), состоящие из множества персональных компьютеров, чьи вычислительные ресурсы задействуются во время простоя машины. Кроме того, современные графические карты для персональных компьютеров поддерживают технологии CUDA и/или OpenCL, позволяющие использовать сотни графических «микропроцессоров» одной карты для произвольных расчетов. Широкое применение получает модель «облачных» вычислений [2], предоставляющая пользователю по его запросу большую вычислительную мощность, при этом стоимость использования 1000 виртуальных машин в течение одного часа будет равна стоимости использования одной машины в течение 1000 часов.

Важной проблемой в использовании перечисленных средств высокопроизводительных вычислений является сложность их внедрения в повседневную практику научных исследований. Часто исследователь, вполне квалифицированный в своей области, сталкивается с проблемами при переносе вычислений в распределенную среду из-за недостаточного знания самих технологии и отсутствия требуемых навыков программирования.

Другой проблемой является использование стороннего программного обеспечения. Это могут быть свободно распространяемые или коммерческие библиотеки специализированных вычислений, например, системы компьютерной алгебры и символьных вычислений, пакеты численных методов оптимизации и т.п. Подобные средства могут быть использованы либо через соответствующее API, либо через вспомогательные приложения, принимающие файлы (в специальном формате) с описаниями задач и алгоритмов вычислений. В итоге, от

[1] Работа выполнена при поддержке ФЦП «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2013 годы» годы (Госконтракт № 07.514.11.4024).

пользователей требуется либо изучение программных интерфейсов, либо подготовка файлов требуемого формата.

Далее, предположим, что неким коллективом реализован некоторый уникальный алгоритм, Пусть это приложение требуется применять в рамках научной кооперации совместно с другим коллективом, также имеющим какие-то свои программные ресурсы. Возникает проблема удаленного использования приложений, если их перенос на «чужие» платформы затруднен. Постепенно складывается практика применения сервис-ориентированной архитектуры [3]: приложения оформляются в виде сервисов, удаленно доступных с помощью некоторого промежуточного программного обеспечения (ППО). Дальнейшая интеграция проводится на основе обмена данными между этими сервисами.

Если научные приложения доступны как сервисы с унифицированным интерфейсом, то новые «составные» приложения (по сути - сценарии вычислений) можно составлять из них как из готовых «строительных блоков». Применение декларативного языка, позволяющего задавать порядок передачи данных от сервиса к сервису, упрощает создание составных сервисов. Это же относится и к применениию редакторов, позволяющих строить составные сервисы в форме, например, workflow-диаграмм. Вообще, переход от универсальных императивных языков к декларативным проблемно-ориентированным языкам часто позволяет повысить скорость разработки специализированных приложений. В качестве примеров можно привести языки AMPL или GAMS для описания задач математического программирования и алгоритмов оптимизации, язык сценариев GNU Make для зависимостей между файлами при сборке программного проекта, язык SQL для запросов к реляционным базам данных, визуальное программирование в средах Max/MSP и PD для работы со звуком, видео и всевозможными средствами ввода в реальном времени и т.д.

Применение сервис-ориентированной архитектуры в научных исследованиях основано на различных технологиях. Изначально применялись технологии веб-сервисов (WSDL, SOAP) для грид-систем. Но эта архитектура может быть реализована и на основе других технологий. Например, на основе удаленных объектов (CORBA, Ice) или — архитектурного стиля REST и протокола HTTP — так называемых RESTful-веб-сервисов.

1. Сервис-ориентированная архитектура

Сервис-ориентированная архитектура — это форма архитектуры распределенных систем, обычно характеризуемая следующими свойствами [4]:

– логическое представление. Сервис представляет собой абстрагированное логическое представление реальных программ, баз данных, бизнес-процессов и т.п. Представление определяют через призму того, что делает сервис, производя высокоуровневую операцию;

– ориентированность на сообщения. Сервис формально определяется форматом входных и выходных сообщений, а не внутренним устройством программного агента (язык реализации, структура процессов, структура данных). При использовании СОА не важно, как работает агент, реализующий сервис. Избегая каких-либо подробностей о внутренней структуре агента, можно интегрировать любой программный компонент или приложение, который можно «обернуть» в код обработки сообщений, обеспечивающий соответствие формальному описанию сервиса;

– контрактный принцип. Сервис описывается машинно-читаемыми метаданными. Описание поддерживает открытую природу СОА: в описание включаются лишь те параметры, которые нужны для использования сервиса;

– гранулярность. Как правило, сервисы используют небольшое число операций с относительно большими и сложными сообщениями;

– ориентированность на применение в сети;

– платформенная независимость. Входные и выходные сообщения отправляются в платформенно-независимом стандартизованном формате через интерфейсы.

Модель «приложение-как-сервис» (Software-as-a-Service, SaaS), заключающаяся в оформлении приложения в виде удаленно доступного сервиса, обладает рядом преимуществ в срав-

нении с традиционной моделью распространения приложений. Владелец сохраняет полный контроль над приложением и средой его выполнения, что упрощает поддержку и обновление приложения, а также его коммерциализацию. Пользователю приложения, в свою очередь, не требуется производить закупку необходимого оборудования, установку, настройку и постоянную поддержку приложения. Для коммерческих сервисов затраты пользователя могут быть сведены к оплате за фактически выполненные вычисления.

Разработка проблемно-ориентированных Grid-сервисов затруднена сложностью освоения и применения существующего промежуточного ПО Grid, что требует специальной квалификации уже от разработчика приложения. Кроме того, вычисления в Grid имеют свою специфику в сравнении с традиционными высокопроизводительными вычислениями. Так, вообще говоря, заранее неизвестно на каком узле Grid будет запущено задание. При запуске задания могут происходить различные сбои, требующие его перезапуска. Для эффективного планирования вычислений требуется учитывать гетерогенность и иерархическую структуру ресурсов Grid. Отсутствуют высокоуровневые программные инструментарии, скрывающие от программиста детали взаимодействия с Grid и реализующие типовые схемы распределенных вычислений в Grid, что позволило бы уменьшить трудоемкость разработки приложений.

Как можно видеть, СОА не накладывает требований выбора определенных технологий. Более того, требования СОА достаточно общие и она может быть реализована средствами объектно-ориентированного ППО, например CORBA, Ice, RMI, технологиями Веб-сервисов (стандарты WS-*, ...), с помощью RESTful-веб-сервисов и т.д.

2. Объектно-ориентированное промежуточное программное обеспечение

В объектно-ориентированном ППО (CORBA, DCOM, Java RMI, ZeroC Ice и т.д.) взаимодействие между приложениями осуществляется путем вызова методов удаленных объектов. Интерфейс объекта описывают на том или ином декларативном языке определения интерфейсов. Эти описания интерфейсов (и типов данных) преобразуются специальным компилятором в готовые фрагменты программного кода, применяемые, как для реализации самих удаленных объектов, так и для их вызова из других приложений.

За счет возможности передачи ссылок на объекты и запуска нескольких объектов в одном процессе появляется возможность относительно просто создавать составные сервисы.

Идеологи объектно-ориентированных технологий распределенного программирования в начале 90-х годов ставили перед собой задачу добиться интероперабельности приложений написанных на разных языках программирования и функционирующих на различных вычислительных платформах за счет унификации форматов сообщений, передаваемых по сети. Заметным достижением для CORBA было применение протоколов GIOP/IIOP. Одновременно считалось, что модель программирования распределенных систем не должна, по возможности, отличаться от «традиционного» стиля программирования «монолитных» приложений. Следует сказать, что перечисленные выше технологии (особенно, это относится к CORBA и Ice), фактически обеспечили достижение указанных целей.

Но, со временем, стало ясно, что сетевые вычисления имеют свою специфику. Задачи упаковки и распаковки данных для передачи по сети не являются здесь основными. Истинными проблемами распределенных вычислений являются: возможность отказов отдельных программных компонент; сетевые сбои; отсутствие централизованного управления ресурсами; необходимость в синхронизации результатов, поступающих от удаленных «решателей» и т.п. Исследователи, начинающие разрабатывать распределенное приложение для решения своей «прикладной проблемы», вскоре обнаруживают, что в основном занимаются не ею, а перечисленными «программистскими» задачами [5].

3. RESTful-веб-сервисы

В настоящее время становится популярным способ реализации сервис-ориентированной

архитектуры, основанный на архитектурном стиле REST, реализованном в протоколе HTTP.

MathCloud – это распределённая среда, ориентированная на поддержку математических исследований и базирующаяся на технологиях Web и грид [6]. Цель среды состоит в предоставлении унифицированного доступа к сетевым сервисам решения различных классов математических задач. Для унификации механизма удалённого доступа к сервисам в MathCloud на уровне протоколов и форматов данных используется архитектурный стиль REST (Representational State Transfer). Архитектурный стиль REST лежит в основе современного World Wide Web. В частности, протокол HTTP обеспечивает ряд основных «ограничений» REST, при этом REST представляет собой ресурс, идентификатор и представление ресурса. Web-сервисы, удовлетворяющие ограничениям REST, называют RESTful веб-сервисами [8].

В унифицированном REST-интерфейсе MathCloud под сервисом понимается одна операция обработки входных данных, оформленная в виде ресурса [9]. Для вызова операции пользователь использует метод POST протокола HTTP. В ответ сервисом создается веб-ресурс «задание», позволяющий получить статус и результаты выполнения задания клиента. Таким образом, в текущей реализации интерфейса MathCloud все запросы обрабатываются асинхронно. Метод GET-сервиса позволяет получить его описание. Также в интерфейсе предусмотрен способ передачи параметров и результатов запросов в форме файлов.

Для поддержки интеграции сервисов при решении прикладных задач в состав инструментария MathCloud входит система управления сценариями на основе workflow-подхода. Данная система реализует описание, хранение, выполнение и публикацию сценариев совместного использования сервисов. Сценарии описываются в виде ориентированных ациклических графов при помощи визуального редактора. Созданный сценарий автоматически становится новым RESTful-сервисом, формально неотличимым от остальных. Разработанная система имеет распределенную клиент-серверную архитектуру. Клиентская часть системы включает редактор сценариев, а серверная – сервис управления выполнением сценариев.

Сама идея использования визуальных редакторов для создания сценариев совместного использования сервисов довольно популярна. В настоящее время, помимо инструментария MathCloud, существуют и развиваются такие среды, как P-GRADE, Taverna, Triana и другие.

Система MathCloud представляет собой логичное развитие инструментариев IARnet [13] и IARnet2 [14], основанных на технологиях объектно-ориентированного ППО с применением популярных в настоящее время следующих технологических решений:

– описание интерфейсов сервисов производят на языке JSON (Java Script Object Notation);

– поддерживаются реализации сервисов в виде приложений командной строки. Такие приложения крайне распространены в научной среде. Дополнительным плюсом является реализация отложенной передачи параметров-файлов при вызове сервисов. Передается лишь URI-ссылка на файл;

– редактор сценариев, написанный на JavaScript и работающий в браузере, заметно повышает привлекательность инструментария;

– система управления сценариями обеспечивает выполнение сценариев, создаваемых в редакторе. Работая отдельно от редактора, она обеспечивает независимое выполнение сценариев, позволяя следить за состоянием запущенного задания по уникальному URI;

– присутствует возможность создания сервисов на языке Java. Однако разработка сервисов командной строки происходит намного проще и сами сервисы работают стабильнее благодаря отсутствию внутреннего состояния.

4. Сервисы оптимизации

С применением инструментариев IARnet, IARnet2 и MathCloud было разработано несколько сервисов оптимизации. Сервисы оптимизации используют солверы GLPK, lp_solve, Ipopt, Bonmin [11, 12].

GLPK – это разработанный доцентом МАИ, к.т.н. А.О. Махориным LP/MILP солвер, который реализует различные модификации симплекс метода, включает вариант метода внут-

ренней точки и для решения целочисленных задач использует методы отсечений и ветвей-границ. Важной особенностью пакета является наличие встроенного транслятора с алгебраического языка оптимизационного моделирования GNU MathProg, разработки того же автора.

Lp_solve, LP/MILP солвер, разрабатывается уже более десяти лет в Эйндховенском технологическом университете, Нидерланды. К настоящему времени доступна пятая версия, выпущенная в 2005 году и постоянно развиваемая до сих пор.

Ipopt, NLP солвер – один из немногих современных пакетов, реализующих метод внутренней точки. Солвер широко применяется в других системах оптимизации для решения NLP-задач, входит в состав других пакетов проекта COIN-OR, например, Bonmin.

Bonmin, MINLP солвер, является одним из наиболее развитых пакетов проекта COIN-OR и одним из наиболее современных универсальных пакетов решения дискретных задач, который реализует различные варианты методов решения выпуклых MINLP-задач.

Пакеты Lp_solve, Ipopt и Bonmin совместимы с трансляторами языков оптимизационного моделирования AMPL и GAMS.

При работе с сервисами оптимизации в системах IARnet возникали следующие трудности:

– необходимость отдельно реализовывать механизмы передачи начальных данных и результатов в виде файлов;

– необходимость, в случае IARnet2, компилировать описание интерфейса сервиса либо производить обобщенный вызов сервиса в случае IARnet.

В случае MathCloud создание сервиса оптимизации происходит предельно просто, а именно:

– разрабатывается JSON-описание («дескриптор») интерфейса сервиса, содержащее типы входных и выходных параметров;

– в том же дескрипторе определяются правила преобразования параметров при работе сервиса. Например, значение входного параметра перед вызовом пакета может быть записано в файл. Аналогично задают правила для результатов;

– наконец, создается shell-скрипт, например, на языке Bash, производящий подготовку данных и собственно вызов солвера.

5. Сервисы компьютерной алгебры

Для систем IARnet и инструментария MathCloud были разработаны несколько версий сервиса символьных вычислений на основе системы компьютерной алгебры Maxima [10].

Особенностью первых версий сервисов Maxima было неявное наличие состояния у сервиса. Так удавалось избежать затрат на запуск самой системы компьютерной алгебры. В случае операций, выполняемых за короткое время в сервисах на основе объектно-ориентированного ППО, такая оптимизация целесообразна.

С другой стороны, при работе с сервисами MathCloud накладные расходы на один вызов уже сравнимы с запуском системы Maxima (как консольного приложения). Да и корректное взаимодействие с постоянно запущенными процессами Maxima требовало довольно сложного программного кода. Оказалось целесообразнее использовать пакетный режим работы с Maxima. Также упрощается отмена заданий, так как достаточно просто завершить сам процесс системы компьютерной алгебры. Кроме того, на основе накопленного опыта применения Maxima, оказалось, что обработку результатов расчетов удобнее также делать в Maxima и передавать данные между вызовами с помощью файлов в «родных» форматах системы. Таким образом, отпадала необходимость в большей части функциональности сервисов, обладающих внутренним состоянием, и стало возможно заменить их предельно простыми сервисами командной строки, обменивающимися файлами с данными и результатами.

Заключение

Для поисковых научно-исследовательских проектов, где требуется разработка проблемно-ориентированных распределенных вычислительных систем, применение RESTful-веб-сервисов заметно сокращает трудоемкость программной разработки. Они позволяют относительно легко получить и, при необходимости, изменить, действующий прототип распределенного вычислительного сценария для решения той или иной вычислительной задачи.

Литература

1. Foster I., Kesselman C. The grid: blueprint for a new computing infrastructure. Morgan Kaufmann, 2004.
2. Fox A., Griffith R. et al. Above the clouds: A Berkeley view of cloud computing // Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS. 2009. Vol. 28.
3. Foster I. Service-oriented science // Science. 2005. Vol. 308, no. 5723. Pp. 814–817.
4. Booth D., Haas H., McCabe F. et al. Web services architecture. 2004. URL: http://www. w3.org/TR/2004/NOTE-ws-arch-20040211.
5. Kendall S., Waldo J., Wollrath A., Wyant G. A Note on Distributed Computing, Sun Microsystems // Inc., Mountain View, CA. 1994.
6. Астафьев А., Афанасьев А., Лазарев И. и др. Научная сервис-ориентированная среда на основе технологий Web и распределенных вычислений // Труды Всероссийской научной конференции (г. Новороссийск) «Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность». М.: Изд-во МГУ, 2009. С. 463–467.
7. Fielding R. Architectural styles and the design of network-based software architectures: Ph. D. thesis / University of California. 2000.
8. Richardson L., Ruby S. RESTful web services. O'Reilly Media, 2007.
9. Сухорослов О. Унифицированный интерфейс доступа к алгоритмическим сервисам в Web // Проблемы вычислений в распределенной среде. 2009. Т. 46 из Труды ИСА РАН. С. 60–82.
10. Voloshinov V.V., Smirnov S.A. Error-Free Inversion of Ill-Conditioned Matrices in Distributed Computing System of RESTful-Services of Computer Algebra // Distributed Computing and Grid-Technologies in Science and Education: Proceedings of the 4th Intern. Conf. (Dubna, June28-July 3, 2010). – Dubna: JINR, Д-11-2010-140, 2010, pp. 257-263
11. В.В. Волошинов, В.С. Неверов, С.А. Смирнов. Использование распределенной среды REST-сервисов для решения трудоемких вычислительных задач // Материалы XVII Международной конференции по вычислительной механике и современным прикладным программным системам (ВМСППС'2011), 25-31 мая 2011 г., г. Алушта. - М.: Изд-во МАИ-ПРИНТ, 2011, с.235-237
12. В.В. Волошинов, В.С. Неверов. Обработка данных рентгеновской дифрактометрии наноматериалов в распределенной среде REST-сервисов // Журнал "Информационные технологии и вычислительные системы", №4, 2011, с. 10-20
13. Волошинов В.В., Естехин О.С., Сухорослов О.В. Архитектура и принципы организации системы IARnet. В сб.: «Проблемы вычислений в распределенной среде. Модели обработки и представления данных. Динамические системы». Труды ИСА РАН. – М.: КомКнига, 2005.
14. Sukhoroslov O.V. On using Ice middleware in the IARnet framework. // XXI International Symposium on Nuclear Electronics & Computing (NEC'2007) (Varna, Bulgaria, September 10-17, 2007): Proceedings of the Symposium. - Dubna: JINR, 2008. - pp. 405-411.

Волошинов Владимир Владимирович. Заведующий лабораторией Института системного анализа РАН. Окончил Московский физико-технический институт в 1984 году. Кандидат физико-математических наук. Автор более 50 печатных работ. Область научных интересов: теория и численные методы оптимизации, программно-алгоритмическое обеспечение научных исследований, технологии распределенных вычислений. E-mail: vladimir.voloshinov@gmail.com, vv_vol@isa.ru

Смирнов Сергей Андреевич. Аспирант Московского физико-технического института. Автор 6 печатных работ. Область научных интересов: распределенные вычисления. E-mail: sasmir@gmail.com