ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ №?, 2015, с. ??–??

УДК 004.75

В. В. Волошинов, В. С. Неверов, С. А. Смирнов

Интеграция программных средств оптимизационного моделирования в неоднородной вычислительной среде на основе системы AMPLX

Аннотация. Рассматривается проблема создания распределенных систем оптимизационного моделирования на основе уже существующих программных средств: 1) пакетов численных методов для базовых классов задач математического программирования; 2) трансляторов высокоуровневых языков описания оптимизационных моделей и сценариев расчетов по эти моделям (например, AMPL, GAMS, Mosel и др.). Дается краткий обзор развития программных технологий по данной теме. Предлагается программная технология, основанная на REST-сервисах доступа к указанным пакетам и транслятору языка оптимизационного моделирования AMPL. Это позволит модифицировать любую AMPL-программу так, чтобы при выполнении ее обычным AMPL-транслятором решение всех задач будет выполняться пулом «удаленных» сервисов оптимизации, причем независимые подзадачи будут обрабатываться параллельно. Разработанная система AMPLX основана на программной платформе «облачного типа» Everest, http://everest.distcomp.org, и доступна в исходных кодах (http://gitlab.com/ssmir/amplx). В качестве примера рассматриваются декомпозиционные алгоритмы Данцига-Вульфа, Бендерса, а также алгоритм типа ветвей и границ для решения одной нелинейной задачи оптимизационной идентификации наноструктрных параметров углеродных пленок из вакуумной камеры установки термоядерного синтеза Т-10 по данным нейтронной и рентгеновских дифрактометрий.

Ключевые слова и фразы: задачи оптимизации, декомпозиционные алгоритмы, языки оптимизационного моделирования, язык AMPL, распределенные вычисления, REST-сервисы.

Введение

Одним из направлений наших исследований является развитие программных технологий и методик их применения для создания проблемно-ориентированных вычислительных систем на основе распределенной вычислительной среды сервисов оптимизационного

Поддержано грантом РФФИ 13-07-00987.

моделирования (ОМ). Здесь под сервисами ОМ подразумевается унифицированные программные компоненты, обеспечивающие удаленный доступ к пакетам численных методов решения задач математического программирования и трансляторам алгебраических языков, предназначенных как для описания самих оптимизационных моделей, так и сценариев расчетов на основе этих моделей (AMPL, GAMS, Mosel–Xpress, Zimpl и др. [1]). Использование таких языков значительно упрощает процесс обмена данными с пакетами численных методов (ввода исходных данных оптимизационной задачи, получение результатов решения), позволяет описывать оптимизационные модели в достаточно «естественной», символьной форме, дает возможность программировать достаточно сложные вычислительные сценарии расчетов на основе одной или нескольким задачам оптимизации (возможно, динамически формируемым в ходе расчетов).

Основной задачей является разработка средств интеграции указанных программных ресурсов (установленных в неоднородной вычислительной инфраструктуре исследовательского коллектива) в проблемно-ориентированные вычислительные системы, обеспечивающие автоматическое выполнение трудоемких сценариев расчетов. А именно: «прозрачное» изменение вычислительной мощности (подключение/отключение вычислительных ресурсов); динамическое формирование новых подзадач и обмен результатами их решения; возможность параллельного решения независимых подзадач; обмен данными с другими приложениями (например, для импорта параметров из баз данных, визуализации результатов и пр.).

1. Краткий обзор тенденций развития технологий оптимизационного моделирования

К настоящему времени создан обширный парк базового программного обеспечения оптимизационного моделирования, который можно разбить на две основные группы: 1) коммерческие и свободно доступные пакеты численных методов; 2) высокоуровневые средства оптимизационного моделирования на основе трансляторов алгебраических языков оптимизационного моделирования (AMPL, GAMS, Zimpl, Mosel-Xpress . . .). Применение указанных трансляторов значительно сокращает трудоемкость основных этапов работы с оптимизационными моделями: 1) формального описания исходных оптимизационных задач и сценариев вычислений по этим моделям; 2) ввода исходных данных и обработки результатов решения. В недавно вышедшем сборнике

статей по теме прикладного оптимизационного моделирования [1] отмечается высокая актуальность развития указанных языков. Основные из них, AMPL, GAMS, Mosel-Xpress стали, де-факто, «общепринятыми» стандартами, применяемыми как в «научно-исследовательских» так и в «промышленных» целях. Разработчики наиболее популярных библиотек численных методов (как коммерческих, так и с «открытым кодом»), оснащая свои пакеты соответствующими интерфейсами, делают их AMPL/GAMS-совместимыми. В наших исследованиях мы применяем систему AMPL (an Algebraic Mathematical Programming Language), www.ampl.com, [2].

1.1. Языки оптимизационного моделирования

Работы по созданию алгебраических языков оптимизационного моделирования ведутся уже более 30 лет (в англоязычной литературе, для их обозначения используется термин AML, Algebraic Modelling Language). Основными составляющими систем на основе AML являются: собственно язык для описания оптимизационных моделей, средства автоматического дифференцирования и унифицированный интерфейс взаимодействия с пакетами. Эти системы обеспечивают автоматизацию рутинных и весьма трудоемких этапов оптимизационного моделирования: символьное задание исходных данных оптимизационных задач (параметров, переменных, ограничений и целевых функций); подстановка конкретных значений параметров, в т.ч. из баз данных; вызов того или иного пакета оптимизации; получение и, если необходимо, обработка результатов решения.

Исторически первым таким языком, по-видимому, был GAMS (General Algebraic Modeling System), www.gams.com [1, 3, 4]. По данным из обзора [4], первое его упоминание датируется 1976 годом, когда был сделан доклад на конференции в Будапеште. Работы по созданию были изначально инициированы Международным Банком реконструкции и развития. Первоначальными «заказчиками» работ были экономисты и финансисты, испытывавшие потребность в решении сложных экономических задач. Это обстоятельство обусловило специфический синтаксис языка GAMS, непривычный для инженеров, научных сотрудников и других специалистов, имеющих опыт программирования на языках высокого уровня. С 1987 года язык стал коммерческим продуктом, разработка и распространение которого ведет GAMS Development Corporation, www.gams.com.

Основным «конкурентом» GAMS является язык AMPL (A Modeling Language for Mathematical Programming), www.ampl.com, [1, 2, 4]. Его разработка была начата в Bell Laboraties с середины 80-х годов прошлого века группой специалистов, в состав которой входил один из известнейших специалистов в области разработки программного обеспечения Брайан Керниган (соавтор Денниса Ричи по популярной книге «Язык программирования C»). По-видимому, это обстоятельство обусловило более привычный для инженеров и ученых синтаксис языка, который одновременно похож на язык набора математических формул TeX и сильно упрощенный вариант языка C.

Оба языка AMPL и GAMS позволяют записывать соотношения оптимизационных задач (параметры, переменные, целевую функцию, ограничения и т.д.), допуская разделение «символьного» описания задачи (т.н. «модельное» представление) и набора числовых значений параметров. Последние «подставляются» в модель в результате работы специального транслятора, на вход которого поступают модель задачи и конкретные значения параметров. На выходе - специальная структура данных (в виде файла), адаптированная для дальнейшего автоматического дифференцирования и непосредственно передаваемая в пакеты оптимизации. Следуя терминологии, принятой в AMPL, в дальнейшем, такой файл, содержащий все необходимое для запуска процедуры решения, мы будем называть стабом (stub, стаб-файлом). Указанные трансляторы, также предоставляют доступ к результатам работы численных алгоритмов. Более того, предусмотрены средства импорта данных из реляционных СУБД, что критически важно при работе с действительно большими задачами промышленного уровня. На рис. 1 представлена общая схема применения AML-языков для ввода данных в пакеты оптимизации, на примерах AMPL и GAMS. Видно, что с этой точки зрения оба языка, практически, идентичны. Язык Mosel-Xpress [13] также, во многом, соответствует указанной схеме. Однако он имеет ряд особенностей, которые мы отметим позже.

1.2. Системы оптимизации в распределенной вычислительной среде

Актуальность переноса систем оптимизационного моделирования в среду распределенных вычислений вызвана следующими обстоятельствами. Алгоритмы решения многих классов оптимизационных задач представляют собой многоэтапные итеративные и/или рекурсивные процедуры, сводящиеся к решению динамически формируемых

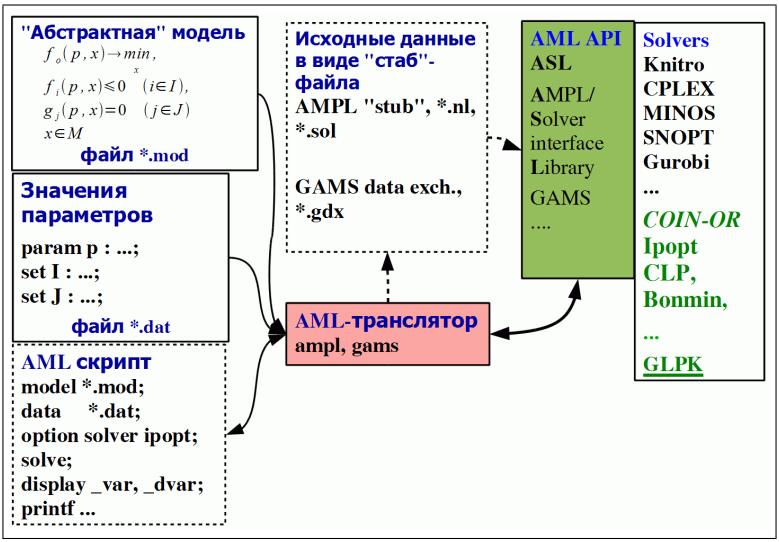

Рис. 1. Общая схема применения алгебраических языков оптимизационного моделирования

наборов вспомогательных подзадач. К таким классам относятся: оптимизация систем с блочной структурой (что, например, характерно для транспортных задач или задач размещения производственных мощностей); анализ многоуровневых иерархических систем; параметрическая оптимизация (например, в исследованиях по формированию предварительного облика технических систем). Для реализации соответствующих вычислительных алгоритмов, необходимо предложить удобные программные средства, Для решения большого числа промежуточных подзадач, в т.ч. независимых друг от друга, естественно образом ставит проблему запуска и интенсивных обменов данными между пакетами оптимизации, функционирующими на отдельных серверах, кластерах, суперкомпьютерах, узлах Грид-систем или в облачных инфраструктурах.

Тенденция к применению программного обеспечения как сервисов в распределенных вычислительных системах является одним из ведущих направлений развития информационных технологий уже более 15 лет. Надо отметить, что сам термин сервис, применительно к интеграции специализированных приложений, используемых в научных исследованиях, появился позже после широко известной

публикации Яна Фостера [5]. В последнее время, в связи с бурным развитием облачных вычислений, появилась устойчивая аббревиатура SaaS (Software-as-a-Service). Уровень развития программных технологий в области оптимизации и технологий распределенных вычислений уже давно позволяет приступить к практической реализации сформулированных целей. Однако мировой уровень развития распределенных систем оптимизации, как ни странно, невысок. Фактически, единственным и хорошо известным примером применения оптимизационных пакетов как удаленных сервисов был и остается портал "NEOS: server for optimization", http://www.neos-server.org. Портал NEOS функционирует по клиент-серверной модели, позволяя отправлять одиночные задачи оптимизации, заданные в формате одного из языков оптимизационного моделирования (фактически, AMPL и/или GAMS), одному из пакетов, установленных на серверах NEOS. Последние версии основаны на технологии XML-RPC, [6]. Для работы с серверами оптимизации NEOS предусмотрена специальное клиентское приложение Kestrel [6], напоминающее по своей функциональности консоль интерпретатора AMPL. Несмотря на высокую популярность сервиса (только на головном сайте проекта, www.neos-server.org, в год решается 200-300 тысяч задач), лишь несколько процентов из них было отправлено через Kestrel. Это вызвано тем, что протокол, предлагаемый в «паре» NEOS-Kestrel, предусматривает постоянный обмен большими объемами данных промежуточных расчетов между серверами и рабочей станцией пользователя Kestrel. Используемый здесь XML-формат представления данных является избыточным для вычислительных задач и только увеличивает объем сетевого трафика. Кроме того, NEOS не имеет средств обмена данными с другими приложениями, что часто необходимо в расчетах по оптимизационным моделям, например, для подготовки исходных данных и обработки результатов решения задач оптимизации. Можно сделать вывод о том, что ПО NEOS-Kestrel не удобно для создания разнообразных распределенных систем оптимизационного моделирования.

В конце 2006 года, в рамках работ над проектом COIN-OR (Computational Infrastructure for Operations Research), возникла инициатива OS (Optimization Services), http://www.optimizationservices.org, [7, 8]. Ее целью было создание широкого набора спецификаций OS*L (см. Рис. 1), покрывающих весь круг проблем, связанных с созданием систем оптимизационного моделирования, от описания задач и обнаружения сервисов-решателей, до управления ходом вычислений

и обмена результатами решения. Следуя сложившейся в середине десятилетия «моде» на повсеместное (но далеко не всегда оправданное) применение Веб-сервисов, был разработан набор спецификаций OSxL (в форме XML-схем), покрывающих основные потребности оптимизационного моделирования. Для описания и управления выполнением распределенных схем вычислений предполагалось использовать язык BPEL, Business Process Execution Language, www.oasis-open.org/committees/wsbpel. К сожалению, до удобных программных реализаций дело еще не дошло (или уже не дойдет?). Возможно, это связано с недостатками, изначально присущими «традиционной» концепции Веб-сервисов, основанной на «связке» стандартов SOAP, WSDL, UDDI, XML, в перспективности которой сейчас возникают большие сомнения. Также возникли трудности с применением языка BPEL, как по причине его сложности для освоения даже квалифицированными пользователями, так и ввиду отсутствия нетребовательной к ресурсам и свободно распространяемой (некоммерческой) среды выполнения. Потенциальные возможности, заложенные в стандарты на основе языков XML и BPEL, являются избыточными для распределенных систем оптимизационного моделирования. Их применение осложняет достижение поставленных целей из-за излишних требований к «переучиванию» пользователей, привыкших к «традиционным» языкам и системам оптимизационного моделирования.

В связи с этим актуальной является возможность разработки систем оптимизационного моделирования на основе архитектурного стиля REST [9] и имеющихся наработок в этой области. Сервисы оптимизации могут создаваться в форме REST-сервисов. Для управления ходом вычислений предлагается использовать созданный в нашем коллективе программный инструментарий разработки распределенных сервис-ориентированных систем Everest, `http://everest.distcomp.org` [10, 11] (См. далее).

Еще одним интересным проектом является система оптимизационного моделирования Pyomo [12]. Она основана на языке популярном «неспециализированном» языке программирования Python, но пару лет назад Pyomo стал совместимым со стандартом AMPL. В рамках проекта Pyomo ведутся работы по переносу вычислений в распределенную среду на основе модуля Pyro, `http://pyro.sourceforge.net`. Однако, на наш взгляд, недостатком предлагаемого подхода является необходимость описания «распределенного» сценария расчетов также на языке Python. Это не позволяет использовать уже имею-

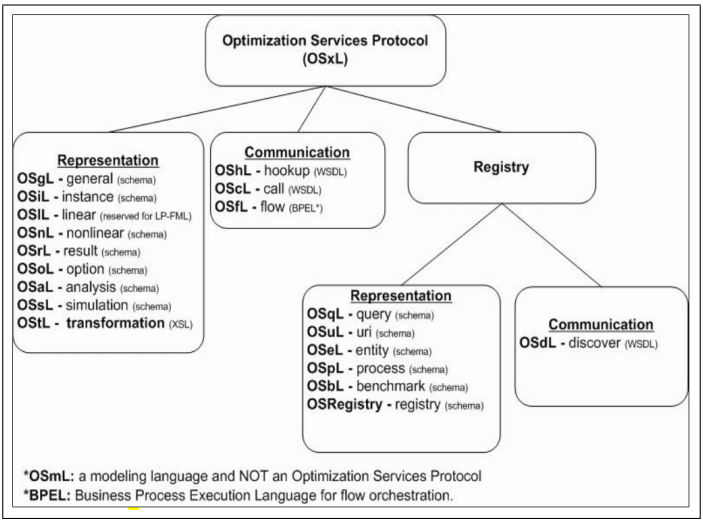

Рис. 2. Спецификации проекта COIN-OR Optimizatin Services

щийся «корпус» описаний алгоритмов, например, на том же языке AMPL. Напомню, что предлагаемый нами подход как раз основан на модификации уже существующих текстов на языке AMPL.

В контексте «распределенной оптимизации» нельзя не отметить систему Mosel-XPRESS [1, 13], разрабатываемой на принципах коммерческого ПО компанией Fico, fico.com. Эта система основана на языке опт. моделирования Mosel (с 2001 года доступен по коммерческой лицензии) и линейке пакетов оптимизации Xpress. С 2010 года публике была представлена новая, существенно переработанная версия языка и транслятора Mosel, где в максимальной степени (по сравнению с языками AMPL и GAMS) были обеспечены возможности программирования распределенных сценариев расчетов по оптимизационным моделям [13]. Фактически, непосредственно на высокоуровневом языке Mosel, пользователи могут программировать распределенный алгоритм расчетов на основе событийно-ориентированной (Event Driven) парадигме многопоточного программирования (см. рис. 3). Здесь следует отметить, что язык Mosel, в отличие от AMPL, GAMS и других языков, интерпретируемых транслятором, требует компиляции в некий промежуточный «байт-код», выполняемый специальной виртуальной машиной. Для обмена данными между одновременно

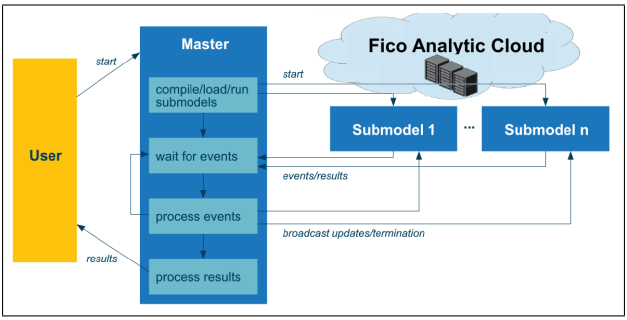

Рис. 3. Событийно-ориентированная модель программирования на языке Mosel-Xpress

выполняемыми потоками среда выполнения Mosel предлагает выбор из двух, протоколов SHMEM (в архитектуре общей памяти) и MEMPIPE (обмен данными между процессорами). Оба эти протокола являются высокоэффективными, но низкоуровневыми, что обуславливает необходимость в однородной вычислительной архитектуре (многоядерный сервер, высокопроизводительный кластер). Последнее обстоятельство привело к тому, что в настоящее время компания Fico предлагает использовать Mosel-Xpress в форме «частного» облачного сервиса, развернутого на «серверной ферме», поддерживаемой самой компанией Fico.

1.3. О возможностях распараллеливания в расчетах по оптимизационным моделям

В сложившейся практике расчетов по оптимизационным моделям можно выделить много ситуаций, когда процесс вычислений можно ускорить за счет параллельного решения независимых подзадач. Перечислим некоторые из них.

Первая относится к многовариантным расчетам на основе одной задачи, но с разными значениями ее параметров (т.н. «parameter sweeping» в англоязычной литературе). После решения большого набора однотипных задач, обычно проводится сравнение полученных решений с целью выбора подходящего сочетания параметров исследуемой оптимизационной модели.

Второй случай соответствует реализации декомпозиционных алгоритмов для задач с блочной структурой, наиболее известными из

которых являются алгоритмы Данцига-Вульфа или Бендерса [14]. Схема алгоритмов такого типа – это итеративное чередование двух этапов: 1) поиск решения одной «координирующей задачи» (master problem), имеющей меньшую размерность, чем исходная; 2) формирование и поиска решения набора независимых подзадач (также, меньшей размерности, чем исходная). Следует отметить, что результаты решения координирующей задачи (прямые и двойственные переменные) используются для формирования подзадач и, обратно, найденные решения подзадач частично определяют параметры master задачи.

Алгоритмы типа «ветвей-и-границ» также хорошо подходят для распараллеливания, поскольку их работа сводится к решению динамически формируемых очередей подзадач: т. н. «оценочных» (для сокращения области перебора) и/или тех, которые используются для получения т. н. «рекордов» (оценок неизвестного оптимального значения целевой функции) [14, 15]. После анализа полученных решений формируются новые наборы подзадач, пока, либо не выяснится неразрешимость задачи, либо не будет получено допустимое решение с гарантированной погрешность отклонения от оптимального значения критерия (точное значение которого может остаться неизвестным).

В работах, посвященных системе GAMS [16, 17] для обозначения типового «программного шаблона» (design pattern), применяемого в алгоритмах декомпозиционного типа была введена полезная аббревиатура GUSS (Gather-Update-Solve-Scatter1). Здесь: Gather соответствует сбору информации о решении набора задач; Update – обновлению данных на основании полученной информации; Solve – поиску решения новых задач (или одной задачи); Scatter – передачи результатов решения (см. рис. 4).

2. Система AMPLX - возможность выполнять сценарии расчетов на языке AMPL в режиме распределенных вычислений

Следует сказать, что среди трех коммерческих AML-систем (AMPL, GAMS, XPRESS-Mosel) только AMPL-транслятор до сих пор не предусматривает никаких встроенных средств распределенной реализации шаблона GUSS. Реализация GUSS в GAMS-трансляторе описана в работах, [16, 17]. Возможности системы XPRESS-Mosel значительно больше соответствуют современным подходам к организации

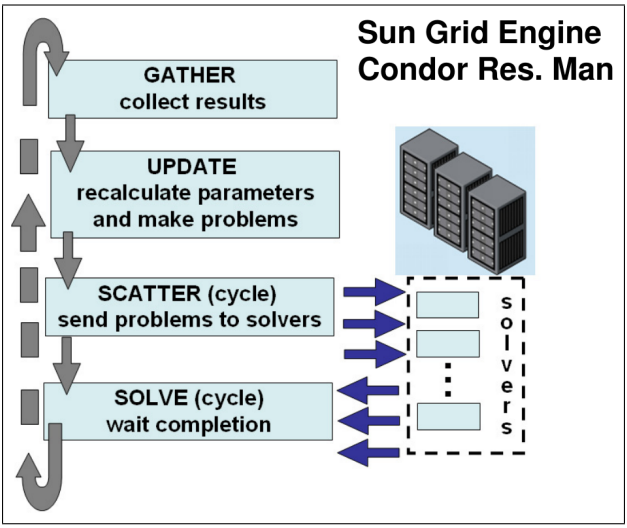

Рис. 4. GUSS – типовой «шаблон» выполнения расчетов по оптимизационным моделям

распределенных вычислений (в частности, поддерживается событийно-ориентированная модель программирования, Message-Driven design pattern). Это не удивительно, т.к., как уже отмечалось выше, система XPRESS-Mosel значительно «моложе» AMPL и GAMS. Однако, ее практическое применение в поисковых НИР осложняется отсутствием Mosel-совместимых солверов с открытым кодом. Кроме того, в настоящее время компания Fico, разработчик XPRESS-Mosel ориентируется на применение своего ПО в специальной облачной инфраструктуре. В тоже время, синтаксис языка AMPL содержит все необходимые элементы для описания «GUSS-сценариев» расчетов по оптимизационным моделям (динамическое формирование AMPL-стабов, поддержка циклов, условные операторы, загрузка результатов решения из *.sol файлов и т.п.). Отсутствует только поддержка сетевого обмена исходными данными и результатами. В ходе проводимой нами работы эти проблемы были решены средствами программного инструментария Everest и Everest Python API.

2.1. Сведения о программном инструментарии Everest

Работы по созданию собственного программного инструментария для программирования и развертывания систем на основе REST-сервисов ведутся в нашем коллективе около пяти лет. Первоначальная и последующая стабильная версии имели название MathCloud, http://mathcloud.org. Последние два года велась активная работа по переходу на новую версию программного инструментария т.н. Everest [**10**, **11**], http://everest.distcomp.org. Общая архитектура Everest представлена на рис. 5. Имея общее назначение - разработка унифицированных RETS-сервисов с возможностью их интеграции в вычислительные сценарии; - предыдущий MathCloud и новый Everest имеют ряд важных отличий.

Комплект ПО Everest является гораздо более зрелым с точки зрения поддержки его разработчиками. Это система с открытым кодом свободно доступным на популярном портале gitlab.com. В семантике Everest появилось новая иерархия понятий:

- *приложение Everest* - собственно REST-сервис с точно определенным REST-интерфейсом записанным в формате JSON; приложение Everest, вообще говоря, является абстракцией, «за спиной» которого может не быть никакого реального вычислительного ресурса (в этом случае выбор и подключение ресурсов происходит непосредственно перед вызовом);
- *вычислительный ресурс Everest* - реальное вычислительное устройство или инфраструктура (сервер, кластер, грид, облако), где будет производиться обработка данных Everest-приложениями;
- *агент доступа к вычислительным ресурсам Everest* - программный модуль (на Python), который делает вычислительный ресурс доступным для подключения к системе Everest (некоторые основные типы ресурсов, представлены на рис. 6);
- *сервер Everest (контейнер приложений)* - центральный сервер, осуществляющий: сохранение дескрипторов всех приложений, регистрацию пользователей, управление правами доступа к созданным приложениям, управление очередями заданий;
- *веб-интерфейс работы с сервером Everest*, включающий средства создания приложений, запуска и контроля за ходом выполнения заданий.

Перечислим ряд особенностей Everest, важных с точки зрения практического развертывания и применения систем оптимизационного

моделирования в распределенной вычислительной инфраструктуре.
1. Автор приложения имеет возможность «прозрачно», т.е. незаметно для пользователей остальных пользователей, повышать/понижать вычислительную «мощность» приложений (сервисов), изменяя список ресурсов (фактически - агентов доступа к ресурсам), ассоциированных с данным приложением. Например, если некий пакет оптимизации будет установлен на новом вычислительном сервере вместе с агентом Everest, то мощность унифицированного сервиса решения задач оптимизации (приложения) увеличится, если указанный агент будет добавлен в список ресурсов приложения. Например (см. рис. 7), сервис решения задач мат. программирования, представленных своим AMPL-стабом, располагает тремя вычислительными ресурсами, представленными тремя агентами доступа (здесь - к одному реальному и к двум виртуальным многоядерным серверам).
2. Платформа Everest получила унифицированный программный интерфейс (Everest Python API), `https://gitlab.com/everest/python-api`. Это позволяет клиентским модулям на языке Python взаимодействовать с сервисами Everest по модели асинхронных вызовов удаленных объектов. Библиотека типов и методов, представленная в Everest Python API, позволяет программировать на Python вычислительные сценарии координированной обработки данных несколькими приложениями Everest, причем независимые задания будут выполняться одновременно несколькими приложениями (или одним приложением, но - на разных вычислительных ресурсах, подключенных к этому приложению !). В предыдущих версиях MathCloud, авторы систем оптимизационного моделирования должны были сами разработать подобное API, включая алгоритм балансировки вычислительной нагрузки. В Everest все эти функции берет на себя сервер Everest.
3. Подсистема балансировки вычислительной нагрузки управляет выполнением заданий, распределяя их между вычислительными ресурсами, подключенными к одному приложению. Пользователю не нужно знать какие именно ресурсы обрабатывают его данные. Эта подсистема постоянно совершенствуется разработчиками Everest, в частности, ожидается возможность выбора различных политик распределения заданий между ресурсами. **4.** Система контроля доступа к приложениям и защиты данных в Everest основана на сочетании двух технологий: защищенного протокола обмена данными между Everest-сервером и агентами по протоколу SSH; системы специальных ключей, т.н. токенов, которые выдаются зарегистрированным пользо-

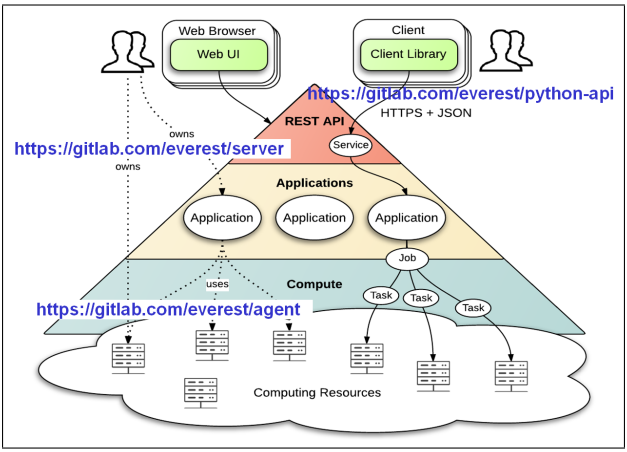

Рис. 5. Архитектура программного инструментария Everest

вателям Everest. Предъявление токенов обязательно, как для работы с Веб-интерфейсом сервера, так и при вызове приложений через Everest API. Токены имеют ограниченный срок действия (сейчас 7 дней). В Everest API предусмотрены специальные средства для автоматического обновления токенов клиентскими приложениями (например, вычислительными сценариями), если истек срок действия предыдущих. **5.** В «базовой» конфигурации Everest, файлы с результатами выполнения заданий «образуются» на вычислительных ресурсах, в рабочих каталогах установленных там агентов Everest. По завершению задания эти файлы пересылаются агентами на центральный сервер, где и становятся доступными для передачи другим приложениям. Если два Everest-приложения «основаны» на программах, которые установлены на одном хосте, то обмен данными между соответствующими приложениями будет идти через центральный сервер. При интенсивных обменах данными это будет снижать производительность создаваемых распределенных систем. В будущем году планируется «расшить» это узкое место, предоставив агентам обмениваться файлами непосредственно между собой (если это допускает политика безопасности хоста, где установлен агент).

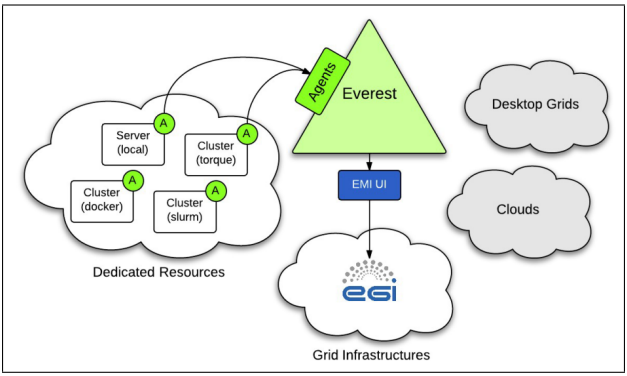

Рис. 6. Вычислительные ресурсы, подключаемые к Everest

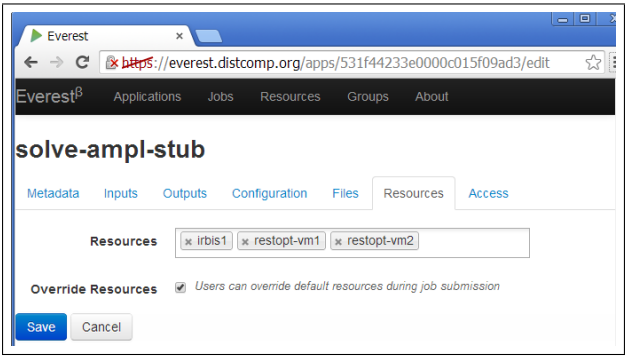

Рис. 7. Ресурсы, подключенные к одному из базовых приложений AMPLX (сервис решения задачи, представленной в форме AMPL-стаба)

2.2. Сервисы решения задач математического программирования

Были разработаны несколько типовых сервисов (или, в терминологии Everest, приложения) описания двух из которых приводятся в следующих разделах. Все сервисы были созданы с помощью мастера создания Everest-приложений, доступного в браузере и реализованного на JavaScript.

Сервис ampl-stub. Это сервис генерации AMPL-стабов на основании символьного описания задачи математического программирова-

ния в файле *.mod и, возможно, дополнительного файла с числовыми параметрами задачи в файле *.dat

БУДУТ ДОБАВЛЕНЫ СВЕДЕНИЯ о сервисе

Сервис solve-ampl-stub. Это сервис решения задач мат. программирования, представленной AMPL-стабом заданным пакетом численных методов, включая возможность передачи настроек пакета. В настоящее время сервис обеспечивает унифицированный доступ к следующим пакетам:

- Ipopt (Coin-OR **I**nterior **P**oint **OPT**imizer, NLP), https://projects.coin-or.org/Ipopt;
- CBC (**C**oin-OR **B**ranch-and-**C**ut, LP, MILP), https://projects.coin-or.org/Cbc;
- SCIP (**S**olving **C**onstraint **I**nteger **P**rograms, LP, MILP, MINLP), http://scip.zib.de (По нашей просьбе разработчики пакета SCIP внесли в его код некоторые модификации, чтобы обеспечить корректную запись оптимальных множителей Лагранжа в файл найденного решения в AMPL-формате).

Этот минимальный набор пакетов обеспечивает поддержку решения основных типов задач математического программирования (LP/MILP/NLP). Для поддержки решения нелинейных частично-целочисленных задач планируется подключить AMPL-совместимый пакет: Bonmin (COIN-OR Basic Open-source Nonlinear Mixed INteger programming, MINLP), https://projects.coin-or.org/Bonmin.

Результатом работы приложения являются два файла: solve–ampl–stub.log.txt (консольная выдача пакета при поиске решения) и problem.sol (найденное решение в AMPL-формате, включающее оптимальные значения переменных и множителей Лагранжа).

2.3. Сведения о программной архитектуре AMPLX

Напомним основные требования к программной реализации системы (предназначенной для интеграции системы AMPL в распределенную среду приложений Everest):

- возможность выполнения любых программ (сценариев расчетов) на языке AMPL, корректно выполняемых AMPL-транслятором, возможно, после некоторой модификации самой программы согласно определенному набору правил;
- возможность обработки заранее неизвестного числа подзадач, заранее неизвестным набором ресурсов (список задач и доступных сервисов

являются параметрами задания, а состав доступной вычислительной инфраструктуры может меняться);
- ускорение процедуры обработки набора подзадач на основе того или иного алгоритма балансировки вычислительной нагрузки в условиях неопределенности (заранее неизвестной трудоемкости подзадач и неизвестной производительности сервисов, зависящих от характеристик вычислительных серверов и фактической их загруженности другими заданиями);
- простота интеграции с AMPL-транслятором и применение интерпретируемых языков программирования (bash-shell и Python), для упрощения программной реализации.

Прототип подобной системы был разработан в 2012 году на основе предыдущей версии программного инструментария Everest (известной под названием Mathcloud) [18–20], где отсутствовал унифицированный программный интерфейс асинхронного взаимодействия с сервисами оптимизации, средства прозрачного наращивания вычислительной мощности и балансировки вычислительной нагрузки. Созданная в этом году система состоит из четырех элементов:
- набора «макросов», amplx_*.amp, на языке AMPL (т. н. amplx-адаптер); - набора модулей на Python, которые посредством Everest Python API обеспечивают взаимодействие с сервисом оптимизации solve-ampl-stub (см. выше);
- пула вычислительных ресурсов, подключенных к сервису solve-ampl-stub посредством агентов доступа Everest, запущенных на узлах доступной вычислительной инфраструктуры;
- специального Everest-приложения run-amplx, обеспечивающего доступ к созданной системе для зарегистрированных пользователей.
Логическая структура системы изображена на рис. 8.

Типовой «приём» распараллеливания алгоритма оптимизации, записанного на языке AMPL, выглядит следующим образом. Пусть данный этап алгоритма заключается в решении набора независимых подзадач, зависящих от некоторого параметра p , принимающего значения из заданного множества его значений **SetP**:

```
option solver ipopt #выбор AMPL-совестимого солвера>
option ipopt_options ‘‘acceptable_tol=10e-8 ...’’ # опции солвера
# операторы AMPL
# ...
for {p in SetP} {
# операторы AMPL, предшествующие формированию задач SubProb[p]
  solve SubProb[p];
```

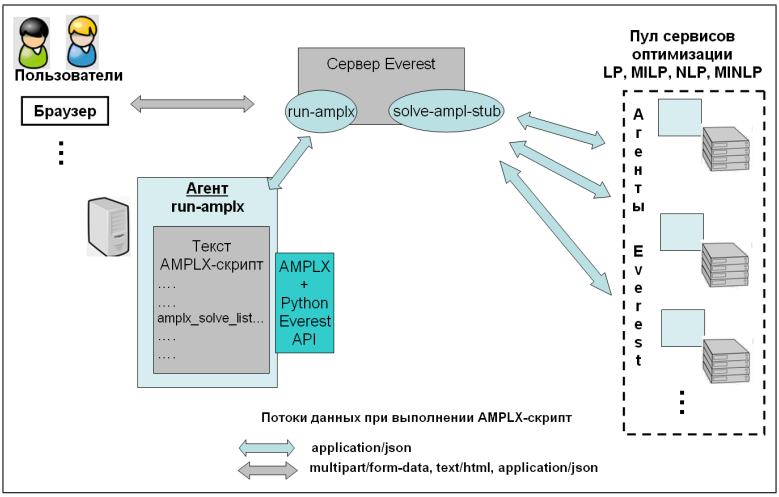

Рис. 8. Программная архитектура AMPLX

```
# операторы, использующие результаты решения SubProb[p]
}
# операторы продолжения сценария опт. моделирования
```

В этом случае «шаблон» модификации кода можно кратко сформулировать так *«один цикл (операторы AMPL* `for` *или* `loop` *) заменить на три фрагмента:*
1) цикл формирования набора подзадач в форме AMPL-стабов;
2) асинхронная параллельная обработка стабов одним приложением Everest (сейчас — solve-ampl-stub) с подключенным к нему пулом AMPL-совместимых пакетов численных методов;
*3) цикл обработки результатов, доставленных AMPL-транслятору в виде набора файлов *.sol с решениями подзадач в AMPL-формате».*
Модифицированный фрагмент кода выглядит так (синим шрифтом выделены операторы и комментарии, относящиеся к AMPLX):

```
commands amplx_globals.amp; #однократная инициализация AMPLX «в начале»
let __amplx_solver := "ipopt"; #выбор AMPL-совместимого Everest-солвера
let __amplx_solver_opts_file := "ipopt.opt"; #имя файла с опциями пакета
# операторы AMPL (без изменений)
# ...
let __amplx_probSet := {}; # очистка списка задач для AMPLX
for {p in SetP} { # цикл формирования списка подзадач
#  (те же !) операторы AMPL, предшествующие формированию задач SubProb[p]
```

```
  problem SubProb[p]; # переключение в контекст подзадачи из списка
  let __amplx_probName := ("SubProb_" & p); # имя подзадачи для AMPLX
  #запись стаба и обновление списка
  commands amplx_write_problem_stub.amp;
}
# параллельное решение списка задач
commands amplx_solve_problems_list.amp;
for {p in SetP} { # цикл обработки результатов решения списка подзадач
#  (те же !) операторы AMPL, предшествующие формированию задач SubProb[p]
  problem SubProb[p]; # переключение в контекст подзадачи из списка
  solution ("SubProb_" & p & ".sol"); # загрузка результатов решения
#  (те же !)операторы, использующие результаты решения SubProb[p]
}
# операторы продолжения сценария опт. моделирования
```

Общая методика применения AMPLX состоит в следующем. В корректно составленную программу на AMPL вставляются специальные команды вида `commands amplx_*.amp` из набора шаблонов amplx-адаптера, которыми следует заменить «обычные» операторы AMPL, связанные с решением одной или нескольких задач оптимизации. Имеются ввиду, либо одиночная команда `solve someProblem`, либо какой-нибудь цикл, где нужно решить набор независимых задач:

```
for {p in setProblems} {
  solve p;
  ... /* некоторые операторы, использующие результаты решения */
}
```

Первая команда заменяется операторами

```
let __amplx_probName := someProblem;
commands amplx_solve_problem.amp;
```

а цикл заменяется на два цикла с одним оператором «посередине» как это уже было показано выше «Внутри» макросов `amplx_*.amp` используется AMPL-команда shell, вызывающая программу на Python, где, собственно и происходит асинхронное взаимодействие с сервисом оптимизации для решения одной или нескольких задач. Распределение вычислительной нагрузки осуществляется Everest-сервером. Файлы с найденными решениями *.sol записываются в тот же каталог, где работает AMPL-транслятор и могут быть загружены в AMPL-программу командами solution <имя_проблемы>.sol Важно, что AMPLx записывает решения различных задач в файлы с различными именами (именами задач в AMPL-программе), что позволяет корректно отождествлять имена задач с полученными файлами *.sol.

2.4. AMPLx-адаптер

Amplx-адаптер представлен четырьмя небольшими файлами (`amplx_*.amp`), содержащими группы операторов AMPL, «унифицированных» в соответствии с предлагаемыми нами (разработчиками AMPLX) соглашениями о названиях «amplx»-переменных. А именно, мы «резервируем» для AMPLX названия, которые начинаются с префикса «`__amplx`» Все эти *.amp файлы подключаются к AMPL-программам посредством стандартного в AMPL-оператора commands <имя файла с командами> Назначение файлов адаптера отражено в из названиях, а логика работы определяется составом операторов (весьма небольшим).

amplx_globals.amp

Содержит объявления глобальных переменных.

```
param __amplx_solver, symbolic, default "ipopt"; # ipopt, bonmin, scipampl, cbc
param __amplx_solver_opts_file, symbolic, default ""; #
param __amplx_outFilePath, symbolic, default "userout.txt";

param __amplx_solve, symbolic;
param __amplx_probName, symbolic;
param __amplx_probList, symbolic; # List of problems (names) as STRING
set  __amplx_probSet;         # List of problems (names) as SET
let  __amplx_probSet := {};     # Init as EMPTY

if length($AMPLX_SOLVER) > 0 then {
let __amplx_solver := $AMPLX_SOLVER;
};

if length($AMPLX_OUTFILE) > 0 then {
let __amplx_outFilePath := $AMPLX_OUTFILE;
};
```

amplx_solve_problem.amp

Решение одной задачи удаленными солверами. Название задачи - в переменной `__amplx_probName`. Заменяет оператор `solve (__amplx _probName)`

```
# All below are symbolic parameters wich must be defined before
# __amplx_probName - name of the problem

let __amplx_probSet := ;
commands amplx_write_problem_stub.amp;
commands amplx_solve_problems_list.amp;
solution (__amplx_probName & ".sol");
```

amplx_solve_problems_list.amp

Параллельное решение нескольких задачи сервисами оптимизации. Список названий задач должен быть помещен в параметр `__amplx_probSet`.

```
# All below are symbolic parameters wich must be declared before
# __amplx_solve  - remote solve command
# __amplx_probList - list of problems (stubs) name separated by blank,
#               i.e. "sp1 sp2"
# __amplx_probSet - list of problems (stubs) name as AMPL-Set, i.e. sp1 sp2

let __amplx_probList := "";
for { p in __amplx_probSet } {
  let __amplx_probList := sprintf("%s %s", __amplx_probList, p);
  }

let __amplx_solve := sprintf('python %s %s %s -- %s',
  $AMPLX_SOLVE_RUNNER, __amplx_solver, __amplx_solver_opts_file, __amplx_probList);
shell (__amplx_solve);
```

amplx_write_problem_stub.amp

Создание стаба задачи, с названием, сохраненным в `__amplx_probName`, и запись его на диск в файл с именем <название задачи>.nl, с одновременным пополнением набора задач `__amplx_probSet`. Удобно применять перед вызовом `amplx_solve_problems_list`.

```
# All symbolic parameters wich must be declared before
write ("g" & __amplx_probName);
#add problem name of written stub to __amplx_probSet
let __amplx_probSet := __amplx_probSet union {__amplx_probName};
```

2.5. Экспериментальная вычислительная инфраструктура сервисов оптимизационного моделирования в Everest

Для отладки созданного ПО была создана экспериментальная вычислительная инфраструктура. Помимо, управляющего сервера everest.distcomp.org, она включала один многоядерных сервер (irbis1) и два виртуальных многоядерных сервера, развернутые на серверах Центра распределенных вычислений под управлением супервизора «хостинга» виртуальных систем Xen Mahanger. Сведения о технических характеристиках, установленных элементах ПО Everest и ПО, относящегося к оптимизации приводятся ниже в таблицах 1 и 2.
Вычислительный сервер irbis1: OS Linux (x86_64) Red Hat; CPU Intel(R) Xeon(R) E5620 @ 2.40GHz (16 ядер); память 16 Gb.
Два виртуальных сервера (xen-main, xen-copy): OS Linux (x86_64)

Таблица 1. Everest-агенты сервисов оптимизации и их параметры

Everest-агент, URL (https://everest.distcomp.org/resources/*)	Сервер и максимальное число одновременно выполняемых заданий
irbis-agent *=52eb8d4e420000f401166a2e	irbis1 4
irbis-agentLight 5448cf663400006e18502721	irbis1 24
restopt-vm1 544e54293300003f0038c674	xen-main 6
restopt-vm2 544e82673300003f0038c687	xen-copy 6

Ubuntu; CPU Intel(R) Xeon(R) CPU E5-2620 0 @ 2.00GHz (8 ядер); память 8 Gb.

На всех серверах было установлено следующее ПО:
- Python 2.7;
- пакеты оптимизации: Ipopt 3.11.7, CBC 2.8.8, SCIP 3.1.0 (с патчем, добавленным по нашей просьбе разработчиками для корректной записи множителей Лагранжа в файл с найденным решением);
- Everest-агенты для подключения созданных Everest-приложений (сервисов оптимизации).

Следует сказать, что для устранения возможных проблем с взаимной блокировкой очереди заданий Everest (а при выполнении задания для run-amplx система AMPLX динамически создает очереди заданий для сервисов solve-ampl-stub) на сервере irbis1 было использовано два агента, irbis_agent и irbis_agentLight:

Таким образом, сконфигурированная система, в ходе выполнения заданий для приложения run-amplx позволяет одновременно решать 16 задач математического программирования

3. Апробация системы AMPLx на примере декомпозиционного алгоритма Данцига-Вульфа

Рассматривается задача доставки конечного набора продуктов из пунктов их исходного размещения (производства, складирования и т.п.) в места потребления (назначения, доставки) по имеющейся транспортной сети [14]. Стоимость транспортировки зависит как от

Таблица 2. Сервисы оптимизации и подключенные к ним ресурсы

Everest-приложение, URL (https://everest.distcomp.org/apps/*)	используемые ресурсы
ampl-stub *=52eba4944200001102166a30	irbis-agentLight
solve-ampl-stub 531f44233e0000c015f09ad3	irbis-agent, restopt-vm1, restopt-vm2
run-amplx 544a686e3400001f3a502771	irbis-agentLight

вида продукта, так и от мест размещения и назначения и линейно зависит от объемов перемещаемого продукта. Критерием качества решения является суммарная стоимость транспортных расходов, причем требуется учесть ограничения на пропускную способность транспортной сети.

БУДУТ ДОБАВЛЕНО ФОРМАЛЬНОЕ ОПИСАНИЕ ЛП-ЗАДАЧИ

Задачи указанного вида имеют многочисленные практические приложения в логистике и исследовании свойств коммуникационных сетей. Особенностью таких задач является возможность их решения методами декомпозиционного типа (Данцига-Вульфа, Бендерса и т.п.). Описания и теоретическое обоснование таких методов можно найти во многих работах по методам оптимизации.

Эти алгоритмы сводятся к итеративной процедуре из повторяющихся двух этапов: 1) одновременного решения нескольких независящих друг от друга задач линейного программирования (соответствующих отдельным продуктам); 2) решения одной «координирующей» задачи линейного программирования. Причем, во–первых, исходные данные второй задачи зависят от результатов решения задач 1-го этапа, во–вторых, после решения задачи 2-го этапа изменяются целевые функции «однопродуктовых» задач. Надо также отметить, что размерность задач 1-го этапа и координирующей задачи 2-го этапа заметно меньше, чем у исходной задачи линейного программирования. Что и является обоснованием применения указанного алгоритма в литературе (см. рис. 9).

Указанный сценарий решения хорошо подходит для проверки работоспособности технологий распределенных вычислений по оптимизаци-

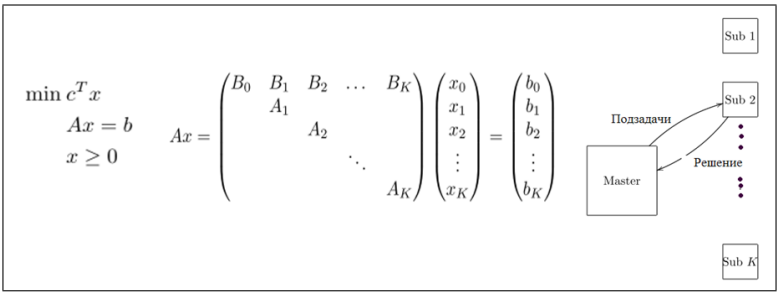

Рис. 9. Схема работы алгоритма Данцига-Вульфа для задачи с блочной структурой

онным моделям. Более того, разработчики языков оптимизационного моделирования GAMS и AMPL, для демонстрации возможностей своих разработок, включают реализации декомпозиционных алгоритмов в список типовых примеров [2].

Особенность декомпозиционных алгоритмов в том, что, в ходе их выполнения, решения одной (т.н. координирующей или - агрегирующей) задачи математического программирования чередуется с решением набора независимых подзадач, исходные данные которых зависят от результата решения координирующей задачи. После решения всех подзадач, результаты этих решения используются для изменения исходных данных координирующей задачи и весь цикл повторяется заново. При этом, после решения координирующей задачи на каждой итерации проверяется некоторое условие остановки, позволяющее гарантировать, что уже найдено либо точное, либо приближенное с заданной погрешностью решение исходной «основной» задачи. Надо также отметить, что размерность подзадач и координирующей задачи заметно меньше, чем у исходной задачи линейного программирования. Что и является основанием для сокращения времени вычислений (особенно на этапе одновременного решения независимых подзадач) в режиме распределенных вычислений, когда одновременно применяются несколько пакетов оптимизации.

Демонстрационная AMPL-программа multi2.run, хотя и реализует декомпозиционный алгоритм, но предусматривает последовательное решение (в цикле) независимых подзадач «локальным» пакетом, запускаемым на том же компьютере, где работает AMPL-транслятор. Цель эксперимента состояла в том, чтобы взять эту готовую AMPL-

программу и «заставить» ее работать на основе разработанной системы AMPLX в режиме распределенных вычислений, когда решение всех задач оптимизации, возникающих по ходу работы алгоритма Данцига-Вулфа, осуществлялось удаленными Everest-приложением solve-ampl-stub (с учетом подключенных к нему вычислительных ресурсов многие подзадачи будут решаться параллельно).

Нами была использована задача multi2, входящая в состав примеров на сайте AMPL в разделе “LOOPING AND TESTING 2: implementing algorithms through AMPL scripts ”, http://www.ampl.com/NEW/LOOP2/. Символьная модель и параметры задачи приведены в ниже. AMPL-программа, реализующая алгоритм Данцига-Вульфа для этой задачи также присутствует на сайте AMPL (см. файл multi2.mod и multi2.run). multi.2.run - исходная программа на языке AMPL, реализующая алгоритм Данцига-Вульфа. В результате применения к исходному тексту «правил AMPLX» были получены модифицированные (для трех различных пакетов Ipopt, CBC и SCIP, подключенных к solve-ampl-stub) варианты программы multi2.run: multi2_amplx_cbc.amp, multi2_amplx_ipopt.amp и multi2_amplx_scip.amp. Все тексты доступны по ссылке http://dcs.isa.ru/~vladimirv/restopt/amplx/dw/. Запуск всех трех AMPLX-сценариев завершался тем, что алгоритм выдавал решение , совпадающее с тем, что получалось в ходе работы исходной AMPL-программы multi2.run. Это подтвердило корректность предложенной схеиы переноса AMPL в распределенный режим работы.

4. Реализация в системе AMPLx алгоритма ветвей-и-границ для одной непрерывной нелинейной невыпуклой задачи

Идентификация структурного состава углеродистых наноматериалов в диапазоне размеров 1-10 нм по результатам рентгеновской дифрактометрии в диапазоне значений модуля вектора рассеяния q от нескольких единиц до нескольких десятков обратных нанометров представляют практический интерес для определения нанометровых углеродных структур в порошковых образцах, пленках и различных углеродистых наноматериалах. Интерпретация результатов синхротронной рентгеновской дифрактометрии на углеводородных пленках из токамака Т-10 потребовала массовых вычислений рентгеновских дифрактограмм от множества углеродных наноструктур различных топологий и размеров: фуллерены; углеродные нанотрубки различных радиусов, киральностей и длин; аксиально симметричные эллипсоиды

(включая сферы); тороидальные углеродные нанотрубки с малым аспектным отношением и эллиптическим сечением; все возможные полуфрагменты всех перечисленных структур; различные многослойные структуры, включая графеновые хлопья и пр.

Применение процедуры оптимизационной идентификации [21] стало возможным благодаря методу [22] приближенного описания позиций углеродных атомов в искривленном графене. Этот метод был использован в комплексе кодов XaNSoNS, на основе которого в среде Mathcloud (www.mathcloud.org) были созданы веб-сервисы, позволяющие использовать XaNSoNS удаленно, а также интегрировать его модули в распределенные вычислительные сценарии, в том числе и в сценарии обработки экспериментальных данных рентгеновского рассеяния [23].

Следующим шагом в исследовании структурного состава является обобщение алгоритма рентгеновской дифрактометрии на идентификацию структурного состава углеродистых наноматериалов путем совместной обработки результатов рентгеновской и нейтронной дифрактометрии. Подробная физическая постановка задачи приводится в работе [24].

Рассматривается оптимизационная задача, в которой совместно минимизируются ошибки, представляющие собой разности между экспериментальными и модельными интенсивностями рассеяния для рентгеновской и нейтронной дифракции. Здесь мы рассмотрим случай однородных аморфных смесей чисто углеродных наноструктур, «свободных» атомов углерода (фактически входящих в углеводородную аморфную компоненту; при таком рассмотрении мы пренебрегаем корреляциями атомов водорода в углеводородных молекулах) и примесей. Обобщение излагаемого ниже алгоритма на более сложные химические смеси не представляет труда.

БУДУТ ДОБАВЛЕНЫ ФОРМУЛЫ (СМ. http://arxiv.org/pdf/1301.3418v1.pdf)

В сложившейся практике восстановления наноструктурного состава по результатам оптимизационной идентификации используются три критерия качества совпадения с результатами эксперимента: L_2 - минимизация суммы квадратов отклонений (по всем замеряемым значениям); L_1 - минимизация суммы абсолютных значений отклонений (модулей разностей); L_∞ - минимизация максимального отклонения.

Задача совместной оптимизационной идентификации структурно-

го состава образца по данных двух экспериментов (по каждому из трех критериев) оказалась невыпуклой задачей из-за наличия билинейных ограничений с участием одной переменной t. Непосредственное применение пакетов численных методов нелинейной оптимизации, например, Ipopt, для таких задач, вообще говоря, позволяет найти только локальные минимумы погрешностей. Поэтому для поиска приближенного решения (с заданной заранее точностью по значению целевой функции) был предложен алгоритм типа ветвей-и-границ, т.е. задача оптимизационной идентификации трактовалась как задача глобальной оптимизации. Ветвление алгоритма производилось по интервалам значений скалярной переменной t. Благодаря тому, что при фиксированном значении t задачи являются, либо линейными (для L_1 и L_∞), либо выпуклыми квадратичными (для L_2), поиск приближенно оптимальных решений проводился в результате параллельного решения набора задач для динамически формируемого набора значений параметра t. Для оценки точности нахождения оптимума использовался стандартный прием релаксации билинейных соотношений линейными неравенствами и формирования оценочных, также выпуклых задач, для все более коротких интервалов значений t. Решение этих оценочных задач также проводилось параллельно.

Вначале данный алгоритм был реализован на языке оптимизационного моделирования AMPL. Для распараллеливания решения независимых задач на многоядерном сервере использовалась утилита `parallel`, `http://www.gnu.org/software/parallel`. Решение задач линейного программирования проводилось пакетом CBC (для критериев L_1 , L_∞), а квадратичных (для критерия L_2) - пакетом Ipopt. После этого схема расчетов была модифицирована для работы в системе AMPLX. На основе нового AMPLX-сценария было создано Everest приложение optBnB-NLP-amplx, стартовая веб форма которого представлена на рис. 10.

При запуске задания, помимо файла с исходными данными (AMPL-dat) в поле «List of criteria» и других параметров расчета можно перечислить какие из трех критериев (L_1 , L_2 , L_∞) оптимизационной идентификации будут использоваться. Исходная AMPL-программа, реализующая специальный вариант метода ветвей-и-границ, была устроена так, что каждый из критериев порождал отдельный «подсценарий». Внутри каждого из сценариев генерация AMPL-стабов всех подзадач выполнялась последовательно (устранение этого «недостатка» и переход к параллельной генерации стабов потребовал бы очень

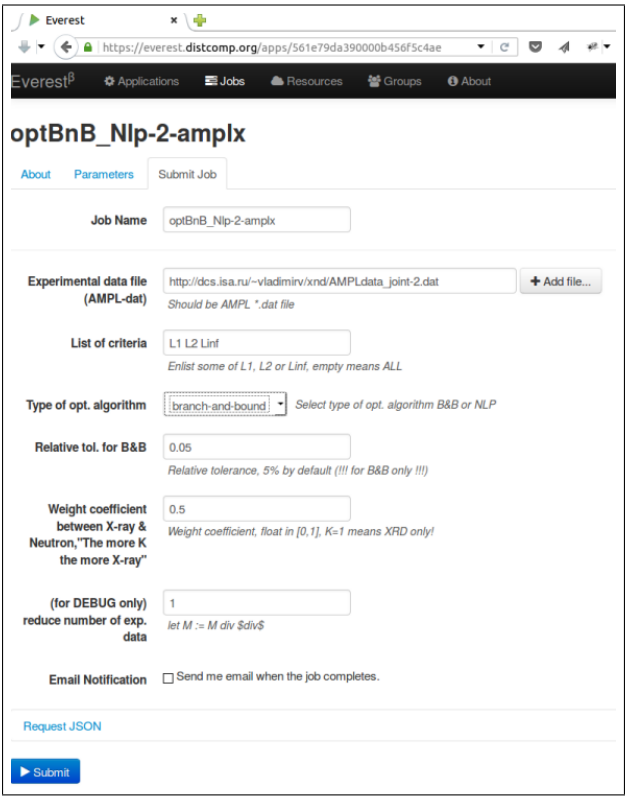

Рис. 10. Пример стартовой веб-формы запуска расчетов в системе AMPLX

серьезной работы). Затем начиналась фаза «параллельного» решения созданных подзадач, после которой происходило отсеивание заведомо «неперспективных», для поиска оптимального значения параметра t, интервалов его значений и дробление оставшихся интервалов, порождающих новые оценочные подзадачи. Эти циклы повторяются до тех пор, пока не будет гарантированно достигнута требуемая точность нахождения минимального значения погрешности аппроксимации.

Задание, где был «заказан» расчет по всем трем критериям занял на созданной экспериментальной вычислительной инфраструктуре ~26 минут. Это на шесть минут превышало продолжительность расчета с теми же исходными данными, когда для ускорения обработки

набора независимых подзадач применялась утилита `parallel`, использующая 8 ядер на 16-ти ядерном сервере irbis1. Учитывая, что (как было указано в подразделе 2.5) системе AMPLX было доступно 16 ядер, результат не выглядит обнадеживающим.

После некоторого анализа ситуации стало понятно как можно попытаться ускорить вычисления, учитывая указанную выше особенность работы AMPL-программы, включающей довольно продолжительные «участки» последовательных операций по генерации AMPL-стабов. А именно, посредством веб-формы, рис. 10, было одновременно запущено три задания, в каждом из которых «заказывался» расчет по одному(!) из критериев L_1 , L_2 или L_∞ . В результате все три задания были завершены в течении 11 минут!. Здесь проявились преимущества использования подсистемы балансировки вычислительной нагрузки, встроенной в сервер приложений Everest.

БУДУТ ДОБАВЛЕНЫ РИСУНКИ, ПОЯСНЯЮЩИЕ ЭТОТ ЭФФЕКТ

Заключение

Ранее предложенная методика выполнения произвольной программы на языке оптимизационного моделирования AMPL в распределенной среде была реализована на основе новой платформы создания REST-сервисов облачного типа, Everest, http://everest.distcomp.org. Созданная система включает сервисы оптимизации для основных типов задач математического программирования (линейных и нелинейных, включая задачи с частично-целочисленными переменными).

Фактически, создано расширение стандартного транслятора AMPL и методика несложной модификации произвольных сценариев расчетов на языке AMPL (включая алгоритмы декомпозиционного типа), обеспечивающие поддержку «типового шаблона» «распараллеливания» расчетов по оптимизационным моделям, т. н. GUSS (Gather-Update-Scatter-Solve). Встроенные в Everest средства балансировки вычислительной нагрузки и возможность «прозрачного» наращивания вычислительной мощности Everest-приложений (в данном случае - сервисов оптимизации из предыдущего раздела) позволило получить систему (мы назвали ее AMPLX, `http://gitlab.com/ssmir/amplx` сравнимую по своим функциональным возможностям с GAMS Grid и Mosel-XPRESS Optimization Suite (в том, что касается разработки распределенных систем оптимизационного моделирования).

Система AMPLX дополняет стандартный AMPL-транслятор, позволяющяя выполнять произвольные сценарии расчетов по оптимизационным моделям, когда решение всех задач математического программирования (динамически формируемых в ходе расчетов) будет выполняться пулом указанных выше сервисов оптимизации, причем независимые подзадачи решаются параллельно. Теперь пользователь может записывать произвольные алгоритмы оптимизационного моделирования непосредственно на языке AMPL, применяя несколько специальных «AMPL-макросов», согласно предложенной методике.

Программное обеспечение AMPLx было верифицировано на примерах распределенного выполнения декомпозиционных алгоритмов Данцига-Вульфа и Бендерса для задач линейного программирования с блочной структурой. Также на основе AMPLx был перенесен в распределенную среду алгоритм типа ветвей и границ для решения одной нелинейной задачи оптимизационной идентификации наноструктрных параметров углеродных пленок из вакуумной камеры установки термоядерного синтеза Т-10 по данным нейтронной и рентгеновских дифрактометрий.

Список литературы

[1] Kallrath J. Algebraic Modeling Systems: Modeling and Solving Real World Optimization Problems. Applied Optimization, vol. **104**: Springer Science & Business Media, 2012. — 254 p. ↑ 2, 3, 4, 8.

[2] Fourer R., Gay D.M., Kernighan B.W. AMPL: A Modeling Language for Mathematical Programming, second edition.: Duxbury Press/Brooks/Cole Publishing Company, 2002. — 538 p. ↑ 3, 4, 24.

[3] M. R. Bussieck, A. Meeraus. *General algebraic modeling system (GAMS)* // Modeling languages in mathematical optimization: Springer, 2004, p 137–157. ↑ 3.

[4] C. A. C. Kuip. *Algebraic languages for mathematical programming* // European Journal of Operational Research, 1993. Vol. **67**, no. 1, p 25–51. ↑ 3, 4.

[5] I. Foster. *Service-oriented science* // Science. Special Issue: “Distributed computing”, 2005. Vol. **308**, no. 5723, p 814–817. ↑ 6.

[6] E. D. Dolan, R. Fourer, J.-P. Goux, T. S Munson, J. Sarich. *Kestrel: An interface from optimization modeling systems to the NEOS server* // INFORMS Journal on Computing, 2008. Vol. **20**, no. 4, p 525–538. ↑ 6.

[7] R. Fourer, J. Ma, K. Martin. *Optimization services: A framework for distributed optimization* // Operations Research, 2010. Vol. **58**, no. 6, p 1624–1636. ↑ 6.

[8] R. Fourer, J. Ma, K. Martin. *OSiL: An instance language for optimization* // Computational optimization and applications, 2010. Vol. **45**, no. 1, p 181–203. ↑ 6.

[9] R. T. Fielding. *Architectural styles and the design of network-based software architectures*, University of California, Irvine, University of California, Irvine, (2000), http://www.ics.uci.edu/~fielding/pubs/dissertation/top. ↑ 7.

[10] O. V. Sukhoroslov, A. O. Rubtsov, S. Yu. Volkov. *Development of distributed computing applications and services with Everest cloud platform* // Computer Research and Modeling, 2015. Vol. **7**, no. 3, p 593–599. ↑ 7, 12.

[11] O. Sukhoroslov, S. Volkov, A. Afanasiev. *A Web-Based Platform for Publication and Distributed Execution of Computing Applications* // Parallel and Distributed Computing (ISPDC), 2015 14th International Symposium on IEEE, 2015, p 175–184. ↑ 7, 12.

[12] W. E. Hart, C. Laird, J.-P. Watson, D. L Woodruff. Pyomo–optimization modeling in python, vol. **67**: Springer, 2012. — 238 p. ↑ 7.

[13] S. Heipcke. *Xpress–Mosel:Multi–Solver, Multi–Problem, Multi–Model, Multi–Node Modeling and Problem Solving* // Algebraic Modeling Systems: Springer Science & Business Media, 2012. Vol. **104**, p 77–110. ↑ 4, 8.

[14] М. Мину. Математическое программирование. Теория и алгоритмы: Пер. с фр, и предисловие А. И. Штерна.: М.: Наука, 2002. — 488 с. ↑ 10, 22.

[15] Л. Д. Попов. *Опыт многоуровневого распараллеливания метода ветвей и границ в задачах дискретной оптимизации* // Автоматика и телемеханика, 2007, № 5, с 171–181. ↑ 10.

[16] M. Bussieck, M. C. Ferris, A. Meeraus. *Grid-enabled optimization with GAMS* // INFORMS Journal on Computing, 2009. Vol. **21**, no. 3, p 349–362. ↑ 10.

[17] M. R. Bussieck, M. C. Ferris, T. Lohmann. *GUSS: Solving Collections of Data Related Models Within GAMS* // Algebraic Modeling Systems: Springer, 2012, p 35–56., http://www.gams.com/modlib/adddocs/gusspaper.pdf. ↑ 10.

[18] V. V. Voloshinov, S. A. Smirnov. *On development of distributed optimization modeling systems in the REST architectural style* // Distributed Computing and Grid-Technologies in Science and Education: Proceedings of the 5th Intern. Conf. 16–21 July 2012. — Dubna, Moscow reg.: JINR, Dubna, 2012. ↑ 17.

[19] В. В. Волошинов, С. А. Смирнов. *Принципы интеграции программных ресурсов в научных вычислениях* // Информационные технологии и вычислительные системы, 2012, № 3, с 66–71. ↑ 17.

[20] V. V. Voloshinov, S. A. Smirnov. *Optimization Modeling in Heterogeneous Distributed Computing Infrastructure* // "20th Conference of the International Federation of the Operational Research Societies" (IFORS 2014), Barcelona, Spain, 13-18 July, 2014: report. — Barcelona, Spain, 2014. ↑ 17.

[21] A. B. Kukushkin, V. S. Neverov, N. L. Marusov, I. B. Semenov, B. N. Kolbasov, V. V. Voloshinov, A. P. Afanasiev, A. S. Tarasov, V. G. Stankevich, N. Yu. Svechnikov, etc. *Few-nanometer-wide carbon toroids in the hydrocarbon films deposited in tokamak T-10* // Chemical Physics Letters, 2011. Vol. **506**, no. 4, p 265–268. ↑ 26.

[22] L. A. Chernozatonskii, V. S. Neverov, A. B. Kukushkin. *A calculation model for X-ray diffraction by curved-graphene nanoparticles* // Physica B: Condensed Matter, 2012. Vol. **407**, no. 17, p 3467–3471. ↑ 26.

[23] В. В. Волошинов, В. С. Неверов. *Обработка данных рентгеновской дифрактометрии наноматериалов в распределенной среде rest сервисов* // Информационные технологии и вычислительные системы, 2011, с 10–20. ↑ 26.

[24] V. S. Neverov, V. V. Voloshinov, A. B. Kukushkin, A. S. Tarasov. *Algorithm of amorphous carbonaceous nanomaterial structure identification with a joint X-ray and neutron diffraction data analysis* // arXiv preprint arXiv:1301.3418, 2013. ↑ 26.

Об авторах:

Владимир Владимирович Волошинов

Зав. лаб. Ц-3 ИППИ РАН, старший научный сотрудник, к.ф.-м.н. Научные интересы: теория и практика применения оптимизационных моделей в разных областях науки и техники; различные вопросы распределенных вычислений

e-mail: vladimir.voloshinov@gmail.com

Владислав Сергеевич Неверов

н.с., к.ф.-м.н.; НИЦ «Курчатовский институт». Научные интересы: физика плазмы и наноструктур, распределенные вычисления, численное моделирование физических процессов

e-mail: vs-never@mail.ru

Сергей Андреевич Смирнов

И.о. научного сотрудника лаборатории № Ц-1 ИППИ РАН, к.т.н. Научные интересы: распределенные вычисления, облачные вычисления, программные аспекты применения пакетов оптимизации.

e-mail: sasmir@gmail.com

Пример ссылки на эту публикацию:

В. В. Волошинов, В. С. Неверов, С. А. Смирнов. «Интеграция программных средств оптимизационного моделирования в неоднородной вычислительной среде на основе системы AMPLX», *Программные системы: теория и приложения*, 2015, **??**:?, с. ??–??.

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

Vladimir Voloshinov, Vladislav Neverov, Sergey Smirnov. *Integration of Optimization Modelling Software in the Heterogeneous Computing Environment via AMPLX system.*

Abstract. We discuss the problem of developing a distributed systems for optimization modelling on the basis of already existing software: 1) solvers for basic types of mathematical programming problems; 2) interpreters of a high-level algebraic modelling and scripting languages (e.g., AMPL, GAMS, Mosel etc.). After a brief survey of state-of-the-art of the issue we describe our approach, which is based on REST-services providing remote access to above solvers and AMPL-interpreter. The technology enables to modify any AMPL-script in such a way that being run by the same AMPL-interpreter all subproblems will be solved by a pool of remote solvers and independent subproblems will be solved in parallel. Our, rather simple and "open source" AMPLX toolkit (http://gitlab.com/ssmir/amplx) is based on the Everest computing platform, http://everest.distcomp.org. AMPLX system have been verified via well-known Danzig-Wolfe and Benders algorithms. Also a kind of branch-and-bound algorithm for special nonlinear optimization problem of structure identification of amorphous carbonaceous nanomaterials with a joint x-ray and neutron diffraction data analysis.

Key Words and Phrases: optimization problems, decomposition algorithms, optimization modelling languages, AMPL language, distributed computing, REST-services.

Sample citation of this publication:

Vladimir Voloshinov, Vladislav Neverov, Sergey Smirnov. "Integration of Optimization Modelling Software in the Heterogeneous Computing Environment via AMPLX system", *Program systems: theory and applications*, 2015, **??**:?, pp. ??–??. (*In Russian.*) URL http://psta.psiras.ru/read/