УДК 004.633, 004.032.24, 004.032.34, 004.042

УПРАВЛЕНИЕ ФАЙЛАМИ В РАМКАХ МОДЕЛИ ПОТОКОВ ДАННЫХ ДЛЯ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ

А.М. НАЗАРЕНКО[1], Н.О. ПЕРЕСТОРОНИН[1], А.А. ПРОХОРОВ[1]
[1]ООО «ДАТАДВАНС», г. Москва, Россия

Системы управления вычислениями, основанные на модели потоков данных, в настоящее время широко используются для автоматизации различных инженерных и научных расчетных задач, требующих совместного использования в рамках одного расчета целого ряда специализированных программных средств. Такая система обеспечивает интеграцию используемого программного обеспечения в единую расчетную схему, передачу данных между элементами схемы, управляет их запуском и обработкой данных. В настоящей работе рассматривается случай управления распределенными асинхронными вычислениями. Соответствующая модель потоков данных, описанная в первой части статьи, обеспечивает естественную автоматизацию и распределенное выполнение сложных итеративных вычислительных процессов, в том числе таких, в которых одна итерация внешнего процесса содержит несколько вложенных циклов выполнения. Основное внимание уделено вопросу организации обмена файлами между различными программными пакетами в распределенной вычислительной схеме с иерархической структурой. Предложенная методика работы с файлами учитывает общие требования к обработке данных в автоматизированных вычислениях: изолированность и возможность отслеживания происхождения данных, поддержку кэширования и повторного использования данных, поддержку параллельной обработки.

Ключевые слова: программирование потоков данных, распределенная файловая система, интеграция процессов, автоматизация вычислений, pSeven.

ВВЕДЕНИЕ

Решение актуальных научных и инженерных задач обычно предполагает использование целого набора специфического программного обеспечения. Действительно, в научных исследованиях активно используются программы, реализующие расчетные алгоритмы, численные методы, модели сложных систем; инженеры пользуются различными САПР и приложениями вычислительной гидродинамики. Большинство реальных задач требуют совместного применения многих приложений, а часто и сложной последовательности расчетов. Возможность автоматизировать такие расчеты и повторно использовать ранее полученные результаты является важным фактором, влияющим на производительность труда исследователей и инженеров в широком спектре предметных областей.

Распространенным подходом для достижения необходимого уровня автоматизации является использование одной из множества доступных научных и инженерных систем управления потоками работ [1–4, 8, 10–13]. Такие системы, как правило, обладают графическим пользовательским интерфейсом, где визуально задается описание процесса расчета (который часто представляется в виде графа), и предоставляют среду исполнения этих процессов. В таких случаях последовательность вычислений представляется в виде потока работ – набора взаимосвязанных задач с четко определенным порядком исполнения. Подобно языкам программирования, которые традиционно основываются на абстракциях, формальная семантика которых определяется моделью вычислений, модели потока работ предоставляют семантическую базу для систем управления потоками работ [5]. Различные модели потоков работ имеют различные сильные и слабые стороны. Очевидно, что в зависимости от специфики конкретной задачи, некоторые модели потока работ лучше отвечают требованиям со стороны пользователей – исследователей или инженеров.

В настоящей работе предлагается модель анализа и исполнения потока работ, разработанная для обеспечения естественной автоматизации и распределенного выполнения сложных итеративных вычислительных процессов (таких как моделирование методами Монте-Карло,

поиск по пространству параметров, оптимизация «черного ящика»), где последовательность расчетов включает ряд специализированных расчетных приложений, обменивающихся файлами в ходе выполнения расчетного процесса. (Подобные приложения, как правило, работают с локальными файлами и требуют наличия входного файла при запуске и сбора выходных файлов по завершении работы). Предлагаемая модель потока работ предназначена для широкого спектра задач, которые требуют многократного проведения одной и той же последовательности вычислений с различными наборами входных данных. Более того, предлагаемый подход учитывает сложные случаи, когда одна итерация процесса верхнего уровня может включать несколько вложенных циклов. Такая структура вычислений необходима, в частности, для проведения коллаборативной оптимизации [9], которая довольно часто применяется при исследовании пространства проектных параметров в инженерных приложениях. Также рассматриваются типичные требования к возможностям автоматизации процессов: изоляция исполнения, повторное использование и кэширование данных, параллельное исполнение, отслеживание происхождения данных.

Следует отметить, что представленная здесь модель потока работ является развитием модели, используемой в системе управления потоками работ pSeven, которая разрабатывается авторами в течение последних 5 лет. Предлагаемый подход к проблемам управления потоками работ основан на практическом опыте использования данной системы в решении реальных научных и инженерных задач на протяжении разработки.

МОДЕЛЬ ПОТОКА РАБОТ

Существуют два основных подхода к заданию потока работ – описание потока управления и описание потока данных. В первом случае явно задается порядок исполнения задач, а управление данными происходит в неявном виде. Во втором случае задаются только зависимости между данными, а порядок исполнения следует из зависимостей. Оба подхода имеют свои преимущества и используются в существующем программном обеспечении.

Предлагаемая модель потока работ основана на описании потоков данных. Основной причиной является присущая такой модели возможность параллелизации – задачи без зависимостей от данных всегда могут исполняться параллельно. Действительно, во многих практических приложениях производительность играет решающую роль, а с учетом современных направлений развития аппаратного обеспечения и архитектуры, параллельные вычисления являются ключом к эффективному использованию ресурсов. Также следует учитывать то, что схема потоков данных, вероятно, более проста для задания и понимания. При задании потока управления пользователю также необходимо представлять и то, как происходит обмен данными, чтобы быть уверенным, что каждая задача вовремя получит входные данные. С другой стороны, модель потока данных гарантирует, что исполнение задачи начинается только тогда, когда все зависимости по данным удовлетворены.

В последнее время проводилось много исследований в области развития парадигмы программирования потоков данных [6, 7]. Основные различия между ними в обработке циклов и условий. Итеративные вычисления и условные операторы, естественные в потоке управления, сложны для описания в терминах потоков данных. Многие системы, основанные на представлении потоков данных, просто исключают управляющие конструкции (по крайней мере частично) из модели потока работ, а итеративность реализуется с помощью внешних средств [8, 12]. У такого подхода есть существенный недостаток – поскольку управляющие конструкции не являются частью модели, их использование становится затруднительным, а иногда и невозможным. Другие системы вводят специальные задачи, управляющие порядком исполнения, и обрабатывают их специальным образом, отличая их от обычных задач [11]. Добавление специальных типов задач противоречит принципу разделения ответственности и усложняет модель.

Существуют также системы, позволяющие задачам выдавать выходные данные только на часть из полного набора выходов в зависимости от какого-либо условия и таким образом реализующие условные ветвления [2–4]. Данная возможность в сочетании с возможностью объединять задачи в замкнутый цикл позволяет создавать конечные циклы. Такие модели потока работ являются довольно гибкими, но при этом подвержены ошибкам, а также трудны для статического анализа, требующегося для выбора эффективной стратегии исполнения и обнаружения пользовательских ошибок.

Предлагаемая модель потока данных разработана с целью устранения вышеупомянутых недостатков. Она поддерживает произвольный уровень вложенности управляющих конструкций (вложенные циклы и условия), при этом сама модель довольно проста, что позволяет применять большинство известных методов анализа потоков данных, правила исполнения едины для всех задач, а ненужные задачи автоматически пропускаются при исполнении процесса.

В данной модели поток работ состоит из именованных задач. У каждой задачи есть наборы (возможно пустые) именованных входов и выходов, называемых портами. Данные передаются по связям – направленным соединениям типа «один к одному» между выходным и входным портами разных задач. Создание нескольких связей, ведущих к одному входному порту, не допускается, что исключает возможность возникновения состояния гонки. Для начала исполнения задачи требуется наличие данных на всех ее входах, а по завершении задача всегда выдает все выходы (это облегчает кэширование данных задачи). Циклические связи запрещены, чтобы избежать бесконечных циклов.

Все неподключенные входы и выходы считаются внешними портами самого потока работ. Для начала его выполнения требуется задать значения всех внешних входов (входная конфигурация) и указать список внешних выходов, значения которых требуется вычислить (требуемые выходы). Отметим, что не существует обязательного требования рассчитать значения всех выходов. Также в некоторых случаях поток работ может запускаться и без задания входов.

По сути, поток работ в предлагаемой модели представляется направленным ациклическим мультиграфом, где задачи являются узлами, а связи – ребрами графа. Такое представление упрощает анализ и исполнение потока работ.

Существуют две различные модели исполнения потока работ, называемые pull (по требованию) и push (по поступлению) [14]. Предлагаемая модель совмещает оба подхода: pull используется при предварительном анализе графа для поиска ненужных задач, а push – в ходе исполнения потока работ. Как сказано выше, для запуска потока работ необходимо задать входную конфигурацию и указать требуемые выходы. При анализе графа сначала в обратном направлении прослеживаются все пути от незатребованных выходов к внешним входам, и все задачи на этих путях отмечаются как ненужные. Затем так же прослеживаются все пути от требуемых выходов к внешним входам, и все задачи на этих путях отмечаются как необходимые. Таким образом, задачи, пройденные дважды, будут отмечены как необходимые. В результате все задачи будут отмечены либо как необходимые, либо как ненужные, либо останутся неотмеченными. На данном этапе уже можно провести валидацию того, что все задачи, отмеченные как необходимые, могут быть выполнены при данной входной конфигурации. При успешном прохождении валидации начинается исполнение задач, в ходе которого данные передаются по связям, причем ненужные задачи игнорируются. Исполнение останавливается, если не остается готовых к запуску задач. Отметим, что игнорирование ненужной задачи означает, что она не запускается даже в том случае, когда на ее входы поступил необходимый набор данных, – это позволяет сэкономить вычислительные ресурсы. Поскольку прохождение валидации означает, что все необходимые задачи способны запуститься, то гарантируется и возможность вычисления всех требуемых выходов потока работ.

Неотмеченные задачи также могут быть запущены, несмотря на то, что в рамках такой модели исполнения они по существу являются бесполезными (поскольку не генерируют каких-

либо данных, необходимых для вычисления требуемых выходов). Рассматриваемый алгоритм запускает исполнение тех неотмеченных задач, которые зависят от необходимых задач, и игнорирует неотмеченные задачи, которые зависят только от ненужных. Данное правило обосновывается тем, что на практике такие задачи, как правило, появляются на участках, где происходит экспорт данных из потока работ, – например, для журналирования значений.

Очевидно, активация ветвей исполнения зависит от требуемых выходов, и если таковые отсутствуют, то не будет запущено ни одной задачи. На рис. 1 показан пример результатов анализа потока работ. Затребован только выход *a*, поэтому задача *A* является необходимой, а задача *B* – ненужной. Задачи *C* и *D* остаются неотмеченными, однако *C* будет запущена, поскольку она зависит от необходимой задачи *A*, а *D* не запустится, поскольку зависит от ненужной задачи *B*.

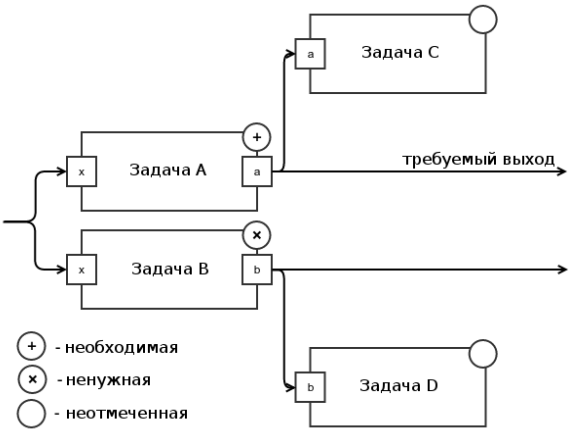

Рис. 1. Зависимые задачи

В описанной выше модели могут быть отражены и более сложные зависимости, однако она все же не способна на представление итеративных процессов и условий.

Для решения данной проблемы в обсуждаемую модель введен такой элемент потока работ, как составная задача. Внешне составная задача ничем не отличается от обычной – она также имеет входы, выходы и подчиняется описанным выше правилам запуска. Отличие в ее внутреннем устройстве: составная задача имеет отдельный набор внутренних выходов и входов, которые соединяются с элементами внутреннего потока работ. Не существует какого-либо исходно заданного соответствия между внутренними и внешними портами: задание этого соответствия является основной ролью составной задачи. Составные задачи могут содержать другие составные задачи, что позволяет создавать иерархические процессы произвольного уровня вложенности. Наличие иерархии полезно, даже если не рассматривать создание циклов и условий, поскольку она дает возможность удобно структурировать описание потока работ.

Запустившись, составная задача может исполнять свой внутренний поток работ несколько раз, возможно в параллельном режиме. На каждой внутренней итерации составная задача устанавливает новую входную конфигурацию и набор требуемых выходов для вложенного потока работ, используя свои внутренние порты.

Рассмотрим функцию *map*, которая применяет другую заданную функцию к каждому элементу входного массива, возвращая массив результатов. Эта функция имеет довольно широкое применение, в частности, может быть использована вместо простого (не имеющего изменяемого внутреннего состояния) цикла, выполнение которого не зависит от результатов предыдущих итераций. Соответствующая задача *Map* (рис. 2) принимает список значений на внешний вход *x_list*, поочередно передает каждый элемент принятого списка внутреннему процессу через внутренний выход *x*, аккумулирует результаты, получаемые на внутренний вход *f*, и по

окончании работы выдает список результатов во внешний выход *f_list*. Обработка каждого элемента происходит независимо, что позволяет проводить параллельную обработку массива произвольного размера, т. е. дает возможность масштабирования.

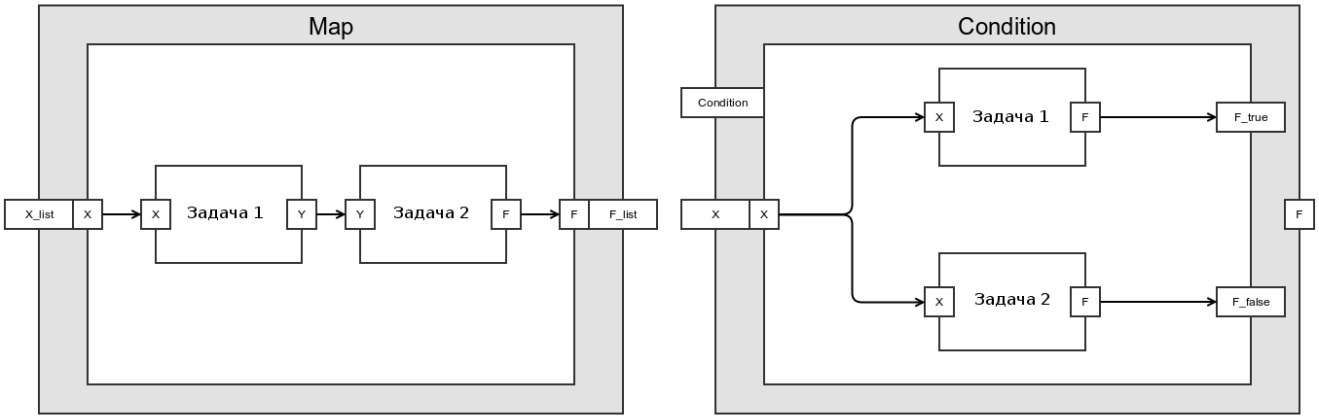

Рис. 2. Составные задачи *Map* и *Condition*

Таким образом, любой итеративный процесс, не обладающий внутренним состоянием, может быть представлен в виде составной задачи. Итераторы с внутренним состоянием также реализуемы, поскольку составная задача может определять и логику состояний в дополнение к логике соответствия внешних и внутренних портов. Кроме того, составные задачи могут реализовывать такие условные операторы, как *if...else* и *switch*, запрашивая вычисление значений на внутренних входах в различных комбинациях. Задача *Condition* на рис. 2 определяет логику условной обработки входа *x*. Вход *Condition* принимает логическое значение (выражение); в зависимости от значения (результата вычисления выражения) задача требует вычислить значение либо на внутреннем входе *F_true*, либо на *F_false*. Далее происходит исполнение внутренних подзадач согласно общим правилам модели (рис. 1).

Важно, что на каждом уровне иерархии сохраняются единые базовые свойства модели, что позволяет применять известные методы статического анализа и управления выполнением задач.

УПРАВЛЕНИЕ ФАЙЛАМИ

Очевидно, что использование файлов для передачи данных в реальных задачах неизбежно. В частности, это необходимо в случае наличия задач, которые обеспечивают взаимодействие с внешними программами (хорошим примером являются различные САПР). Некорректное использование файлов может привести к нарушению работы и неопределенному поведению (включая возникновение условия гонки и потерю детерминированности): поскольку файловая система может работать как общая память, возможно создать между задачами неявные зависимости, которые не могут быть обнаружены при статическом анализе потока работ. Данная проблема важна, но не является фундаментальной, поскольку всегда существует способ избежать создания таких неявных связей. В дальнейшем обсуждении мы предполагаем, что файловая система используется корректно: задачи изолированы, никакая задача не модифицирует файлы, принадлежащие другой задаче, и явно определены зависимости, в силу которых какой-либо задаче необходим доступ к чтению файлов другой задачи.

К управлению файлами в потоке работ предъявляются определенные требования. В первую очередь необходимо принять во внимание распределенное выполнение задач, что

требует возможности передавать файлы между задачами. Во-вторых, поскольку задачи могут исполняться параллельно, следует обратить внимание на возможность возникновения условия гонки или конфликта имен. Наконец, в потоке работ часто обрабатываются большие объемы данных, что требует определенного уровня производительности. Сравним два распространенных подхода к обмену файлами, учитывая данные требования.

Одной из возможностей является управление файлами с использованием сетевой файловой системы общего назначения, в которой доступ к некоему каталогу разрешен всем узлам сети, на которых происходит распределенное выполнение потока работ. В таком случае обмен файлов может быть упрощен до обмена файловыми путями. Преимуществом такого подхода является то, что подсистема управления исполнением может вообще не учитывать управление файлами при условии корректного поведения всех задач. С другой стороны, эта же особенность является и недостатком, так как наличие конфликтов обработки данных зависит от исполняемого потока работ. Кроме того, распределенные файловые системы сложны в настройке, платформозависимы и могут не обеспечивать необходимый уровень безопасности. Что касается производительности при таком подходе, то она существенно ограничена при его применении где-либо за исключением вычислительных кластеров.

Другой подход предполагает прямую пересылку содержимого файлов через порты и создание отдельного порта для каждого файла. В отличие от описанного выше такой подход не требует дополнительной настройки и при определенных свойствах подсистемы управления исполнением может быть оптимизирован для обеспечения необходимой производительности. Стоит также отметить, что в модели потоков данных единые правила обработки файлов и других данных автоматически исключают возможность возникновения условия гонки. К сожалению, существенные ограничения налагает требование создания отдельного порта для каждого файла. Для примера рассмотрим задачу, имеющую несколько независимых файловых выходов, количество которых не известно заранее. В этом случае невозможно узнать, какое количество выходных портов следует создать. Иногда возможным решением является архивация всего каталога с выходными файлами и пересылка полученного архива через порт. Однако архивация требует дополнительных ресурсов как при создании, так и при открытии архива – что, помимо прочего, бессмысленно в случае, когда передающая и принимающая задачи запускаются на одном и том же сетевом узле. Кроме того, один или несколько выходных файлов часто являются «главными» и явно или неявно ссылаются на остальные файлы; вследствие этого задача, генерирующая комплект выходных файлов, должна не только передать его следующей задаче, но и каким-то образом сообщить, какие из файлов являются главными. Например, многие итеративные решатели генерируют файлы с результатами каждой итерации, а количество итераций может быть неизвестно. При этом решатель также создает файл, содержащий ссылки на все файлы с результатами итераций (т. е. этот файл является главным). При рассматриваемом подходе задача, запускающая такой решатель, должна иметь как минимум два выхода: один для передачи архива со всеми выходными файлами и второй для передачи какого-либо указателя на главный файл. В конечном счете упомянутые аспекты делают передачу файлов через порты неэффективной и часто неудобной в реальных задачах.

Первый подход чаще используется в высокопроизводительных кластерах, а второй популярен в существующих системах управления потоками работ. Тем не менее у обоих подходов есть серьезные ограничения. Чтобы преодолеть их, мы предлагаем собственный подход к управлению файлами, тесно связанный с моделью потоков данных, обсуждавшейся в предыдущей части.

Условимся, что задача может создавать выходные файлы только в своей собственной «песочнице». «Песочница» – это просто локальный каталог, поэтому никаких специальных методов доступа не требуется. «Песочницы» подчиняются структуре, основанной на иерархии потока работ (рис. 3) и обеспечивают некоторый уровень изоляции задач.

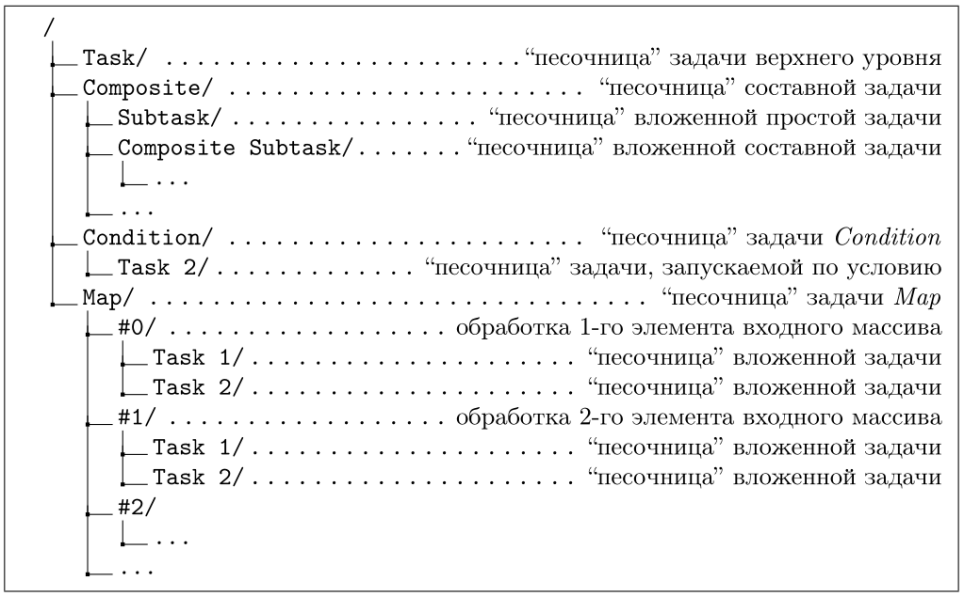

Рис. 3. Структура каталогов «песочницы»

Задача может читать файлы в «песочницах» других задач. В случае распределенного выполнения структура «песочниц» копируется на каждый узел, однако все «песочницы» за исключением тех, которые требуются для исполнения локальной задачи, остаются пустыми. Например, когда задача *B* получает файл от задачи *A*, подсистема исполнения разрешает задаче *B* прочитать определенные файлы из «песочницы» задачи *A*, и эти файлы передаются на узел, исполняющий задачу *B*, где она и получает к ним доступ. Очевидно, если обе задачи исполняются на одном узле, никакого копирования файлов не происходит.

В зависимости от конфигурации сети файлы могут передаваться как напрямую между вычислительными узлами, так и через некий центральный узел. Однако по завершении исполнения потока работ все «песочницы» синхронизируются с главным узлом с целью сохранения истории вычислений. Это требование связано с тем, что файлы, созданные в процессе исполнения, часто нужны для дальнейшего анализа, а также с тем, что система управления потоком работ должна давать возможность прослеживать ход вычислений. Прослеживание вычислений в данном случае означает поддержание ясного соответствия между выходными файлами и наборами входных данных для каждого запуска каждой задачи. В случае отсутствия многоуровневой иерархии и компонентов, управляющих ходом исполнения, установить данное соответствие просто ввиду того, что каждая задача исполняется только один раз. Однако появление циклов в модели потока работ требует введения дополнительных правил хранения данных в «песочнице» для того, чтобы устанавливать соответствие входных и выходных данных.

С этой целью предлагается многоуровневая структура, представленная на рис. 3. Вернемся к рассмотренной ранее составной задаче *Map* – она может обрабатывать элементы входного массива параллельно и в произвольном порядке. На рис. 3 показана структура каталогов «песочницы» данной задачи, в которой каждый подкаталог соответствует конкретной итерации. В качестве имени такого подкаталога естественно использовать индекс элемента входного массива. Однако подсистема исполнения сама по себе не способна связать запуск внутреннего потока работ задачи *Map* с каким-либо конкретным элементом (индексом), поскольку она не рассматривает дополнительную логику, реализуемую составной задачей. (Как указано ранее, в рамках предлагаемой модели все задачи эквивалентны и обрабатываются по единым правилам.). Чтобы обеспечить корректное именование подкаталогов «песочницы», составная задача должна сама передать имя итерации (вместе с входной конфигурацией и списком требуемых выходов) внутреннему потоку работ, чтобы сделать это имя доступным подсистеме исполнения.

ИНЖЕНЕРНАЯ ОПТИМИЗАЦИЯ

С целью проиллюстрировать практические преимущества предлагаемой модели потока работ рассмотрим типичную инженерную задачу – оптимизацию конструкции изделия. На верхнем уровне абстракции цель оптимизации состоит в том, чтобы найти такие значения параметров модели конструкции, при которых значение некоторой ее характеристики (целевой функции) минимально, причем значения параметров удовлетворяют ограничениям, как правило, заданным в виде функциональных неравенств. На практике существует много различных формулировок задачи оптимизации, здесь для краткости мы остановимся на одной из простейших.

Предположим, что каждое вычисление целевой функции требует последовательного запуска двух приложений, а для вычислений ограничений используется только одно из них. Действительно, для вычисления ограничений часто достаточно рассчитать массовые характеристики с помощью какой-либо CAD-системы, работающей с моделью геометрии изделия, тогда как для вычисления целевой функции необходимо передать модифицированную модель CAE-системе.

В терминах предлагаемой модели данный процесс состоит из трех задач: составной задачи *Optimizer*, реализующей оптимизационный алгоритм и управляющей ходом исполнения, задачи *CAD*, которая принимает значения параметров, генерирует файл с новой геометрией модели и рассчитывает значения ограничений, и задачи *CAE*, которая принимает геометрическую модель и рассчитывает значение целевой функции (рис. 4).

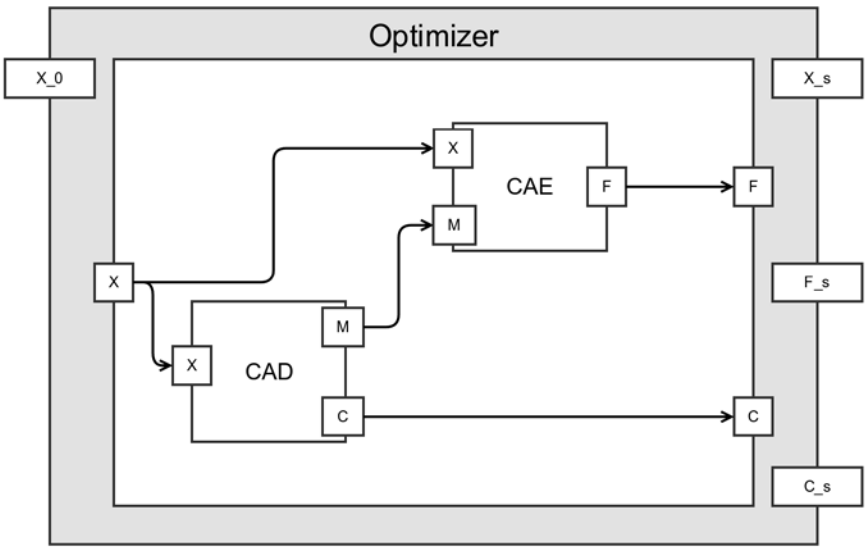

Рис. 4. Составная задача оптимизации *Optimizer*

Здесь *Optimizer* – составная задача со вложенными задачами *CAD* и *CAE*. В начале исполнения задача *Optimizer* ожидает поступления начальных значений параметров на порт *X_0*. При получении этих значений начинается итеративный процесс – оптимизационный алгоритм генерирует значения параметров конструкции *X* и запрашивает значение или целевой функции *F*, или ограничений *C*, или всех функций задачи. Когда оптимальное решение найдено, задача *Optimizer* выводит его в порты *X_s*, *F_s* и *C_s*. Отметим, что важное отличие задачи оптимизации от простого цикла (такого, который может быть заменен задачей типа *Map*) в том, что внутренние итерации оптимизатора зависят от результатов предыдущих итераций, т.е. данная задача имеет изменяемое внутреннее состояние.

Когда задача *Optimizer* запрашивает только значения ограничений *C*, задача *CAE* не запускается, поскольку она не нужна для расчета значений выходов. С другой стороны, когда за-

прашивается значение целевой функции *F*, всегда запускаются обе задачи, *CAD* и *CAE*, и задача *Optimizer* получает и значения ограничений, хотя и не запрашивала их. Это происходит из-за того, что значения ограничений необходимы для вычисления целевой функции. Будут ли эти значения использованы, зависит уже от логики оптимизационного алгоритма.

Рассмотрим теперь управление файлами в оптимизационном цикле. Как говорилось ранее, возможность просмотра файлов, созданных в ходе исполнения потока работ, в реальных задачах необходима. В данном примере требуется идентифицировать выходные файлы задач *CAD* и *CAE*, соответствующие конкретному набору конструкторских параметров.

Задача *Optimizer* использует возможность указывать имена итераций внутреннего потока работ для того, чтобы организовывать «песочницы» своих подзадач естественным образом, группируя их в соответствии с наборами входных параметров (рис. 5)[1].

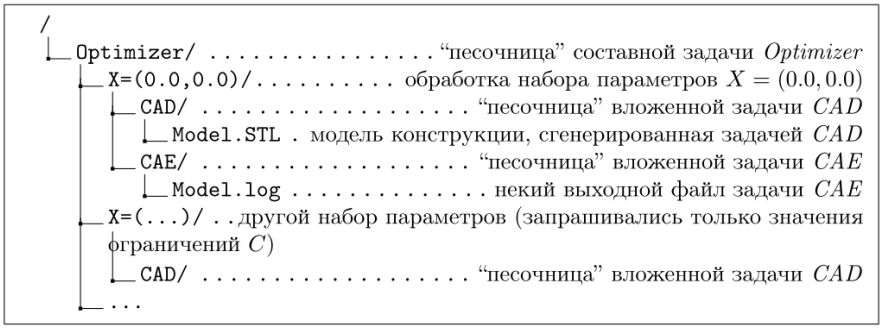

Рис. 5. Структура каталогов «песочницы» задачи *Optimizer*

Данная структура не обязательно соответствует порядку выполнения вычислений. Например, для некоторой точки в пространстве параметров сначала могут быть вычислены значения ограничений, а на другой итерации – значения целевой функции. Несмотря на то что вычисления происходили на разных итерациях, результаты будут сохранены в один и тот же подкаталог, ассоциированный с данным набором параметров. Как следствие, это делает возможным довольно прямолинейную реализацию кэширования входных и выходных данных с целью экономии вычислительных ресурсов. Например, если оптимизационный алгоритм запрашивает значение целевой функции в точке, где значения ограничений уже вычислялись, имеет смысл не запускать задачу *CAD* заново, поскольку ранее уже получены результаты ее исполнения с таким набором входных значений. Для этого нужно сохранить выходные файлы задачи (это уже предусмотрено, файлы сохраняются в «песочнице» данной задачи) и соответствующие значения входных и выходных портов – эти значения могут быть сохранены в специальный файл, кэширующий данные задачи. Важно, что кэш должен хранить только значения портов, что дает возможность реализовать его без существенных дополнительных затрат вычислительных ресурсов.

В заключение, по завершении оптимизации «песочница» задачи *Optimizer* будет содержать все созданные подзадачами файлы и данные вычислений, организованные в естественную и удобную для использования структуру.

ЗАКЛЮЧЕНИЕ

Вычисления, основанные на модели потока работ, являются полезным инструментом для специалистов различных предметных областей, позволяющим автоматизировать часть исследовательских процессов и при этом не требующим навыков программирования. Однако в различ-

[1] В случае когда значения параметров не могут быть указаны в имени подкаталога, как на рис. 5, задача оптимизации может создать дополнительный файл, содержащий наборы значений параметров и соответствующие им имена подкаталогов.

ных моделях потоков работ наблюдается ряд давно существующих проблем, которые в некоторых случаях усложняют или даже делают невозможным их применение. В рамках модели потоков данных наибольшие затруднения связаны с реализацией итеративных процессов и условными ветвлениями, в особенности при наличии многих уровней вложенности.

Отдельным вопросом является управление файлами. Распределенное выполнение делает ручное управление файлами неприменимым даже для опытных пользователей, следовательно, эту роль должна взять на себя система управления потоком работ.

Как решение предложена модель потока работ, учитывающая необходимость управления файлами. Данная модель успешно применялась для описания ряда сложных инженерных задач и в настоящее время реализуется в рамках проекта по созданию интеграционной платформы для применения распределенных и облачных вычислений при решении научных и инженерных задач.

Благодарности. Работа выполнена при финансовой поддержке РФФИ в рамках научного проекта № 15-29-07043.

Условия для выполнения работ по проекту предоставлены ООО «ДАТАДВАНС».

Модельные эксперименты проводились с использованием ресурсов КЦОД НИЦ «Курчатовский институт» (http://computing.kiae.ru).

СПИСОК ЛИТЕРАТУРЫ

1. Afanasiev A., Sukhoroslov O., Voloshinov V. MathCloud: Publication and Reuse of Scientific Applications as RESTful Web Services. In 12th International Conference "Parallel Computing Technologies"(PaCT 2013), vol. 1662, pp. 394–408. Springer, 2013.

2. Altintas I., Berkley C., Jaeger E., Jones M., Ludascher B., Mock S. Kepler: an extensible system for design and execution of scientific workflows. Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004, August 2015, pp. 423–424.

3. Автоматизация инженерных расчетов, анализ данных и оптимизация с помощью программного комплекса PSE/MACROS. / Е.В. Бурнаев, Ф.В. Губарев, С.М. Морозов, А.А. Прохоров, Д.С. Хоминич // Межотраслевая информационная служба. 2013. № 4 (165). С. 41–50.

4. Churches D., Gombas G., Harrison A., Maassen J., Robinson C., Shields M., Taylor I., Wang I. Programming scientific and distributed workflow with Triana services. Concurrency Computation Practice and Experience, 2006. Vol. 18, No. 10, pp. 1021–1037.

5. Elmroth E., Hernández F., Tordsson J. Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment. Future Generation Computer Systems, February 2010, Vol. 26, No 2, pp. 245–256.

6. Johnson G. LabVIEW graphical programming. Tata McGraw-Hill Education, 1997.

7. Johnston W., Hanna J., Millar R. Advances in dataflow programming languages. ACM Computing Surveys, March 2004, Vol. 26, No 1, pp. 1–34.

8. Malone B., Papay M. Modelcenter: an integration environment for simulation based design. In Simulation Interoperability Workshop, 1999.

9. Martins J., Lambe A. Multidisciplinary Design Optimization: A Survey of Architectures. AIAA Journal, 2013, Vol. 51, No 9, pp. 2049–2075.

10. Seider D., Fischer P., Litz M., Schreiber A., Gerndt A. Open Source Software Framework for Applications in Aeronautics and Space. In 2012 IEEE Aerospace Conference, 2012, pp. 1–11.

11. Wikipedia. Modefrontier – wikipedia, the free encyclopedia, 2015.

12. Wikipedia. Optimus platform – wikipedia, the free encyclopedia, 2015.

13. Wolstencroft K., Haines R., Fellows D., Williams A., Withers D., Owen S., Soiland-Reyes S., Dunlop I., Nenadic A., Fisher P., Bhagat J., Belhajjame K., Bacall F., Hardisty A.,

de la A. Hidalga, Vargas M., Sufi S., Goble C. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. Nucleic acids research, 41, 2013.

14. Zhao Y. A Model of Computation with Push and Pull Processing. Master's report, technical memorandum ucb/erl m03/51, University of California at Berkeley, 2003.

СВЕДЕНИЯ ОБ АВТОРАХ

Назаренко Алексей Михайлович, старший программист, ООО «ДАТАДВАНС», электронный адрес: alexey.nazarenko@datadvance.net.

Пересторонин Никита Олегович, старший программист, ООО «ДАТАДВАНС», электронный адрес: nikita.perestoronin@datadvance.net.

Прохоров Александр Александрович, руководитель отдела разработки ПО, ООО «ДАТАДВАНС», научный сотрудник, Институт проблем передачи информации им. А.А. Харкевича РАН, электронный адрес: alexander.prokhorov@datadvance.net.

HIERARCHICAL DATAFLOW MODEL WITH AUTOMATED FILE MANAGEMENT FOR ENGINEERING AND SCIENTIFIC APPLICATIONS

Aleksey M. Nazarenko
DATADVANCE, Moscow, Russia, alexey.nazarenko@datadvance.net
Nikita O. Perestoronin
DATADVANCE, Moscow, Russia, nikita.perestoronin@datadvance.net
Aleksandr A. Prokhorov
DATADVANCE, Moscow, Russia
Institute for information transmission problems RAS (Kharkevich Institute), Moscow, Russia
alexander.prokhorov@datadvance.net

ABSTRACT

Solving modern scientific and engineering problems typically implies using multiple task-specific software applications and often a complex sequence of computations must be performed. Adopted approach to achieve the required level of automation is to use one of the many available scientific and engineering workflow systems, which can be based on different workflow models. This paper introduces a workflow model targeted to provide natural automation and distributed execution of complex iterative computation processes, where the calculation chain contains multiple task-specific software applications which exchange files during the process.

The proposed workflow model addresses a wide range of applications and targets complex cases when a single iteration of a top-level process may contain multiple nested execution loops. Typical requirements to process automation are considered as well: execution isolation, data re-use and caching, parallel execution, data provenance tracking.

Key words: automation of scientific processes, automation of engineering processes, distributed computing, workflow, dataflow, pSeven.

REFERENCES

1. Afanasiev A., Sukhoroslov O., Voloshinov V. MathCloud: Publication and Reuse of Scientific Applications as RESTful Web Services. In 12th International Conference "Parallel Computing Technologies" (PaCT 2013), 2013, vol. 1662, pp. 394–408. Springer.

2. Altintas I., Berkley C., Jaeger E., Jones M., Ludascher B., Mock S. Kepler: an extensible system for design and execution of scientific workflows. Proceedings. 16th International Conference on Scientific and Statistical Database Management, 2004, (August 2015):423–424, 2004.

3. Burnaev E., Gubarev F., Morozov S., Prokhorov A., Khominich D. PSE/MACROS: Software Environment for Process Integration, Data Mining and Design Optimization. *The internet representation of scientific editions of FSUE "VIMI" (The All-Russian Research Institute for Interindustry Information - a Federal Informational and Analytical Center of the Defense Industry, a Federal State Unitary Enterprise)*, (4):41–50, 2013.

4. Churches D., Gombas G., Harrison A., Maassen J., Robinson C., Shields M., Taylor I., Wang I. Programming scientific and distributed workflow with Triana services. Concurrency Computation Practice and Experience, 18(10):1021–1037, 2006.

5. Elmroth E., Hernández F., Tordsson J. Three fundamental dimensions of scientific workflow interoperability: Model of computation, language, and execution environment. Future Generation Computer Systems, 26(2):245–256, February 2010.

6. Johnson G. LabVIEW graphical programming. Tata McGraw-Hill Education, 1997.

7. Johnston W., Hanna J., Millar R. Advances in dataflow programming languages. ACM Computing Surveys, 36(1):1–34, March 2004.

8. Malone B., Papay M. Modelcenter: an integration environment for simulation based design. In Simulation Interoperability Workshop, 1999.

9. Martins J., Lambe A. Multidisciplinary Design Optimization: A Survey of Architectures. AIAA Journal, 51(9):2049–2075, 2013.

10. Seider D., Fischer P.M., Litz M., Schreiber A., Gerndt A. Open Source Software Framework for Applications in Aeronautics and Space. In 2012 IEEE Aerospace Conference, pages 1–11, 2012.

11. Wikipedia. Modefrontier — wikipedia, the free encyclopedia, 2015. [Online; accessed 15 August 2015].

12. Wikipedia. Optimus platform — wikipedia, the free encyclopedia, 2015. [Online; accessed 15 August 2015].

13. Wolstencroft K., Haines R., Fellows D., Williams A., Withers D., Owen S., Soiland-Reyes S., Dunlop I., Nenadic A., Fisher P., Bhagat J., Belhajjame K., Bacall F., Hardisty A., de la Hidalga A., Vargas M., Sufi S., Goble C. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. Nucleic acids research, 41(Web Server issue), 2013.

14. Zhao Y. A Model of Computation with Push and Pull Processing. Master's report, technical memorandum ucb/erl m03/51, University of California at Berkeley, 2003.