УДК 004.4'2, 004.7

В.Д. Павленко, канд. техн. наук,
В.В. Бурдейный

ТЕХНОЛОГИЯ ПРОГРАММИРОВАНИЯ ВЫЧИСЛЕНИЙ В КЛАСТЕРНЫХ СИСТЕМАХ С ИСПОЛЬЗОВАНИЕМ ТРАНСПАРЕНТНОГО РАСПАРАЛЛЕЛИВАНИЯ, ОСНОВАННОГО НА ЗАКАЗАХ

Рассматриваются реализационные основы создания прикладного инструментария для поддержки технологии транспарентного распараллеливания вычислений в кластерных системах на основе заказов. Описываются основные архитектурные решения, принятые при разработке и создании инструментария параллельных вычислений.

Ключевые слова: *параллельные вычисления, кластеры, инструментарии, архитектура программного обеспечения.*

V.D. Pavlenko, PhD,
V.V. Burdeynyy

TECHNOLOGY OF PROGRAMMING OF COMPUTING IN THE CLUSTER SYSTEMS WITH USING TRANSPARENT PARALLELIZING BASED ON ORDERS

Realization fundamentals for creation of the applied tools are examined for support of technologies orders based transparent parallelizing of cluster computing systems. Basic architectural decisions, accepted at development and creation of tool of parallel computations, are described.

Keywords: *parallel computing, claster, frameworks, software architecture.*

В.Д. Павленко, канд. техн. наук,
В.В. Бурдейний

ТЕХНОЛОГІЯ ПРОГРАМУВАННЯ ОБЧИСЛЕНЬ У КЛАСТЕРНИХ СИСТЕМАХ З ВИКОРИСТАННЯМ ТРАНСПАРЕНТНОГО РОЗПАРАЛЕЛЮВАННЯ, ЩО ЗАСНОВАНЕ НА ЗАМОВЛЕННЯХ

Розглядаються реалізаційні основи створення прикладного інструментарію для підтримки технології транспарентного розпаралелювання обчислень у кластерных системах на основі замовлень. Описуються основні архітектурні рішення, які прийняті при розробці та створенні інструментарію паралельних обчислень.

Ключові слова*: паралельні обчислення, кластери, інструментарії, архітектура програмного забезпечення.*

Введение. Параллельные вычисления активно развиваются в последнее время [3, 6, 7, 10], что обусловлено в первую очередь появлением все более сложных задач, решение которых на современных последовательных ЭВМ за приемлемое время получить невозможно [4, 8]. В частности, одной из важных задач в области параллельных вычислений является создание средств разработки параллельных приложений, позволяющих быстро создавать эффективные параллельные программы и распараллеливать существующие последовательные программы, а также предоставлять пользователю средства поддержки разработки, например, анализа трасс выполнения, отладки параллельных приложений или нахождения узких мест.

Для решения указанной задачи используются два основных подхода. Первый заключается в предоставлении прикладным программистам средств ручного распараллеливания, как правило, низкоуровневых средств организации потоков, синхронизации или коммуникации между вычислительными узлами. Преимущества такого метода – в возможности создания средств, основанных на широко используемых императивных языках программирования (и потому возможности использования существующих реализаций алгоритмов) и широким возможностям для ручной оптимизации, а недостатки – в сложности тестирования, отладки и поддержки параллельных приложений.

Другой подход заключается в создании полуавтоматических и автоматических средств распараллеливания. К таким средствам, в частности, относится технология транспарентного распараллеливания [5, 9].

Целью данной работы является реализация технологии транспарентного распараллеливания в виде прикладного инструментария параллельных вычислений; решение задач, связанных с реализацией и практиче-

ским использованием предлагаемого инструментария, обоснование основных архитектурных решений, принятых при создании инструментария.

Технология транспарентного распараллеливания. Основная идея технологии транспарентного распараллеливания заключается в выделении наборов алгоритмов, которые могут быть на высоком уровне описаны одной структурой и могут быть эффективно распараллелены сходным образом и реализации шаблонов параллельных алгоритмов в терминах таких структуры, после чего разработанные шаблоны могут быть использованы для распараллеливания конкретных прикладных алгоритмов.

Преимуществом такого подхода является возможность реализации достаточно сложных средств поддержки разработки и выполнения параллельных программ в рамках зафиксированных структур, которая достигается путем сужения класса эффективно распараллеливаемых алгоритмов. Впрочем, при появлении алгоритма, не распараллеливаемого эффективно в рамках ни одной из уже реализованных структур, для него может быть реализована новая структура.

На данный момент в рамках технологии транспарентного распараллеливания была рассмотрена и реализована одна структура алгоритма, основанная на предположении, что в распараллеливаемой программе выделен набор процедур, в процессе работы которых не модифицируются никакие переменные, кроме параметров и данных, недоступных вне данного вызова процедуры, а входные и выходные параметры передаются по значению. Выполнение программы должно сводиться к выполнению одной из этих процедур.

Первый принцип, на котором основан предлагаемый метод распараллеливания, вводит понятие заказа на вычисления. Заказ – единица выполняемой на одном из компьютеров кластера работы, т.е. объем работы, который следует выполнить на одном из компьютеров кластера полностью и не может быть разбит на более мелкие части. В качестве такого объема работы (заказа) вводится выполнение одной процедуры без выполнения тех процедур, которые она вызывает, а выполнение каждой из них выделяется в отдельный заказ. Чтобы определить точку старта программы, одну из процедур необходимо описать как главную. Через параметры этой процедуры должны передаваться исходные данные и результаты вычислений.

Ускорение достигается благодаря второму принципу, основанному на существовании отрезков времени между получением некоторых данных и первым их использованием в дальнейших вычислениях, которые часто могут быть специально увеличены путем изменения порядка вычислений.

При традиционном подходе при вызовах процедур вызывающая процедура продолжает свою работу только после окончания работы вызываемой процедуры. Другими словами, вызывающая процедура начинает ожидание результатов выполнения вызываемой процедуры в момент вызова, а заканчивает в момент завершения вызываемой процедуры. Вместо этого предлагается начинать ожидание в момент, когда данные, вычисляемые вызываемой процедурой, потребуются для дальнейшей работы (если в этот момент они уже вычислены, то начинать ожидание не нужно), и заканчивать ожидание в момент, когда эти данные получены.

Программная реализация технологии. Предлагаемый подход предполагает использование параллелизма заданий и модели *MIMD* (*Multiple Instruction Multiple Data*) [1]. Взаимодействие между компьютерами при этом сводится к следующим пяти операциям: получению заказа, его отправке, запросам о наличии значений данных, получению вычисленных в другом заказе данных и отправке результатов выполнения заказа.

Прикладному программисту из этих пяти операций доступны только две: отправка заказа и получение вычисленных в другом заказе данных. Важно, что детали передачи данных между компьютерами несущественны для программиста, а потому могут быть инкапсулированы в инструментарии, реализующем предлагаемую технологию.

В рамках данной технологии выполняемая программа является набором инструкций, описывающих работу кластера в целом, в то время как традиционно использующаяся для написания приложений для кластеров

технология MPI [2] основана на том, что программа – набор инструкций для каждого из компьютеров кластера в отдельности. Такой подход, когда программа создается для кластера так, как если бы она полностью выполнялась на однопроцессорном компьютере, во многих случаях предпочтительна для разработчика прикладных программ. Областью применения предложенной технологии распараллеливания являются вычислительные алгоритмы, реализуемые с использованием параллелизма заданий.

Предложенный подход к распараллеливанию вычислений может быть реализован на достаточно большом количестве языков программирования. В данной работе используется язык программирования *Java*, поэтому дальнейшие рассуждения будут производиться в терминах этого языка программирования.

Так как для распараллеливаемой программы было введено требование о том, что в программе выделены процедуры, параметры которых передаются по значению, а в процессе работы какие-либо изменения вносятся только в параметры и временно создаваемые и недоступные извне структуры данных, можно сделать следующие выводы.

Во-первых, передача данных из одной процедуры в другую возможна только через параметры, причем, поскольку параметры передаются по значению, взаимодействие происходит только в два момента времени – в момент вызова и в момент завершения работы вызываемой процедуры. Этим гарантируется, что процедуры во время выполнения не будут работать с какими-либо данными, доступными извне. Таким образом обеспечивается, с одной стороны, возможность переноса выполнения отдельной процедуры на другой компьютер кластера, а с другой – возможность одновременного выполнения нескольких процедур на одном компьютере, что позволяет после приостановки выполнения какой-либо процедуры использовать освободившиеся вычислительные ресурсы.

Во-вторых, выполнение всякой процедуры сводится к внесению некоторых изменений в переданные значения параметров, а поэтому всякий вызов процедуры можно заменить внесением в переданные значения параметров тех изменений, которые бы выполнила эта процедура в процессе своей работы. Такие изменения могут быть внесены или в момент вызова, или после него, но не позднее первого обращения к значению этого параметра.

Приведенными соображениями обеспечивается корректность предложенной технологии, понимаемая как совпадение результатов выполнения алгоритмов при отсутствии и при наличии распараллеливания, а также как возможность увеличения скорости выполнения за счет распараллеливания вычислений.

В терминах языка *Java* введенные предположения о пользовательском коде означают, что в программе должен быть выделен набор статических методов. Передача параметров по значению вызывает трудности во многих языках программирования. В частности, в *Java* по значению передаются лишь входные параметры примитивных типов, а организовать передачу по значению можно только для одного выходного параметра примитивного типа, передавая его через возвращаемое значение метода. Поэтому заменим это требование другим, позволяющим сделать те же выводы относительно корректности технологии. Потребуем, чтобы результат и время выполнения процедуры не изменились, если значение каждого из входных параметров не примитивного типа было заменено результатом десериализации сериализованного значения данного параметра, а все значения выходных параметров были заменены значениями по умолчанию.

Средства для отправки заказов могут быть организованы путем генерации набора методов, по сигнатурам совпадающих с выделенными методами, и отправляющих соответствующие заказы на сервер. Методы при этом могут выделяться в исходном коде программы либо при помощи специальных комментариев, либо аннотированием выделенных методов, либо аннотированием методов абстрактного класса именами выделенных методов (заказы, на выполнение которых будет отправлять генерируемый потомок этого класса), либо декларированием методов во внешнем файле. Классы с методами для отправки заказов могут быть сгенерированы либо в виде исходного кода, или непосредственно в виде байт-кода.

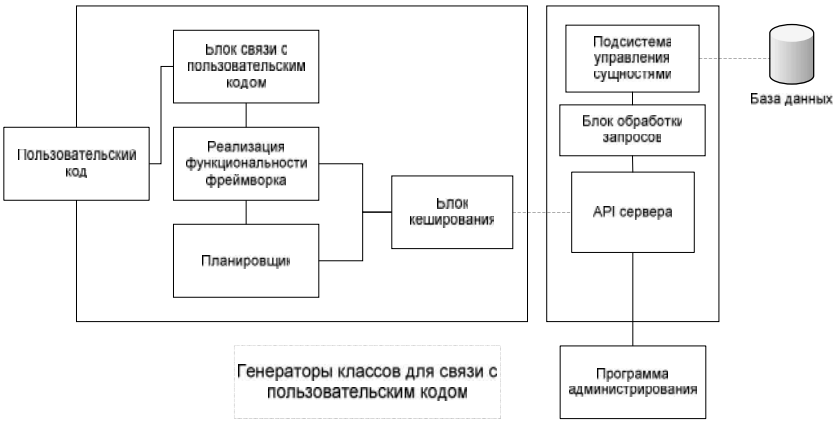

Рис. 1. Архитектура разработанного инструментария

Описанная технология реализована в виде прикладного инструментария параллельных вычислений. Его архитектура приведена на рис. 1.

При реализации инструментария была выделена серверная часть, на которую возложены функции как по координации действий клиентов, так и по хранению промежуточных данных и предоставлению их клиентам. Такое решение принято для упрощения обеспечения надежности, поддержки динамического изменения состава кластера и управления кластером с единого узла.

Дополнительным средством обеспечения надежности является внешняя база данных, которая может использоваться для хранения промежуточных данных вычислений. Это позволяет продолжать вычисления с относительно небольшими потерями времени после аппаратного или программного сбоя сервера или одного из клиентов, поскольку в случае сбоя сервера он может быть перезапущен с восстановлением состояния вычислений из базы данных, а в случае сбоя клиента его задачи могут быть переданы на выполнение другим клиентам.

Серверная часть должна обеспечивать выполнение по запросам от клиентов следующих функций: отправки заказа, его получения, отправки значения, получения значения и отправки уведомления о завершении выполнения заказа. Также для отладки параллельных приложений к набору этих функций добавлены функции для получения текущего времени на сервере и вывода отладочных сообщений в создаваемый на сервере файл. Базовыми понятиями, на основе которых организован инструментарий, являются заказ, идентификатор значения и идентификатор неизвестного значения.

Идентификатор значения используется в качестве ссылки на передаваемое значение параметра заказа. Всякое описание заказа содержит лишь идентификаторы, а не сами значения параметров. Это позволяет избегать повторной передачи одних и тех же значений параметров, если они используются в различных вызовах, и тем самым снижать нагрузку на сеть за счет кеширования значений, привязанных к идентификаторам.

Понятие идентификатора неизвестного значения возникает из формулировки второго принципа предложенного подхода: поскольку после отправки заказа выполнение вызвавшей его процедуры должно продолжаться, всякому входному и транзитному параметру этого заказа необходимо присвоить некоторый идентификатор, который затем позволит получить реальное значение соответствующего параметра. Таким образом, результатом выполнения заказа является установка связи идентификатора неизвестного значения каждого из выходных или транзитных параметров этого заказа с некоторым идентификатором значения или идентификатором неизвестного значения. Возможность установки связи идентификатора неизвестного значения не с идентификатором значения, а с другим идентификатором неизвестного значения не вполне очевидна, но она возникает, например, в ситуации, когда некий заказ в качестве выходного параметра передает значение величины, вычисляемой в другом заказе, и не обращается к этой величине в процессе выполнения.

Понятие заказа возникает как формаль-

ное описание информации, требуемой для выполнения заказа. Эта информация включает в себя указание о том, какая процедура должна быть выполнена, информацию о параметрах в момент старта процедуры и в момент ее завершения. Информация о параметрах в момент старта представляет собой набор идентификаторов значений или идентификаторов неготовых значений, присвоенных входным и транзитным параметрам заказа, значения которых были переданы в момент вызова, замененного отправкой заказа. Информация о выходных параметрах – набор идентификаторов неготовых значений, которые были присвоены выходным и транзитным параметрам заказа. Следует отметить, что подобное представление позволяет достаточно быстро ответить на вопрос о том, совпадает ли информация о входных параметрах двух заказов на выполнение одной и той же процедуры. Если различные идентификаторы значений не могут быть связаны с одним и тем же значением, достаточно потребовать равенства всех соответствующих параметров. Причем равенство параметров вводится следующим образом: идентификаторы значений равны, если они совпадают, идентификаторы неизвестных значений или идентификатор значения и идентификатор неизвестного значения – если они связаны. Такое сравнение позволяет в случае детерминированных алгоритмов быстро находить ситуацию, в которой некоторый заказ отправлен в процессе вычислений более одного раза, и вместо последующих выполнений подставлять результаты первого выполнения. В частности, аналогичная ситуация может возникать при повторном решении задачи после внесения изменений во входные данные, если часть вычислений при этом не изменяется.

Кроме клиентской и серверной частей в состав инструментария входит также программа администрирования, позволяющая получать информацию о состоянии кластера и выполнении программ, запускать выполнение параллельных приложений и получать результаты счета, а также программа для генерации служебного кода, используемая при разработке.

Рассмотрим детально подсистемы инструментария, представленные на рис. 1.

Блок взаимодействия с пользовательским кодом и блок генерации исходного кода. Функциональность, доступная пользователю, заключена в нескольких классах. Во-первых, это генерируемый класс библиотеки, который для каждой из выделенных пользователем процедур содержит метод для отправки заказа на выполнение этой процедуры. Список выделенных процедур при этом должен декларироваться пользователем в *XML* файле, причем для каждой процедуры должно быть задано ее расположение, имя генерируемого метода, а также для каждого параметра – тип, имя и направление. Во-вторых, это класс *ServConnector*, содержащий средства для работы с расположенными на сервере файлами и добавления записей в лог-файл, ведущийся на сервере. В-третьих, это класс *DataHandler*, от которого должны быть порождены все классы, использующиеся в качестве типов параметров процедур. Пользователю доступны две группы методов этого класса.

1. Абстрактные методы для сериализации и восстановления состояния. Для передачи параметров процедур не используется стандартный механизм сериализации, поскольку при передаче выходных параметров их значения нужно вносить в уже существующие объекты, в то время как существующий механизм позволяет лишь создать новый объект. Поэтому создаваемые пользователем классы должны реализовывать эти методы.

2. Методы *preRead*(), *preChange*(), *preReplace*(), которые должны вызываться перед обращением к данным, частичным изменением данных и полной заменой содержимого объекта соответственно. Для потомков этого класса вводится правило, что они должны предоставлять доступ к содержащимся в них данным только через свои методы, в которых вызываются эти три унаследованных метода. Методы *preRead* и *preChange* убеждаются, что в объекте есть данные, а *preReplace* сбрасывает флаг о том, что в объекте нет данных, если он был установлен. Введение двух методов *preRead* и *preChange* позволяет, если после сериализации данных объекта производился только доступ на чте-

ние, обнаруживать эту ситуацию и не производить второй раз сериализацию.

Блок взаимодействия с пользовательским кодом является связующим звеном между конкретными прикладными алгоритмами и инструментарием. Его задача состоит в том, чтобы совместно с сгенерированными классами, с одной стороны, предоставить выполняющимся заказам возможность отправки заказов и базовый класс для неготовых величин, реализующий их семантику, и, с другой стороны, предоставить остальным частям инструментария доступ к пользовательской задаче как к некоторому “черному ящику”, способному по предоставленному описанию заказа выполнить его, связав идентификаторы неготовых значений выходных параметров с идентификаторами готовых значений или другими идентификаторами неготовых значений, или же в случае невозможности завершить выполнение предоставить подробную информацию о сбое. Также данный блок занимается сбором статистики и управлением набором выполняющихся потоков вычислений, предоставляя локальному планировщику необходимую информацию и приостанавливая или запуская выполнение потоков согласно полученным от него указаниям.

С точки зрения исходного кода блок связи с пользовательским кодом представлен классом потока выполнения заказа *ExecuterThread*, классом вспомогательной функциональности инструментария *ServConnector*, классом связи с потоками выполнения *StubConnector* и базовым классом для структур данных *DataHandler*, размещенными в пакете *org.parallel.client*. Класс потока выполнения отвечает за получение сведений о заказе и его выполнение, а класс связи содержит методы, используемые для отправки заказов, получения данных по требованию, а также некоторые средства управления потоками выполнения, которые используются планировщиком.

С блоком взаимодействия с пользовательским кодом тесно связан генератор исходного кода служебных классов, размещенный в пакете *org.parallel.sourcegen*. Его работа сводится к чтению предоставленного *XML* файла с описанием требуемых методов для отправки заказов, проверке его корректности (проверка наличия упоминаемых в файле классов и методов не производится, чтобы не усложнять разработку), генерации исходного кода классов библиотеки, локальной библиотеки, удаленной библиотеки и исполнителя. Проверка наличия упоминаемых классов не производится, так как они, в свою очередь, могут использовать классы библиотеки и такая проверка привела бы к кольцевой зависимости и усложняла бы компиляцию приложения. Назначения этих служебных классов таковы:

Класс библиотеки содержит абстрактные методы для отправки заказов, которые могут использоваться методами выполнения заказов. Для того, чтобы получить доступ к классу библиотеки, метод выполнения заказа должен принимать его как параметр и информация об этом должна быть приведена в файле описания пользовательской задачи. При этом генератор будет записывать в исходном коде вызовы методов с передачей этого параметра.

Класс локальной библиотеки – реализация класса библиотеки, в которой каждый из методов реализован как локальный вызов соответствующего метода. Этот класс не применяется в процессе выполнения пользовательской программы, но может быть использован для локальной отладки создаваемых приложений.

Класс удаленной библиотеки – реализация класса библиотеки, в которой каждый из методов реализует отправку соответствующего заказа средствами инструментария.

Класс исполнителя содержит методы, используемые инструментарием в процессе выполнения пользовательской программы.

Блок взаимодействия с пользовательским кодом совместно со сгенерированными классами предоставляет пользовательскому коду операцию отправки заказа, а также реализует операции выполнения заказа и получения значения. Алгоритмы этих операций представлены ниже.

Планировщик. Планировщик, по сравнению с остальными частями инструментария, содержит достаточно сложную логику. Он реализован классом *org.parallel.client.Scheduler*. Реализация пла-

нировщика основана на относительно простой эвристике наивного планирования, предусматривающей преимущество выполняющихся заказов над еще не стартовавшими. Каждый раз, когда общее количество процессов выполнения заказов оказывается меньше количества процессоров в компьютере, планировщик сначала пытается заполнить процессоры путем продолжения выполнения ранее приостановленных заказов, а затем, если это не удается, загружает новые заказы с сервера. Если это сделать не удается, планировщик начинает периодически опрашивать сервер, загружая появляющиеся задания (подобный опрос сервера планируется в последующих версиях инструментария заменить получением уведомлений от сервера). Затем, когда количество выполняющихся заказов сравняется с количеством процессоров, опрос сервера прекращается.

Планировщик вызывается в двух ситуациях: в случае завершения выполнения заказа и перед началом ожидания данных.

```
• Алгоритм выполнения заказа:
○ получить заказ с сервера;
○ для (каждого параметра процедуры) {
○  если (параметр - входной или транзитный) {
○   если (значение параметра известно) {
○    установить значение параметра из данных о заказе;
○   } иначе {
○    связать параметр с идентификатором из данных о заказе;
○   }
○  } иначе {
○   установить значение по умолчанию;
○  }
○ }
○ выполнить соответствующую процедуру;
○ для (каждого выходного или транзитного параметра процедуры) {
○  отправить на сервер значение параметра;
○ }
○ уведомить планировщик об освобождении процессора;
• Алгоритм отправки заказа:
○ отправить на сервер номер процедуры, которую нужно выполнить;
○ для (каждого входного или транзитного параметра) {
○  если (значение параметра известно) {
○   отправить значение параметра на сервер;
○  } иначе {
○   отправить на сервер связанный с параметром идентификатор;
○  }
○ }
○ получить с сервера набор идентификаторов;
○ для (каждого выходного или транзитного параметра) {
○  связать параметр с очередным идентификатором;
○ }
○ уведомить планировщик о появлении заказа;
• Алгоритм получения значения:
○ получить идентификатор, связанный со значением;
○ запросить значение с сервера по идентификатору;
○ если (значение еще не вычислено) {
○  сообщить идентификатор планировщику;
○  уведомить планировщик об освобождении процессора;
○  приостановить работу до указания от планировщика;
○  запросить значение с сервера по идентификатору;
○ }
○ сбросить связь значения с идентификатором;
```

Отметим, что при работе планировщика, вызываемого перед началом ожидания данных, нужно учитывать то обстоятельство, что планировщик вызывается из потока выполнения заказа. Это означает, что возможна ситуация, в которой планировщик может попытаться произвести некоторые действия над текущим потоком. Например, попытаться его разбудить еще до начала ожидания данных. Случай, когда планировщик вызывается после завершения выполнения заказа, подобных проблем не вызывает, поскольку никаких действий над текущим заказом планировщик произвести не может.

Диаграмма состояний заказа представлена на рис. 2. Из шести представленных состояний в процессе своего выполнения на клиентском компьютере заказ может быть только в одном из трех состояний: готов, блокирован, выполняется. Для эффективного управления заказами в процессе выполнения выполняющиеся и готовые заказы собраны в списки, а ожидающие данных – в ассоциативный массив, в котором идентификаторам неготовых значений поставлены в соответствие списки заказов, их ожидающие.

Блок кеширования. Блок кеширования данных в клиентской части, представленный классом *ProblemServiceProxy* пакета *org.parallel.client,* хранит полученные от сервера данные о значениях, связанных с идентификаторами значений, а также о связях идентификаторов неготовых значений с идентификаторами значений и друг с другом. Для хранения значений, связанных с идентификаторами значений, используется механизм мягких ссылок. Данный класс является реализацией удаленного интерфейса *ProblemService*. То, что все запросы к серверу проходят через этот блок, позволяет не получать повторно уже имеющиеся на клиенте данные, что уменьшает нагрузку на сеть.

API сервера. Взаимодействие между клиентом и сервером организовано с использованием технологии RMI. Все классы, используемые для удаленных вызовов, собраны в пакете *org.parallel.protocol*. Удаленные методы собраны в два интерфейса – *InfoService* и *ProblemService*. Интерфейс *InfoService* содержит методы, используемые для настройки клиентской части инструментария в начале процесса вычислений и используемые программой администрирования.

Интерфейс *ProblemService* содержит используемые в процессе вычислений методы для отправки заказов, получения заказов, отправки и получения данных, получения и отправки сведений о связях идентификаторов, отладочные и другие методы.

Также пакет *org.parallel.protocol* содержит некоторые используемые в описанных двух интерфейсах вспомогательные классы – перечисления для состояния заказа и направления использования параметра, а также класс описания заказа.

Блок обработки запросов. Блок обработки запросов клиентов, представленный классами *InfoServiceImpl*, *ProblemServiceImpl* и *Problem* пакета *org.parallel.server*, реализует методы, используемые в процессе решения задач такие, например, как метод для получения идентификатора значения по идентификатору неготового значения, метод для отправки заказа, метод для записи отладочного сообщения в расположенный на сервере файл.

Практически каждый метод, используе-

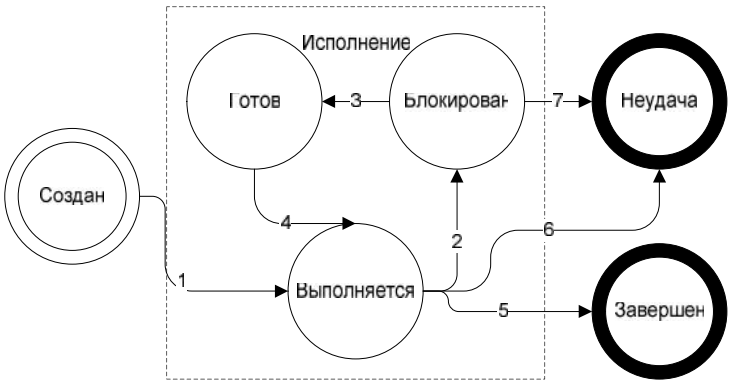

Рис. 2. Диаграмма состояний заказа

мый в качестве удаленного метода, сводится к выполнению соответствующей операции с базой данных – например, к установке связи между идентификаторами, установке статуса заказа или получению данных по идентификатору. Исключение составляют методы, используемые для сохранения отладочной информации на сервере, и методы для доступа к содержащимся на сервере файлам.

Подсистема управления сущностями. Классы, реализующие функциональность по работе с базой данных, собраны в пакете *org.parallel.server.database*. Базовым классом в данной функциональности является *GenericConnector*. Для класса *GenericConnector* реализован потомок, работающий с базой данных с данными одной задачи – *ProblemConnectorImpl*. Для того, чтобы обеспечить возможность хранения промежуточных результатов в памяти без использования внешней базы данных, из данного класса выделен интерфейс *ProblemConnector*, для которого созданы еще две реализации – *ProblemConnectorProxy* и *ProblemConnectorMemoryImpl*. Первая из них использует для хранения данных память, а вторая является прослойкой, добавляющей кеширование на основе механизма мягких ссылок к реализации на основе внешней базы данных. Доступ к базе данных реализован при помощи библиотеки *JDBC* (*Java Database Connectivity*).

Выводы. Представленная технология неявного распараллеливания кластерных вычислений, основанного на заказах позволяет для алгоритмов, реализуемых с использованием параллелизма заданий, достаточно легко переходить от существующих последовательных реализаций к параллельным реализациям, внося незначительные изменения как в код, так и в логику работы алгоритма прикладной задачи.

Предложенная технология реализована в виде инструментария на основе средств языка программирования Java. Описан реализующий ее инструментарий с точки зрения системного разработчика и с точки зрения прикладного программиста. Рассмотрены основные вопросы, возникающие как при разработке инструментария, так и при его практическом использовании.

Список использованной литературы

1. Антонов А.С. Параллельное программирование с использованием технологии MPI. – М.: Изд-во МГУ, 2004. – 71 с.

2. Воеводин В.В. Параллельные вычисления / В.В.Воеводин, Вл. В.Воеводин. – СПб.: БХВ–Петербург: 2002. – 608 с.

3. Open TS: архитектура и реализация среды для динамического распараллеливания вычислений / С.М.Абрамов, А.А.Московский, В.А.Роганов, Ю.В.Шевчук, Е.В.Шевчук, Н.Н.Парамонов, О.П.Чиж // Научный сервис в сети Интернет: технологии распределенных вычислений: Труды Всероссийской научной конференции, 19–24 сентября 2005, Новороссийск. – М.: Изд-во МГУ. – С. 79 – 81.

4. Павленко В.Д. Построение пространства диагностических признаков на основе моделей объектов контроля в виде рядов Вольтерра / В.Д.Павленко, А.А.Фомин, В.В.Череватый // Вторая международная конференция по проблемам управления. Москва, Институт проблем управления им. В.А.Трапезникова РАН, 17–19 июня 2003.: Избранные труды в двух томах. – М.: Институт проблем управления, 2003. – Т.2. – С. 110 – 117.

5. Павленко В.Д. Технология кластерных вычислений с использованием транспарентного распараллеливания, основанного на заказах / В.Д.Павленко, В.В.Бурдейный // Вісник Національного технічного університету «Харківський політехнічний інститут»: Збірник наукових праць. Тематичний випуск «Системний аналіз, управління та інформаційні технології». – Харків: НТУ «ХПІ». – 2007. – № 6. – С. 84–107.

6. Тхань Фыонг Нгуен. Параллельная реализация алгоритмов фильтрации космических изображений / Тхань Фыонг Нгуен, А.Ю. Шелестов // Проблемы управления и информатики. – 2005. – № 5. – С. 121–132.

7. Khoroshevsky V.G. Space-distributed multicluster computer system for parallel multiprogramme regimes modeling / V.G.Khoroshevsky, S.N.Mamoilenko, S.Maidanov, M.S.Sedelnikov // Сборник трудов конференции “Моделирование–2006”, 16–18 мая 2006, Киев: ИПМЭ им. Г.Е.Пухова НАНУ. –

С. 67 – 69.

8. Kolding T.E. High Order Volterra Series Analysis Using Parallel Computting / T.E.Kolding, T.Larsen – http: //citeseer.ist.psu. edu/242948. html.

9. Pavlenko V. Computing Simulation in Orders Based Transparent Parallelizing / V.Pavlenko, V.Burdeinyi // ICIM' 2008: Proceedings 2nd International Conference on Inductive Modelling, September 15–19, 2008, Kiyv: Ukraine. – P.168–171.

10. Roganov V. The Open TS parallel programming system / V.Roganov, A.Moskovsky, S.Abramov // The Twelfth International Conference on Parallel and Distributed Systems, Minneapolis, USA (ICPADS, July 12-15, 2006). – http://skif.pereslavl.ru/skif/index.cgi?module=chap&action=getpage&data=publications\pub2006\opents_ext.doc

Получено 10.01.2012

References

1. Antonov A.S. Parallel programs mation technology with MPI. – Moscow: MGU, 2004. – 71 p. [in Russian].

2. Voevodin V.V., V.V.Voevodin. Parallel computing . – St. Petersburg.: BHV–Petersburg: 2002. – 608 p. [in Russian].

3. Abramov S.M., Moskovsky A.A., Roganov V.A., Shevchuk Yu.V., Shevchuk E.V., N.N.Paramonov, O.P.Chizh. Open TS: architecture and implementation of a dynamic environment for parallel computting / Scientific service on the Internet: technology, energy distributed computing: Proceedings of the Scientific Conference, 19–24 September 2005, Novorossiysk. – Moscow: Moscow State University. –P.79– 81 [in Russian].

4. Pavlenko V.D., Fomin A.A., Cherevaty V.V. Building a space diagnostic features of model based control of objects in the form of Volterra series / Second International Conference on Control. Mowells, the Institute of Control Sciences. V.A. Trapeznikov Academy of Sciences, June 17–19, 2003.: Selected works in two volumes. – Moscow.: Institute of Control, then, in 2003 – V.2. – P.110–117 [in Russian].

5. Pavlenko V.D., Burdeyny V.V. Technology cluster calculations using transparent-tion parallel, based on orders / News Natsionalnogotehnichnogo universitetu "Harkivsky politehnichny institut": Naukova–Zbirnik–Pratzen. Subject–particle vipusk "System analiz, upravlinnya that informatsiynitehnologiï."– Harkiv: NTU "HPI." – 2007. – № 6. –P. 84–107 [in Russian].

6. Thanh Phuong Nguyen, Shelestov A. Parallel implementation of algorithms for filtering of cosmicimages / Cybernetics and Informatics. – 2005. –№ 5. – P. 121–132. [in Russian].

7. Khoroshevsky V.G., Mamoilenko S.N, Maidanov S., Sedelnikov M.S. Spacedistributed multicluster computer system for parallel multiprogramme regimes modeling / Proceedings of the conference. "Modelling 2006", 16–18 May 2006. – Kiev: IPME them. G.E Pukhov National Academy of Sciences. - P. 67–69 [in Russian].

8. Kolding T.E., Larsen T. High Order Volterra Series Analysis Using Parallel Computting – http://citeseer.ist.psu. edu/242948.html [ ].

9. Pavlenko V., Burdeinyi V. Computing Simulation in Orders Based Transparent Parallelizing / ICIM' 2008: Proceedings 2nd International Conference on Inductive Modelling, September 15-19. – 2008. – Kiev: Ukraine. – P.168–171 [in Russian].

10. Roganov V., Moskovsky A., Abramov S. The Open TS parallel programming system / The Twelfth International Conference on Parallel and Distributed Systems, Minneapolis, USA (ICPADS, July 12–15, 2006). – http://skif.pereslavl.ru/skif/index.cgi?module=chap&action=getpage&data=publications\pub2006\opents_ext.doc

Павленко
Виталий Данилович,
канд.техн.наук, ст.науч.сотрудн.,
доц.каф.комп.сист.управл.
Одесск нац. политехн. ун-та,
тел.:(048)734–85–79.
E–mail: pavlenko_vitalij@mail.ru.

Бурдейный
Виктор Викторович,
Аспирант каф. комп.сист.управл.
Одесск. нац. политехн. ун-та,
тел.:+38-066-409-64-64,
E–mail: vburdejny@gmail.com