ВЕСТНИК

ЮЖНО-УРАЛЬСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

2014
Т. 3, №3

ISSN 2305-9052

СЕРИЯ

«ВЫЧИСЛИТЕЛЬНАЯ МАТЕМАТИКА И ИНФОРМАТИКА»

Учредитель — Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования «Южно-Уральский государственный университет» (национальный исследовательский университет)

Основной целью издания является пропаганда научных исследований в следующих областях:

- Вычислительная математика и численные методы
- Математическое программирование
- Распознавание образов
- Вычислительные методы линейной алгебры
- Решение обратных и некорректно поставленных задач
- Доказательные вычисления
- Численное решение дифференциальных и интегральных уравнений
- Исследование операций
- Теория игр
- Теория аппроксимации
- Информатика
- Математическое и программное обеспечение высокопроизводительных вычислительных систем
- Системное программирование
- Распределенные вычисления, облачные и грид-технологии
- Технология программирования
- Машинная графика
- Интернет-технологии
- Системы электронного обучения
- Технологии обработки баз данных и знаний
- Интеллектуальный анализ данных

BULLETIN

OF THE SOUTH URAL STATE UNIVERSITY

SERIES

"COMPUTATIONAL MATHEMATICS AND SOFTWARE ENGINEERING"

2014
Vol. 3, no. 3

ISSN 2305-9052

Vestnik Yuzhno-Ural'skogo Gosudarstvennogo Universiteta.
Seriya "Vychislitel'naya Matematika i Informatika"

South Ural State University

The main purpose of the series is publicity of scientific researches in the following areas:

- Numerical analysis and methods
- Mathematical optimization
- Pattern recognition
- Numerical methods of linear algebra
- Reverse and ill-posed problems solution
- Computer-assisted proofs
- Numerical solutions of differential and integral equations
- Operations research
- Game theory
- Approximation theory
- Computer science
- High performance computer software
- System programming
- Distributed, cloud and grid computing
- Programming technology
- Computer graphics
- Internet technologies
- E-learning
- Database and knowledge processing
- Data mining

Содержание

Contents

УДК 004.93'1, 004.272.23, 004.258, 004.272.45

ЭФФЕКТИВНАЯ ДЕТЕКЦИЯ ЛИЦ НА МНОГОЯДЕРНОМ ПРОЦЕССОРЕ EPIPHANY[1]

А.А. Сухинов, Г.Б. Остроброд

В статье рассматривается возможность использования энергоэффективного микропроцессора Epiphany для решения актуальной прикладной задачи — детекции лиц на изображении. Этот микропроцессор представляет собой многоядерную вычислительную систему с распределенной памятью, выполненную на одном кристалле. Из-за малой площади кристалла микропроцессор обладает существенными аппаратными ограничениями (в частности, он имеет всего 32 килобайта памяти на ядро), которые ограничивают выбор алгоритма и затрудняют его программную реализацию. Для детекции лиц адаптирован известный алгоритм, основанный на каскадном классификаторе, использующем LBP-признаки (Local Binary Patterns). Показано, что микропроцессор Epiphany, имеющий 16 ядер, может на этой задаче в 2,5 раза обогнать одноядерный процессор персонального компьютера той же тактовой частоты, при этом потребляя лишь 0,5 ватта электрической мощности.

Ключевые слова: детекция лиц, локальные бинарные шаблоны, параллельная обработка данных, специализированные микропроцессоры, распределенная память.

Введение

Совершенствование технологий производства процессоров в плане миниатюризации привело к повсеместному использованию многоядерных микропроцессоров, в том числе в мобильных приложениях. Типичные многоядерные процессоры с точки зрения прикладного программирования являются системами с общей памятью, однако их низкоуровневая архитектура — это система с распределенной памятью; каждое ядро имеет свою кэш-память; кэш-памяти отдельных ядер синхронизируются при помощи того или иного протокола когерентности кэшей [1, 2]. Такой подход облегчает создание программного обеспечения, однако существенно усложняет процессор, увеличивая требуемую площадь кристалла и энергопотребление. Кроме того, снижается возможная эффективность использования процессора; при правильном программировании можно получить большее быстродействие от системы с распределенной памятью, чем от «ненастоящей» системы с общей памятью [3].

Энергоэффективный микропроцессор Epiphany производства Adapteva является образцом другого подхода: каждое ядро имеет память небольшого объема, явно управляемую прикладной программой. Ядра связаны между собой сетями передачи данных.

Цель данной статьи — демонстрация возможности эффективного использования микропроцессоров Epiphany для решения актуальной прикладной задачи — детекции лиц на изображении. Будет показано, что микропроцессор Epiphany, имеющий 16 ядер, может на этой задаче в 2,5 раза обогнать одноядерный процессор персонального компьютера той же тактовой частоты, при этом потребляя лишь 0,5 ватта электрической мощности.

Программный код, описанный в статье, доступен на странице проекта в Интернете [4].

[1]Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии — 2014».

1. Архитектура Epiphany

Многоядерные микропроцессоры Epiphany представляют собой IP (intellectual property) блоки, предназначенные для разработки систем-на-кристалле. Блок может быть сконфигурирован на количество ядер от 1 до 4096. Типичная тактовая частота — 1 ГГц, объем памяти ядра — 32 или 64 килобайта. Кроме того, возможно подключение внешней памяти объемом до 2-х гигабайт. Микропроцессор оптимизирован для снижения энергопотребления и занимаемой площади кристалла: при быстродействии 1 GFLOPS одно ядро имеет площадь 0,3 мм^2 и энергопотребление всего 14 мВт.

Архитектура Epiphany активно совершенствуется, поэтому у актуальных экземпляров микропроцессоров характеристики могут отличаться от приведенных.

Малый объем памяти ядра и отсутствие виртуальной памяти не позволяют запускать на микропроцессорах Epiphany популярные операционные системы, поэтому микропроцессор позиционируется как ускоритель вычислений для мобильных приложений, либо как центральный процессор устройства, не имеющего операционной системы.

Тестовые экземпляры микропроцессоров Epiphany доступны в виде готовых модулей Parallela [5], представляющих собой компьютер архитектуры ARM, имеющий размеры кредитной карты. В таком модуле Epiphany установлен в качестве дополнительного процессора, поэтому далее будем называть его *сопроцессором*. Каждое ядро Epiphany имеет:

- 32 или 64 килобайта *локальной памяти*, разделенной на банки по 8 килобайт; каждый банк в один такт может обслуживать только одного потребителя.
- *Арифметико-логическое устройство*, способное за один такт выполнять операции над 32-битными числами (целыми или с плавающей точкой).
- *64 регистра общего назначения.*
- *Двухканальный контроллер прямого доступа к памяти* (DMA), способный самостоятельно выдавать команды пересылки данных.
- *2 таймера* для подсчета различных событий (например, тактов сопроцессора).
- *Контроллер прерываний*, позволяющий устанавливать обработчики событий.
- *Контроллер пересылки данных между ядрами* (маршрутизатор).

Логически память всех ядер объединена в единое 32-битное адресное пространство, что позволяет на программном уровне единообразно передавать данные между любыми ячейками памяти любой пары ядер. В адресное пространство также включена внешняя память и регистры ядер, что позволяет из одного ядра модифицировать регистры другого ядра. Единое адресное пространство позволяет ядру выполнять не только «свой» код, но и код, расположенный в памяти другого ядра или даже во внешней памяти.

Физически ядра связаны в прямоугольную решетку тремя сетями передачи данных:

- *Сеть записи* передает пакеты, содержащие адрес и данные. Когда пакет достигает целевого ядра, данные записываются в требуемые ячейки памяти или регистры.
- *Сеть чтения* передает пакеты, содержащие адрес чтения и адрес записи данных. Такие пакеты направляются ядру, содержащему адрес чтения. По достижению цели необходимые данные считываются из памяти или регистров, и формируется ответный пакет, передаваемый через сеть записи. Интересной особенностью описанного подхода является возможность пересылки данных между двумя адресами, внешними по отношению к ядру, выдавшему команду на такую пересылку.

— *Транзитная сеть* передает данные от границы до границы блока, состоящего из 4×4 ядер. Трафик транзитной сети не затрагивает трафик других сетей, что дает возможность формирования блоков размерами 4×4 ядра, работающих в реальном времени.

Хотя логически передача данных возможна между любой парой ядер, физически за один такт данные передаются только между соседними ядрами по горизонтали или вертикали. Это позволяет избежать необходимости синхронизации фазы тактовой частоты между ядрами, расположенными далеко друг от друга на кристалле. Вместо этого тактовая частота распространяется волной от точки ее ввода в кристалл, что снижает энергопотребление.

Команда записи данных завершается за один такт; никаких действий, подтверждающих запись, не производится. Алгоритмы маршрутизации гарантируют, что пакеты записи будут доставлены получателю (включая внешнюю память) в том же порядке, в котором они были посланы отправителем (при условии, что у этих пакетов один и тот же отправитель).

В связи с необходимостью прохождения ответного пакета, чтение из памяти другого ядра производится значительно медленнее, чем запись в память другого ядра.

На границах кристалла сети передачи данных соединены с электрическими выводами сопроцессора. Это позволяет подключить внешнюю память, основной процессор, а также соединить несколько сопроцессоров в прямоугольную решетку, тем самым увеличив количество ядер в адресном пространстве.

2. Прикладное программирование сопроцессора Epiphany

Для программирования сопроцессора используется адаптированный компилятор GCC, принимающий исходный код на языке Си. Для каждого ядра используется собственный исходный код. Двоичный файл, предназначенный для загрузки в сопроцессор, представляет собой образ памяти всех ядер. Так как регистры ядер отображены в адресное пространство, такой подход позволяет задать сопроцессору нужное состояние на момент старта.

Стандартная библиотека языка Си присутствует, однако ее исполняемый код и структуры данных находятся во внешней памяти, что означает медленность работы функций стандартной библиотеки, а также невозможность использования функций `malloc` и `free` для работы с локальной памятью. Поэтому память каждого ядра целесообразно распределять статически при помощи файла LDF (Linker Definition File), содержащего указания компоновщику по размещению в памяти всех глобальных объектов программы. При программировании следует учитывать следующие факторы:

— Сопроцессор Epiphany может выполнять чтение, запись, и арифметические операции только с 32-битными числами (целыми или с плавающей точкой). Операции над другими типами данных (в том числе меньшего размера) эмулируются компилятором, и поэтому требуют нескольких тактов сопроцессора.

— Невыровненный запрос к памяти (адрес не кратен размеру типа данных) приводит к остановке программы.

— При этом у Epiphany нет проблем с управляющей логикой: вызов функции и возврат из нее происходят быстро, условные переходы не вызывают серьезного снижения быстродействия (их предсказание делается компилятором в статическом режиме).

В связи с указанными особенностями было решено реализовать алгоритм детекции лиц (ту его часть, что работает на сопроцессоре) в рамках следующих ограничений:

— Изображения хранятся в виде восьмибитных целых чисел; все остальные данные хранятся в виде выровненных в памяти 32-битных целых чисел;

— Для вычислений используются только 32-битные целые числа.
— В экземпляре сопроцессора, с которым мы работали, не было операций целочисленного умножения и деления, поэтому из арифметических операций можно было использовать только сложение, вычитание, и битовые операции.

Как будет показано ниже, при правильном выборе алгоритма и его реализации указанные ограничения не приводят к существенному усложнению или замедлению кода.

3. Алгоритм детекции лиц

Для реализации было решено взять один из нескольких алгоритмов детекции лиц, использованных в популярной библиотеке обработки изображений OpenCV [6]. Реализация распространенного алгоритма позволит объективно сравнить производительность процессора Epiphany с другими процессорами, на которых работает OpenCV. В указанной библиотеке алгоритмы детекции лиц имеют следующую структуру:

1. По исходному черно-белому изображению строится *пирамида*, состоящая из *слоев* (изображений) уменьшающегося разрешения. Слои высокого разрешения нужны для детекции лиц маленького размера; слои низкого разрешения используются для детекции крупных лиц. Типичное отношение размеров соседних слоев составляет 1,2.
2. Производится сканирование каждого слоя пирамиды *окном* небольшого размера (например, 24×24 пикселя). При сканировании окна располагаются с перекрытием.
3. Для каждого положения окна вызывается *классификатор*, который решает, является ли предоставленный ему фрагмент изображения лицом. Для принятия решения используются числовые *признаки*, получаемые на основе пикселей окна.
4. Производится *группировка детекций*: окна, получившие положительные отклики классификатора, группируются по геометрическому сходству. Группы, имеющие менее определенного числа элементов (например, трех), отбрасываются. Для оставшихся групп вычисляются средние окна, которые признаются положительными детекциями.

Наибольшее время тратится на детекцию мелких лиц (обработку слоев пирамиды высокого разрешения), поэтому минимальный размер детектируемых лиц нужно ограничивать.

Интересно отметить, что алгоритмы данной структуры ощутимо проигрывают по точности человеческому зрению, несмотря на высокую точность используемых в них классификаторов. Причиной такого проигрыша является недостаточность информации в окне для выполнения решения лицо/не лицо; человек для такого решения использует контекст — большое количество дополнительной информации о наблюдаемой сцене.

Основным элементом алгоритма детекции лиц является классификатор, так как именно он принимает решение лицо/не лицо для текущего окна. Классификатор представляет собой статистическую математическую модель, которая параметризуется автоматически на основе обучающей выборки, в рамках процесса, называемого *машинным обучением* [7]. Машинное обучение выходит за рамки этой статьи, поэтому все коэффициенты классификатора, упомянутые ниже, будем считать известными.

Ввиду сильно ограниченного объема памяти сопроцессора, среди алгоритмов детекции лиц библиотеки OpenCV мы выбрали для реализации самый экономичный в плане памяти. Таковым оказался алгоритм, основанный на LBP-признаках (Local Binary Patterns [8]).

LBP-признак (в реализации OpenCV) — это целое число от 0 до 255, которое ставится в соответствие прямоугольной области изображения (являющейся подобластью окна детек-

ции), ширина и высота которой кратны трем:

$$\begin{aligned} \mathrm{LBP}(x, y, w, h) \stackrel{\text{def}}{=} & \ 2^7 \cdot [s_{00} \geqslant s_{11}] + 2^6 \cdot [s_{10} \geqslant s_{11}] + 2^5 \cdot [s_{20} \geqslant s_{11}] + \\ & + 2^0 \cdot [s_{01} \geqslant s_{11}] + 2^4 \cdot [s_{21} \geqslant s_{11}] + \\ & + 2^1 \cdot [s_{02} \geqslant s_{11}] + 2^2 \cdot [s_{12} \geqslant s_{11}] + 2^3 \cdot [s_{22} \geqslant s_{11}] \ . \end{aligned} \tag{1}$$

Здесь:

(x, y) — координаты левого верхнего пикселя рассматриваемой области;

$(w, h) = (3w, 3h)$ — размеры области для вычисления LBP-признака;

$s_{i,j} \stackrel{\text{def}}{=} \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} p(x + iw + x, y + jh + y)$ — сумма пикселей по прямоугольнику;

$p(x, y)$ — значение пикселя изображения (столбец x, строка y);

$$[a \geqslant b] \stackrel{\text{def}}{=} \begin{cases} 1, \text{ если } a \geqslant b; \\ 0, \text{ иначе.} \end{cases}$$

Смысл LBP-признака простой: рассматриваемая область делится на 3×3 прямоугольника, и отношения яркостей (больше-меньше) восьми нецентральных прямоугольников к центральному кодируются в виде восьмибитного двоичного числа. Следует заметить, что LBP-признаки здесь используются не для формирования гистограмм [8], а как замена признаков Хаара, часто используемых в алгоритмах детекции лиц [9].

Сам классификатор состоит из нескольких *стадий*, и поэтому называется *каскадным*. Только положительное прохождение всех стадий классификатора означает положительный результат классификации («обнаружено лицо»). Такой ход принятия решений позволяет отсеивать большинство окон на ранних стадиях. Положительное прохождение j-й стадии ($j = 1, \ldots, m$) определяется следующим неравенством:

$$\sum_{i=1}^{n^j} w_i^j \geqslant T^j, \quad w_i^j \stackrel{\text{def}}{=} \begin{cases} P_i^j, \text{ если } \mathrm{LBP}\left(x_i^j, y_i^j, w_i^j, h_i^j\right) \in \mathbf{S}_i^j; \\ x_i^j, \text{ иначе.} \end{cases} \tag{2}$$

Здесь:

n^j — количество признаков, используемых в стадии j классификатора;

w_i^j — вес, который ставится в соответствие признаку номер i стадии j;

$\mathbf{S}_i^j$ — множество «желательных» значений признака для области $\left(x_i^j, y_i^j, w_i^j, h_i^j\right)$;

P_i^j — вес, который приобретает область с «желательным» значением признака;

$x_i^j < P_i^j$ — вес, который дается области с «нежелательным» значением признака;

T^j — вес, который должен быть накоплен для успешного прохождения стадии.

Все коэффициенты классификатора собраны в следующем выражении:

$$K_{\mathrm{CV}} \stackrel{\text{def}}{=} \left(\left(x_i^j, y_i^j, w_i^j, h_i^j, \mathbf{S}_i^j, P_i^j, x_i^j\right)_{i=1}^{n^j}, T^j\right)_{j=1}^{m}. \tag{3}$$

Классификатор фронтальных лиц в OpenCV имеет $m = 20$ стадий. На начальной стадии вычисляются 3 признака, на последней — 10 признаков. Суммарный объем коэффициентов в выражении (3) составляет 8 килобайт, однако профилирование программы, использующей библиотеку OpenCV, показало, что загруженный классификатор занимает 300 килобайт оперативной памяти. Это вызвано использованием нескольких буферов памяти (хранящих стадии классификатора, признаки, веса и другие данные), ссылающихся друг на друга при помощи индексов и указателей, использованием классов с таблицами виртуальных функций, а также хранением данных, не нужных для решения данной задачи.

4. Программная реализация детектора лиц

4.1. Расположение структур данных

Прежде всего нужно решить, где располагать *данные классификатора*. Были проработаны следующие варианты:

Данные классификатора во внешней памяти. Это самый простой для реализации вариант. Однако ввиду того, что вычисления, связанные с классификацией (выражения (1), (2)), достаточно простые, а чтение данных из внешней памяти выполняется медленно, большую часть времени ядра будут простаивать в ожидании получения данных. Ситуация усугубляется тем, что все ядра требуют данные одновременно, конкурируя за канал обмена сопроцессора с внешней памятью.

Стадии классификатора распределены между ядрами. При таком подходе каждое окно детекции обрабатывается ядрами сопроцессора по конвейеру. Однако обработка большинства окон прерывается на ранних стадиях; для того чтобы ядра были постоянно обеспечены работой, требуется, чтобы для первых стадий классификатора было выделено больше ядер, чем для последних стадий. Требуется нетривиальная синхронизация и балансировка загрузки (текстурированные участки изображения проходят больше стадий, чем гладкие), что представляется сложным для программирования.

Копия классификатора в каждом ядре. Это самый предпочтительный вариант, так как ядра могут выполнять независимые подзадачи — классификацию различных окон. Однако в нашем распоряжении был сопроцессор, имеющий всего 32 килобайта памяти на ядро, и для хранения классификатора оставалось не более 8-ми килобайт, что гораздо меньше измеренного объема данных классификатора OpenCV (300 килобайт). Как будет показано далее, нам удалось очень компактно сохранить данные классификатора, поэтому именно этот вариант был выбран для реализации.

Затем надо определиться со способом формирования и хранения *пирамиды изображений*:

Размещение в общей памяти только исходного изображения. Каждое ядро обрабатывает фрагмент этого изображения (будем называть его *тайлом*), и строит в локальной памяти соответствующую часть пирамиды. Тайлы должны быть маленькими, чтобы уместиться в памяти ядра, и должны перекрываться, чтобы не потерялись возможные детекции на их стыках. При формировании слоя пирамиды более низкого разрешения ширина перекрытия оказывается недостаточной; ядра должны обмениваться данными, передавая друг другу недостающие пиксели для обработки стыков. Однако все изображение может не вместиться в сопроцессор, часть пикселей, требуемых для обработки границ тайлов, отсутствует, и неясно, как эти пиксели вычислять.

Вся пирамида находится в общей памяти. Вначале сопроцессор строит пирамиду из исходного изображения, располагая ее в общей памяти, а затем слои обрабатываются перекрывающимися тайлами. Это представляется достаточно эффективным, так как обработка тайла занимает гораздо больше времени, чем требуется для его пересылки. Был выбран данный подход, однако соответствующий код не вместился в память ядра сопроцессора, поэтому построение пирамиды выполняется центральным процессором.

В итоге в общей памяти располагаются следующие структуры данных (рис. 1): пирамида изображений, очередь заданий (описаны далее) со счетчиками взятых и выполненных заданий, и данные классификатора. Также в общей памяти располагается счетчик запускаемых ядер сопроцессора. Со стороны Epiphany счетчики защищены мьютексом.

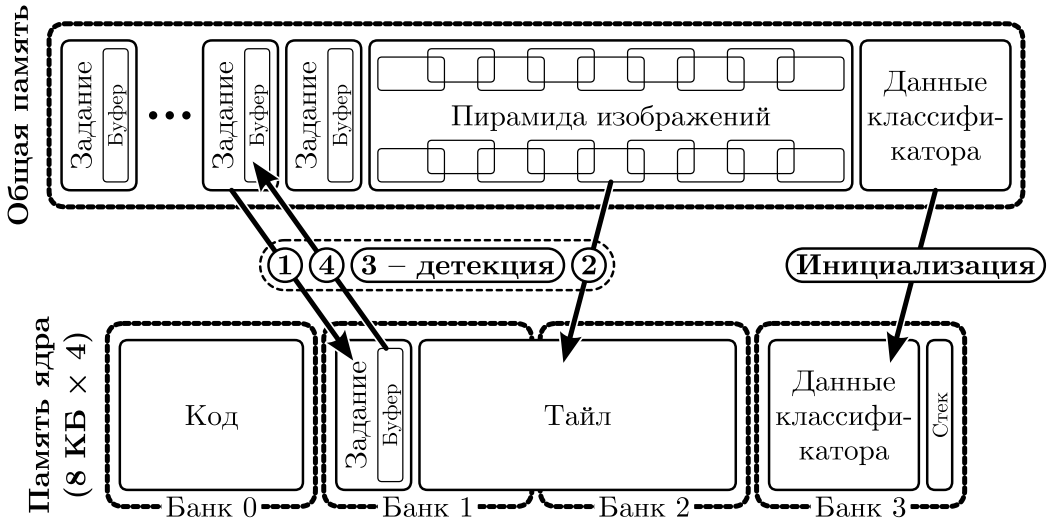


Рис. 1. Схема распределения памяти. Стрелками показаны пересылки данных

Память каждого ядра сопроцессора распределена следующим образом (рис. 1): 8 килобайт выделено на код, 16 килобайт на копию задания и обрабатываемый тайл, и 8 килобайт на данные классификатора и стек выполнения программы.

4.2. Параллельный алгоритм

В общих чертах, *алгоритм работы центрального процессора* следующий:

1. Записать в общую память классификатор.
2. Построить в общей памяти пирамиду изображений.
3. Логически разделить каждый слой пирамиды на частично перекрывающиеся прямоугольные тайлы размерами примерно 128×128 пикселей, и поставить в соответствие каждому тайлу задание, состоящее из координат тайла на изображении, адреса тайла в памяти, и буфера для записи результатов детекции лиц в тайле.
4. Установить счетчик запускаемых ядер в ненулевое значение и ждать, когда счетчик выполненных заданий достигнет общего количества заданий.
5. Выполнить группировку детекций, находящихся в очереди заданий.

Алгоритм работы ядра сопроцессора следующий (рис. 1): когда счетчик запускаемых ядер станет ненулевым, уменьшить его на единицу, скопировать данные классификатора в локальную память, и перейти в цикл обработки заданий. Задание выполняется так:

1. Наращивается счетчик взятых заданий, и соответствующее задание копируется в локальную память. Когда все задания взяты, цикл обработки заданий завершается.
2. Тайл, расположение которого указано в задании, копируются в локальную память.
3. Выполняется детекция лиц в тайле.
4. Результаты копируются в общую память, наращивается счетчик выполненных заданий.

Пересылки данных между внешней памятью и ядром производятся при помощи контроллера DMA. Между ядрами пересылаются только данные, нужные для работы мьютекса.

4.3. Реализация пирамиды изображений

Для пирамиды с равномерным шагом масштаба размеры (w_i, h_i) слоя номер $i = 1, \ldots, n$ определяются следующими соотношениями:

$$(w_1, h_1) = \frac{(w, h)}{s}, \quad (w_{i+1}, h_{i+1}) = \frac{(w_i, h_i)}{k}, \tag{4}$$

где (w, h) — размеры исходного изображения; $s \geqslant 1$ — масштаб начального слоя (определяется требуемым минимальным размером детектируемого лица); $k > 1$ — отношение масштабов соседних слоев (типичное значение: $k \approx 1{,}2$). Количество слоев n обычно наращивается, пока в последнем слое помещается хотя бы одно окно детекции.

Пирамида изображений должна иметь достаточно малый шаг масштаба (параметр k близкий к единице), чтобы для любого лица нашелся слой, на котором оно будет иметь размер, близкий к оптимальному (тому размеру, на котором был обучен классификатор). Для пирамиды с равномерным шагом масштаба максимально возможное отклонение размера проверяемого лица от оптимального составляет $\sqrt{k}$. Чем ближе это значение к единице, тем выше качество детекции лиц, но тем больше требуется слоев.

Очередной слой пирамиды можно получить путем передискретизации исходного изображения или одного из вычисленных ранее слоев большего разрешения. Чтобы уменьшить алиасинг [10] передискретизацию целесообразно выполнять интегрированием пикселей исходного изображения по площади пикселей уменьшенного изображения (метод Area в терминологии OpenCV). Вычислительная сложность подобной процедуры примерно пропорциональна сумме площадей исходного и уменьшенного изображений.

Для снижения объема вычислений возникает желание вычислять каждый слой пирамиды на основе предыдущего. К сожалению, это приводит к появлению муара [10], выражающегося в виде волнообразного изменения четкости изображения с периодом $1/(k-1)$. Сам по себе муар — явление незначительное, однако при последовательном уменьшении изображений он накапливается, существенно снижая качество детекции.

В библиотеке OpenCV проблемы алиасинга и муара при построении пирамиды решены следующим образом: каждый слой пирамиды строится независимо от остальных слоев путем масштабирования исходного изображения методом Area. В итоге время построения пирамиды в OpenCV составляет $T_{\text{OpenCV}} = O(w \cdot h \cdot n)$.

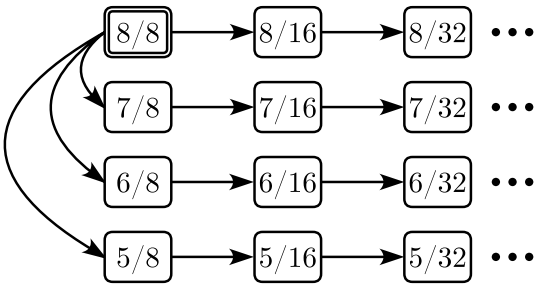


Рис. 2. Схема построения пирамиды изображений

В нашей реализации пирамида имеет фиксированные масштабные множители и строится следующим образом (рис. 2): исходное изображение разбивается на блоки размерами 8×8 пикселей, по которым вычисляются уменьшенные блоки размерами 7×7, 6×6 и 5×5 пикселей. За один проход получаются сразу 3 дополнительных слоя пирамиды, име-

ющих размеры 7/8, 6/8 и 5/8 от оригинала. Любой следующий слой, имеющий номер i, получается двукратным уменьшением (усреднением блоков 2×2 пикселя) слоя номер $i-4$.

При таком подходе отношения масштабов соседних слоев немного различаются, однако это не влияет на качество детекции лиц. Последовательное уменьшение размера в 2 раза не приводит к накоплению погрешности и появлению муара (его период равен 1). Если нужно увеличить минимальный размер детектируемых лиц, следует при детекции пропустить нужное количество слоев высокого разрешения.

Время построения пирамиды описанным способом составляет $T_{\text{PyrEP}} = O(w \cdot h)$. Замеры показали, что OpenCV успевает построить в среднем всего два слоя пирамиды за время, требуемое для построения всей пирамиды (примерно 20 слоев) в нашей реализации.

4.4. Реализация LBP-признаков

Для быстрого вычисления сумм $s_{i,j}$ в выражении (1) библиотека OpenCV использует интегральное преобразование изображения [11]. Сумма по любому прямоугольнику вычисляется за 4 чтения из памяти и 3 арифметических операции. Интегральное изображение вычисляется за один проход по слою пирамиды и хранится в виде чисел типа `int`.

Хранение четырех байт на пиксель интегрального изображения оказалось неприемлемым из-за ограниченной памяти сопроцессора Epiphany, однако мы нашли способ эффективного вычисления LBP-признаков без использования интегрального преобразования. Для этого достаточно аппроксимировать суммы $s_{i,j}$ четырьмя слагаемыми (аналогично численному интегрированию методом центральных прямоугольников). Отличия в результатах детекции лиц, вызванные такой аппроксимацией, оказались несущественными.

Можно было бы оставить от каждой суммы всего по одному центральному пикселю, и принять такие признаки за некоторые новые LBP-признаки, но такой подход уже требовал переобучения классификатора для достижения точности детекции OpenCV. Ценность нашего кода значительно выше, если он может использовать готовые обученные классификаторы из OpenCV, поэтому мы отказались от замены каждой суммы $s_{i,j}$ на одно слагаемое.

Подвыражения вида $2^7 \cdot [s_{00} \geqslant s_{ii}]$ из выражения (1) в OpenCV реализованы при помощи условных операторов. В нашей реализации они вычисляются взятием знакового бита разности $s_{00} - s_{ii}$, и сдвигом этого бита вправо на нужное количество разрядов, что дает дополнительный выигрыш в быстродействии.

4.5. Реализация каскадного классификатора

Требуется не только уменьшить объем данных классификатора, но и сделать структуру данных такой, чтобы исполняемый код был компактным и быстродействующим. Фактически нужно минимизировать сумму объемов данных классификатора и исполняемого кода, использующего эти данные: вместе они должны составлять менее 16-ти килобайт.

Было решено разместить все данные классификатора в одном буфере памяти в том порядке, в котором классификатор их использует. Это позволило избавиться от всех ссылок, которые использованы в OpenCV для связи буферов. В итоге мы пришли к архитектуре классификатора, напоминающей виртуальную машину. Данные классификатора представлены в виде последовательности команд. Каждая команда имеет фиксированный размер и содержит в себе необходимые для выполнения параметры, представленные в виде целых чисел. Возможны 3 вида команд:

— *Вычислить LBP-признак*, и прибавить его оценку к текущей оценке классификатора.
— *Команда окончания стадии*: сравнить текущую оценку с числом, записанным в команде. Если текущая оценка меньше, вернуть «не лицо», иначе обнулить текущую оценку.
— *Команда окончания классификации*: вернуть «лицо».

Для минимизации количества данных, содержащихся в команде 1, были выполнены описанные далее модификации. Прежде всего, условие (2) можно переписать следующим образом:

$$\sum_{i=1}^{n^j} w_i^j \geqslant T^j - \sum_{i=1}^{n^j} \blacksquare_i^j, \quad w_i^j \stackrel{\text{def}}{=} \begin{cases} P_i^j - \blacksquare_i^j, \text{ если } \blacksquare\text{BP}\left(\blacksquare_i^j, \blacksquare_i^j, \blacksquare_i^j, \blacksquare_i^j\right) \in \mathbf{S}_i^j; \\ 0, \text{ иначе.} \end{cases} \tag{2'}$$

Вводя новые коэффициенты

$$T_*^j \stackrel{\text{def}}{=} T^j - \sum_{i=1}^{n^j} \blacksquare_i^j, \quad V_i^j \stackrel{\text{def}}{=} P_i^j - \blacksquare_i^j, \tag{5}$$

и размещая координаты $\left(\blacksquare_i^j, \blacksquare_i^j, \blacksquare_i^j, \blacksquare_i^j\right)$ в четырех байтах числа F_i^j типа `int`, получим условие прохождения стадии, не имеющее весов $\blacksquare_i^j$:

$$\sum_{i=1}^{n^j} V_i^j \cdot \left[\blacksquare\text{BP}\left(F_i^j\right) \in \mathbf{S}_i^j\right] \geqslant T_*^j, \quad \left[\blacksquare\text{BP}\left(F_i^j\right) \in \mathbf{S}_i^j\right] \stackrel{\text{def}}{=} \begin{cases} 1, \text{ если } \blacksquare\text{BP}\left(F_i^j\right) \in \mathbf{S}_i^j; \\ 0, \text{ иначе.} \end{cases} \tag{6}$$

В итоге исходный набор коэффициентов (3) сократился до

$$\blacksquare_{\text{EP}} \stackrel{\text{def}}{=} \left(\left(F_i^j, \mathbf{S}_i^j, V_i^j\right)_{i=1}^{n^j}, T_*^j\right)_{j=1}^{m}. \tag{7}$$

Наибольший объем памяти (по 256 бит) здесь занимают множества $\mathbf{S}_i^j$ «желательных» значений признаков $\blacksquare\text{BP}\left(F_i^j\right)$. Эти множества можно сжать, однако это приведет к значительному замедлению программы, поэтому они были оставлены как есть. Коэффициенты V_i^j и T_*^j определены с точностью до произвольного сомножителя. Если их привести к достаточно большому диапазону (например, $[0; 1{\cdot}10^5]$), и округлить, то результаты классификации не изменятся. Поэтому алгоритм (6) может быть полностью целочисленным.

Так как $\left[\blacksquare\text{BP}\left(F_i^j\right) \in \mathbf{S}_i^j\right] \in \{0; 1\}$, можно избавиться от нежелательного умножения в выражении (6). В данном случае умножение можно заменить унарным минусом и битовым «И», так как число -1 в двоично-дополнительной записи имеет все единичные биты:

$$\sum_{i=1}^{n^j} V_i^j \,\&\, \left(-\left[\blacksquare\text{BP}\left(F_i^j\right) \in \mathbf{S}_i^j\right]\right) \geqslant T_*^j. \tag{6'}$$

Подход, описанный в данном разделе, позволил сохранить данные классификатора в непрерывном участке памяти объемом 6,15 килобайта, что лишь на 10% больше объема коэффициентов (7). Это означает, что разработанная структура данных имеет всего 10% накладных расходов. Напомним, что OpenCV требовалось 300 килобайт для хранения классификатора.

Следует также упомянуть особенности сканирования изображения окном детекции. В OpenCV несколько первых слоев сканируются с шагом 2 пикселя в горизонтальном и вертикальном направлениях (четверть возможных положений окна). В то же время остальные слои сканируются плотно — с шагом один пиксель. Такой подход, по-видимому, сделан с целью повышения быстродействия, однако он приводит к снижению качества детекции

маленьких лиц (имеющих размер от 24-х до 48-ми пикселей), а также к проблемам с группировкой детекций. Из-за различной плотности сканирования на слоях низкого разрешения группы детекций имеют в среднем в 4 раза больше элементов, чем на слоях высокого разрешения. Поэтому не удается подобрать «хорошее» значение для параметра, соответствующего минимальному количеству детекций в группе: мы получаем либо много ложноположительных детекций для крупных лиц, либо много пропусков маленьких лиц.

В нашей реализации окно детекции на всех слоях пирамиды проходит по пикселям в соответствии с узором «шахматная доска» (левый верхний угол окна проходит по половине пикселей). В итоге на предмет наличия лиц проверяется на 40% больше окон, чем проверяется в OpenCV. Это приводит к лучшим результатам детекции (как за счет большего числа окон, так и за счет равномерного их распределения по слоям).

5. Результаты

Несмотря на то, что код, написанный для сопроцессора Epiphany, может быть практически без изменений запущен на центральном процессоре, для сравнения с OpenCV имело смысл сделать еще один, упрощенный вариант кода, предназначенный только для центрального процессора. Этот упрощенный вариант не имеет очереди заданий, разбиения изображения на тайлы и пересылки данных. При запуске на центральном процессоре упрощенный код работает немного быстрее полного кода, и потребляет меньше памяти.

Были проведены измерения на изображениях, имеющих различное разрешение и лица разного размера. Наблюдаемые во всех случаях результаты и соотношения были примерно одинаковыми, поэтому далее рассматривается только одно изображение.

На рис. 3 показаны результаты детекции лиц на изображении размерами 1600×1065 пикселей. Результаты OpenCV показаны квадратами, результаты нашей реализации показаны окружностями. Наша реализация нашла больше лиц за счет упомянутого выше более плотного сканирования слоев пирамиды высокого разрешения. При детекции крупных лиц (48 пикселей и более) наши результаты практически не отличаются от OpenCV.

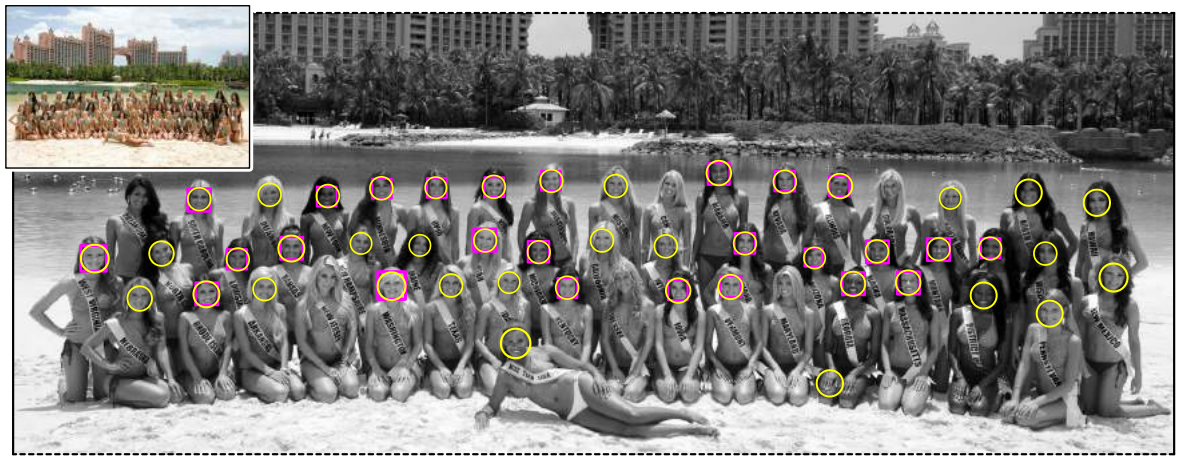


Рис. 3. Фрагмент изображения с результатами детекции лиц (исходное изображение показано слева). Квадраты — результат OpenCV; окружности — результат нашей программы

Сравним количество потребляемой памяти у обеих реализаций. Профилирование показало, что наша реализация при обработке указанного изображения потребовала 3,8 мегабайта дополнительной памяти, в то же время OpenCV потребовала более 8,5 мегабайт.

Приходим к выводу, что представленная реализация потребляет примерно в 2,2 раза меньше оперативной памяти, чем популярная библиотека OpenCV.

Время обработки изображения у OpenCV и у нашей реализации оказалось примерно одинаковым — 0,2 секунды на процессоре Intel Core I7 (2 ядра, 4 потока исполнения, тактовая частота 2,4 ГГц). Учитывая, что наш детектор проверяет на 40% больше окон, мы можем заключить, что наша реализация примерно на 40% быстрее, чем OpenCV (построение пирамиды составляет малую часть от времени детекции), или же обеспечивает лучшую точность детекции при том же времени работы.

Перейдем к экспериментам с Epiphany. Мы работали с прототипной системой, содержащей центральный процессор AMD E-350 (2 ядра, тактовая частота 800 МГц), и раннюю модификацию сопроцессора Epiphany, имеющую 16 ядер и тактовую частоту 400 МГц.

Эксперимент 1: детекция на одном ядре сопроцессора. Построение пирамиды на центральном процессоре потребовало 0,038 секунды. Время детекции лиц, измеренное на сопроцессоре: 16,08 секунды (сюда не включено время синхронизации и пересылки данных). Общее время работы алгоритма, измеренное на центральном процессоре: 16,46 секунды.

Эксперимент 2: детекция 16-ти ядрах сопроцессора. Построение пирамиды на центральном процессоре заняло 0,039 секунды; время детекции лиц, измеренное на ядрах сопроцессора (без учета времени синхронизации и пересылки данных): $1{,}0 \pm 0{,}01$ секунды. Общее время детекции, измеренное на центральном процессоре: 1,08 секунды.

Эксперимент 3: детекция на одном ядре центрального процессора. Использован почти тот же код, что и на сопроцессоре, все вычисления ведутся в памяти центрального процессора. Время построения пирамиды: 0,032 секунды. Общее время детекции лиц: 1,34 секунды.

Из представленных экспериментов можно сделать следующие выводы:

— Синхронизация и передача данных между общей памятью и сопроцессором занимает 2,3% времени в первом эксперименте, и 7,4% времени во втором эксперименте. Рост процентного соотношения накладных расходов закономерен, так как объем передаваемых данных тот же, а общее время работы снижается.

— Эффективность работы кода, запущенного на 16-ти ядрах, равна 95,3%. Это высокий показатель, означающий целесообразность использования всех 16-ти ядер сопроцессора для решения данной задачи.

— Из данных третьего эксперимента следует, что вычислительная сложность построения пирамиды составляет около 2,4% от сложности всего алгоритма, поэтому вынос этого кода на центральный процессор не влияет существенно на общее время работы.

— Из второго и третьего экспериментов не следует делать вывод о нецелесообразности использования сопроцессора, так как, во-первых, в нашем распоряжении был сопроцессор с заниженной тактовой частотой, и, во-вторых, процессор AMD E-350 предназначен для ноутбуков, а Epiphany — для мобильных телефонов. Для адекватного сравнения нужно учесть разницу тактовых частот, а также площадь кристалла и энергопотребление. Получаем, что ядро Epiphany с тактовой частотой 800 МГц примерно в 6 раз медленнее на данной задаче, чем ядро AMD E-350. Это хороший показатель, учитывая, что ядро Epiphany имеет площадь 0,3 мм^2 и энергопотребление 14 мВт, а ядро AMD E-350 имеет площадь 30 мм^2 и энергопотребление 8 Вт.

Заключение

Микропроцессор Epiphany имеет архитектуру, пригодную для решения широкого класса задач. Он оптимизирован для снижения энергопотребления, поэтому имеет перспективы применения в мобильных телефонах и планшетных компьютерах.

Наличие сквозной адресации позволяет реализовывать параллельные алгоритмы с различными схемами передачи данных, находясь в рамках модели программирования языка Си (производитель заявляет, что C++ уже тоже поддерживается).

Кроме указанных преимуществ, микропроцессор имеет недостатки, прежде всего связанные с малым объемом памяти на каждом его ядре, и значительным падением быстродействия при использовании в программе арифметики, не поддерживаемой аппаратно. Эти особенности делают невозможным эффективное использование кодов, разработанных для других процессоров; требуется тщательно продумывать структуры данных и их расположение в памяти, и соответственно переписывать код.

Несмотря на эти сложности, нам удалось разработать и реализовать эффективный алгоритм детекции лиц, совместимый (по данным классификатора) с алгоритмом детекции лиц из распространенной библиотеки OpenCV. Продемонстрирована целесообразность использования параллельной реализации (ускорение решения задачи) и ее высокая эффективность при работе на 16-ти ядрах микропроцессора Epiphany.

Работа выполнена при поддержке Российского научного фонда, грант 14-11-00659.

Литература

1. Papamarcos, M.S. A Low-Overhead Coherence Solution for Multiprocessors with Private Cache Memories / M.S. Papamarcos, J.H. Patel // Proceedings of the 11th annual international symposium on Computer architecture ISCA'84. — 1984. — P. 348–354.

2. Archibald, J. Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model / J. Archibald, J. Baer // ACM Trans. on Computer Systems. — 1986. — Vol. 4, No. 4. — P. 273–298.

3. Baumann, A. The Multikernel: A New OS Architecture for Scalable Multicore Systems / A. Baumann, P. Barham, P.-E. Dagand, T. Harris, R. Isaacs, S. Peter, T. Roscoe, A. Schüpbach, A. Singhania // Proceedings of the 22nd ACM Symposium on OS Principles (Big Sky, MT, USA). — 2009. — P. 29–44.

4. Face Detection using the Epiphany Multicore Processor / URL: `http://www.adapteva.com/white-papers/face-detection-using-the-epiphany-multicore-processor/` (дата обращения: 13.02.2014).

5. Parallela – Supercomputing for Everyone / URL: `http://www.parallella.org/` (дата обращения: 13.02.2014).

6. OpenCV / URL: `http://opencv.org/` (дата обращения: 13.02.2014).

7. Abu-Mostafa, Y.S. Learning from Data / Y.S. Abu-Mostafa, M. Magdon-Ismail, H.-T. Lin. — AMLBook, 2012. — 213 p.

8. Ojala, T. Performance Evaluation of Texture Measures with Classification Based on Kullback Discrimination of Distributions / T. Ojala, M. Pietikäinen, D. Harwood // Proceedings of the

12th IAPR International Conference on Pattern Recognition (ICPR 1994). — 1994 — Vol. 1. — P. 582–585.

9. Viola, P. Rapid Object Detection Using a Boosted Cascade of Simple Features / P. Viola, M. Jones // Computer Vision and Pattern Recognition. — 2001. — Vol. 1. — P. 511–518.
10. Mitchell, D.P. Reconstruction Filters in Computer-Graphics / D.P. Mitchell, A.N. Netravali // ACM SIGGRAPH International Conference on Computer Graphics and Interactive Techniques. — 1988. — Vol. 22, No. 4. — P. 221–228.
11. Crow, F.C. Summed-Area Tables for Texture Mapping / F.C. Crow // Proceedings of the 11th annual conference on Computer graphics and interactive techniques. — 1984. — P. 207–212.

Сухинов Антон Александрович, к.ф. м.н., научный сотрудник, Сколковский институт науки и технологий (Сколково, Российская Федерация), soukhinov@gmail.com.

Остроброд Георгий Борисович, программист ООО «СиВижинЛаб» (Таганрог, Российская Федерация), wdf.gost@gmail.com.

Поступила в редакцию 4 августа 2014 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2014, vol. 3, no. 3, pp. 5–19

EFFICIENT FACE DETECTION ON EPIPHANY MULTICORE PROCESSOR

A.A. Sukhinov, Skolkovo Institute of Science and Technology (Skolkovo, Russia)**,**
G.B. Ostrobrod, CVisionLab (Taganrog, Russia)

It is studied the possibility of usage of energy-efficient Epiphany microprocessor for solving actual applied problem of face detection at still image. The microprocessor is a multicore system with distributed memory, implemented in a single chip. Due to small die area the microprocessor has significant hardware limitations (in particular it has only 32 kilobytes of memory per core) which limit the range of usable algorithms and complicate their software implementation. Common face-detection algorithm based on local binary patterns (LBP) and cascading classifier was adapted for parallel implementation. It is shown that Epiphany microprocessor having 16 cores can outperform single-core CPU of personal computer having the same clock rate by a factor of 2.5, while consuming only 0.5 watts of electric power.

Keywords: face detection, local binary patterns, parallel data processing, specialized processors, distributed memory.

References

1. Papamarcos M.S., Patel J.H. A Low-Overhead Coherence Solution for Multiprocessors with Private Cache Memories // Proceedings of the 11th annual international symposium on Computer architecture ISCA'84. 1984. P. 348–354.
2. Archibald J., Baer J. Cache Coherence Protocols: Evaluation Using a Multiprocessor Simulation Model // ACM Trans. on Computer Systems. 1986. Vol. 4, No. 4. P. 273–298.

3. Baumann A., Barham P., Dagand P.-E., Harris T., Isaacs R., Peter S., Roscoe T., Schüpbach A., Singhania A. The Multikernel: A New OS Architecture for Scalable Multicore Systems // Proceedings of the 22nd ACM Symposium on OS Principles (Big Sky, MT, USA). 2009. P. 29–44.

4. Face Detection using the Epiphany Multicore Processor. URL: `http://www.adapteva.com/white-papers/face-detection-using-the-epiphany-multicore-processor/` (accessed: 13.02.2014).

5. Parallela – Supercomputing for Everyone. URL: `http://www.parallella.org/` (accessed: 13.02.2014).

6. OpenCV. URL: `http://opencv.org/` (accessed: 13.02.2014).

7. Abu-Mostafa Y.S., Magdon-Ismail M., Lin H.-T. Learning from Data. AMLBook, 2012. 213 p.

8. Ojala T., Pietikäinen M., Harwood D. Performance Evaluation of Texture Measures with Classification Based on Kullback Discrimination of Distributions // Proceedings of the 12th IAPR International Conference on Pattern Recognition (ICPR 1994). 1994. Vol. 1. P. 582–585.

9. Viola P., Jones M. Rapid Object Detection Using a Boosted Cascade of Simple Features // Computer Vision and Pattern Recognition. 2001. Vol. 1. P. 511–518.

10. Mitchell D.P., Netravali A.N. Reconstruction Filters in Computer-Graphics // ACM SIGGRAPH International Conference on Computer Graphics and Interactive Techniques. 1988. Vol. 22, No. 4. P. 221–228.

11. Crow F.C. Summed-Area Tables for Texture Mapping // Proceedings of the 11th annual conference on Computer graphics and interactive techniques. 1984. P. 207–212.

Received 4 August 2014.

УДК 004.052.3

ОБЕСПЕЧЕНИЕ ОТКАЗОУСТОЙЧИВОСТИ ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ С ПОМОЩЬЮ ЛОКАЛЬНЫХ КОНТРОЛЬНЫХ ТОЧЕК[1]

А.А. Бондаренко, М.В. Якобовский

Рассматриваются вопросы, связанные с проведением расчетов в распределенных вычислительных системах, компоненты которых подвержены отказам. В работе приводятся: определения системы, сбоя, ошибки, отказа и модели сбоя; наиболее важные результаты исследований отказов в параллельных вычислительных системах, в том числе с большими группами дисков; основные существующие методы восстановления и распространенные программные реализации обеспечения отказоустойчивости. Развивается подход обеспечения отказоустойчивости на уровне пользователя. Данный подход требует непосредственного участия разработчика прикладной программы в реализации метода обеспечения отказоустойчивости, в частности в формировании контрольных точек и процедур восстановления. Предложена схема сохранения в памяти вычислительных узлов данных прикладной программы, формирующих согласованную глобальную контрольную точку. В её рамках осуществляется дублирование локальных контрольных точек, что позволяет восстановить вычислительный процесс, если число отказов не превосходит допустимого для данной схемы уровня. Она может быть использована в различных протоколах восстановления и их модификациях.

Ключевые слова: параллельные вычисления, отказоустойчивость, контрольные точки, MPI.

Введение

Современные суперкомпьютеры состоят из десятков тысяч узлов, каждый из которых оснащен процессорами и, как правило, различными ускорителями. Увеличение количества компонентов системы, без существенного изменения элементной базы или технологии производства, естественным образом ведет к увеличению вероятности отказа некоторых компонентов системы [1, 2], т.е. приводит к тому, что время наработки на отказ вычислительной системы становится мало и сравнимо со временем проведения сеанса расчета. Оценки исследователей в суперкомпьютерной области показывают, что время наработки на отказ может составлять всего несколько часов [3].

В области аппаратного обеспечения активно разрабатываются пути уменьшения вероятности отказа компонент вычислительных систем [4]. Однако невозможно создать стабильное устройство, в особенности такое сложное как память, процессор или коммуникационное оборудование, поэтому важным является разработка программных средств обеспечения отказоустойчивости. С этой точки зрения существенное значение имеют методы восстановления, основанные на контрольных точках. Подобные методы предполагают периодическое сохранение состояния системы в централизованное устройство хранения, что позволяет при выходе из строя части вычислительного поля продолжить расчет с последней сохранённой согласованной глобальной контрольной точки [5-7].

[1] Статья рекомендована к публикации программным комитетом XIII Всероссийской научной конференции «Высокопроизводительные параллельные вычисления на кластерных системах» 2013г.

Главной проблемой масштабируемости технологии создания контрольных точек для современных вычислительных систем является ограниченная пропускная способность централизованного устройства хранения. Для широко распространенного координированного протокола характерной является одновременная запись локальных контрольных точек, приводящая к повышенной загруженности узлов ввода/вывода и централизованного устройства хранения. Некоторыми исследователями [2] отмечается, что время сохранения согласованной глобальной контрольной точки для систем Терафлопсного и Петафлопсного уровня составляет порядка 20-30 минут. Фактически, в перспективе, с ростом числа компонент системы и с уменьшением времени наработки на отказ всей системы, сама возможность успешного сохранения согласованной глобальной контрольной точки окажется под вопросом [2, 8].

При проведении высокопроизводительных вычислений представляется важным решение следующей задачи: разработать принципы сохранения контрольных точек за время, меньшее характерной продолжительности безотказной работы системы, и алгоритмы, обеспечивающие, в случае отказа части оборудования, быстрое автоматическое возобновление расчета на работоспособной части вычислительного поля. В данной работе, на уровне пользователя [1], развивается подход обеспечения отказоустойчивости высокопроизводительных вычислений, требующих регулярных обменов данными между процессами. Этот подход позволяет сократить объем контрольных точек ценою усложнения программного кода, поскольку предполагает непосредственное участие разработчика прикладной программы в реализации метода обеспечения отказоустойчивости (в формировании контрольных точек и процедур автоматического восстановления). В работе представлена схема сохранения локальных контрольных точек в память вычислительных узлов (оперативная, дисковая и т.п.), к которым имеют доступ один или группа MPI-процессов. Данная схема сохранения позволяет произвести восстановление процесса вычислений при возникновении нескольких отказов в MPI-процессах, вызванных отказами в вычислительных узлах.

Первый раздел статьи посвящен описанию отказов в вычислительных системах. В разделе 2 приведены сведения о методах и средствах обеспечения отказоустойчивости в распределенных системах. В последнем разделе изложена схема сохранения локальных контрольных точек.

1. Отказы в распределенных системах

Прежде чем говорить об отказоустойчивости, приведем основные определения и рассмотрим характеристики отказов в распределенных системах.

1.1. Сбои, ошибки и отказы

Приведем описание понятий «система», «сбой», «ошибка» и «отказ» согласно работам [9, 10], принятыми в области гарантированных вычислений представителями научной и технической среды.

Система является объектом, который взаимодействует с другими объектами, то есть другими системами. В роли систем могут выступать, в том числе, аппаратное обеспечение, программное обеспечение, люди и физический мир с его природными явлениями.

Для каждой системы есть другие системы, которые являются внешними по отношению к данной системе, то есть образуют окружающую среду. Система состоит из набора компонент, каждая из которых, в свою очередь, сама является системой, со своей внутренней структурой. Системы, входящие во множество взаимодействующих систем уровня n, являются подсистемами для системы уровня $n+1$. Каждая система уровня n состоит из множества подсистем уровня $n-1$, которые в свою очередь состоят из подсистем уровня $n-2$ и т.д. Эта иерархия систем и подсистем продолжается до уровня, для которого либо не возможна, либо не имеет смысл дальнейшая детализация на подсистемы. Подсистемы на этом последнем уровне называются атомарными компонентами, а сам уровень определяется в зависимости от рассматриваемой задачи.

В работе рассматриваются компьютерные и коммуникационные системы. Данные системы являются искусственными, их функции определяются соответствующими спецификациями. Поведением системы является то, что система делает, чтобы реализовать свою функцию. Корректным называется поведение, обеспечивающее реализацию функции системы.

Со временем состояние некоторых компонент системы может отклониться от состояний, соответствующих корректному поведению системы. Это измененное состояние компонент системы называется ошибкой (error). Наличие ошибки не означает, что поведение системы не может реализовать свою функцию. Однако, может наступить такое событие, когда осуществляемое поведение отклоняется от корректного, то есть система не реализует ожидаемую функцию. В этом случае говорят, что наступил отказ системы (system failure). Пусть возникла ошибка, тогда существует последовательность действий, выполнение которых приведет к отказу системы, если не будут предприняты корректирующие меры. Установленная или предполагаемая, причина ошибки называется сбоем (fault). Таким образом, сбой вызывает ошибку, которая может и не привести к отказу системы.

1.2. Модели сбоев

Такая абстракция как «модель сбоя», имеет существенное значение для разработки методов обеспечения отказоустойчивости. Приведем описание модели сбоев согласно работе [10].

Пользователь рассматривает конкретный уровень системы – уровень вентилей и логических схем, уровень функциональных блоков, уровень микросхем и т.п. В большинстве случаев пользователя не интересуют отказы на нижних уровнях системы. Для него важен факт наличия отказа на том уровне, на котором он рассматривает данную систему, как следствие сбоя на одном из нижних уровней. Один из способов удовлетворить интерес пользователя заключается в описании, для более высокого уровня, эффекта сбоя, произошедшего на нижнем уровне. Абстракция, заключающаяся в определение эффекта сбоя для конкретного уровня, называется моделью сбоя (fault model). Модель сбоя фактически описывает нарушения в функционировании системы, то есть отказы для конкретного уровня, к которым могут привести различные сбои на нижних уровнях. Обычно многие сбои на нижних уровнях могут приводить к одинаковым нарушениям функционирования системы, то есть быть представлены одной моделью сбоя на более высоком уровне. Пользователь выбирает модель сбоев так, чтобы она описывала наиболее важные сбои на нижних уровнях. Если такая модель существует, то пользова-

телю, на интересующем его уровне системы, необходимо учитывать отказы только в выбранной модели сбоев.

Модели сбоев, используемые в настоящее время, значительно варьируются: от очень подробных (на уровне транзисторов), до функционального уровня устройств. Наиболее часто в программных средствах обеспечения отказоустойчивости в качестве модели сбоя на функциональном уровне принимается либо модель «поломка» (crash failure), либо модель «аварийная остановка» (fail-stop failure) устройства вычислительной системы. Здесь под «устройством вычислительной системы» понимается процессор, ускоритель, коммутатор, модуль оперативной памяти, жесткий диск и т.п. Отличие между этими моделями заключается в том, что для модели «поломка» информация о неработоспособности устройства не передается другим компонентам системы, а для модели «аварийная остановка» информация распространяется среди соответствующих компонентов системы.

1.3. Статистика отказов

Создание новых суперкомпьютеров сопряжено с необходимостью решения ряда задач [8], среди которых важное место занимает возможности выполнения расчетов, несмотря на отказ ряда задействованных в расчете вычислительных узлов. Для выработки лучшей стратегии обеспечения отказоустойчивости необходимо иметь набор экспериментальных данных позволяющих делать выводы о статистике отказов в определенных вычислительных системах. Одним из шагов к появлению данных исследований является создание базы отказов. Так LANL (Los Alamos National Laboratory) и NERSC (National Energy Research Scientific Computing Center) опубликовали накопленные данные о сбоях в используемых ими системах [12, 13].

В работе [14] проведен анализ отказов, описанных в 10 наборах данных сроком сбора от 5 до 18 месяцев каждый, полученных с 8 вычислительных систем, на которых решались различные задачи. Результаты работы [14] расходятся с результатами, представленными в других источниках, что дает еще одно подтверждение необходимости создания обширной общедоступной базы отказов. Существование подобной базы отказов даст возможность для каждого исследователя сравнить свои результаты и методики оценки характеристик отказов с результатами других исследований, проведенных на одних и тех же наборах данных.

Существуют различные показатели характеризующие надёжность вычислительных систем [6, 7]. Одними из существенных для анализа отказов являются MTBF и MTBI – среднее время между отказами и между прерываниями. Значения этих показателей для некоторых распределенных вычислительных систем представлены в работе [3]. Среднее время между отказами/прерываниями составляет от 6.5 до 40 часов, а наиболее частыми отказами были: отказы в процессоре и устройствах хранения.

При анализе отказов важным является определение процентного соотношения между категориями отказов. В работе [15] выделили следующие категории отказов: аппаратные, программные, сетевые, связанные с человеческим фактором, связанные с внешним обеспечением работы вычислительной системы (перебои электричества). В результате исследований отказов зафиксированных в 1996-2005 гг. на группе вычислительных систем LANL было выявлено, что аппаратные отказы составляют более 60% от всех зарегистрированных отказов.

В работе [16] проведено исследование, охватывающее в общей сложности более 100 000 жестких дисков, как минимум от четырех различных поставщиков. Наборы данных об отказах собирались от 1 месяца до 5 лет каждый и получены на нескольких крупных высокопроизводительных вычислительных системах. В [16] получены следующие результаты. Ежегодно заменяется более 1% дисков (обычно от 2 до 4% вплоть до 13%). Были найдены свидетельства, что интенсивность отказов в дисках не является постоянной от времени эксплуатации. Наблюдалось раннее начало деградации дисков; доля замененных дисков, постоянно росла с возрастом, хотя часто предполагается, что этот эффект не проявляется до окончания номинального срока службы 5 лет. В итоге авторы работы [16] предположили, что правила замены дисков должны отличаться от текущего описания в спецификациях производителей дисков.

Из приведенных результатов исследований в области отказов больших групп дисков следует, что для распределенных систем Петафлопсного уровня и выше вероятность отказа узла значительна из-за высокой вероятности отказа его компонент, таких как диски, процессоры, память и другие.

2. Обеспечение отказоустойчивости для распределенных систем

Далее рассматриваются теоретические методы восстановления и распространенные программные решения в области обеспечения отказоустойчивости при проведении высокопроизводительных вычислений, характеризующихся выполнением больших объемов вычислений на большом количестве узлов в течение длительного времени (порядка десятков и более часов).

2.1. Методы восстановления

Под отказоустойчивостью понимают способность системы выполнять работу даже при наличии отказов [9]. Рассмотрим методы обеспечения отказоустойчивости в распределенных вычислительных системах, обеспечивающие продолжение расчета после отказа части вычислительных узлов.

Методы восстановления (rollback-recovery) [5 – 7] работы системы после возникновения отказов принято разделять на методы прямого (forward recovery) и обратного (backward recovery) восстановления. Первые восстанавливают систему в безошибочное состояние на основе текущих данных и по результатам анализа отказа, без обращения к предыдущим состояниям системы. Методы прямого восстановления основаны на специальных алгоритмах, например, широко известны стабилизирующиеся алгоритмы [11]. Методы обратного восстановления также заключаются в восстановлении системы в корректное состояние. Однако процесс восстановления использует информацию о предыдущем полном или частичном состоянии системы, запись о котором называется контрольной точкой. Протоколами восстановления, основанными на контрольных точках, являются: некоординированный протокол (uncoordinated checkpointing); координированный протокол (coordinated checkpointing); протокол вынужденных сообщений (communication-induced checkpointing).

На данный момент наибольшее распространение получил координированный прокол. Однако для него создание контрольных точек связано с большой нагрузкой на сеть, возникающей из-за интенсивного использования узлов ввода/вывода системы для записи на централизованное устройство хранения. Среднее время сохранения контрольных точек может быть весьма значительным и для некоторых систем составляет 20-30 минут [2]. С целью уменьшения затрат на частое создание контрольных точек были разработаны методы, которые включают в себя дополнительные механизмы сохранения, позволяющие уменьшить нагрузку на коммуникационную сеть. Приведем описание трех групп подобных методов.

В первую группу входят методы, основанные на некоординированном протоколе и на механизмах создания журналов передачи сообщений (log-based rollback-recovery). Создание журналов передачи сообщений заключается в том, что на стороне посылающего сохраняется содержание сообщения, информация о получателе и о моменте получения данного сообщения. Использование некоординированного протокола позволяет снизить нагрузку на сеть, так как запись локальных контрольных точек может осуществляться в произвольное время. Во время восстановления, используя журналы передачи сообщений, необходимо произвести дополнительную работу на поиск согласованного состояния системы.

Методы, второй группы [17, 18] основаны на сохранении изменений между двумя последовательными контрольными точками, к ним относятся инкрементный (incremental checkpointing) и дифференциальный (differential checkpointing) методы. Первый предусматривает сохранение в текущей промежуточной контрольной точке изменений относительно предыдущей. Второй сохраняет все изменения относительно базовой контрольной точки. Как показано в работе [18] можно дополнительно использовать методы сжатия для контрольных точек. Восстановление происходит из последней контрольной точки с учетом изменений сохранных в промежуточные контрольные точки.

В третью группу входят методы [15, 19, 20], которые предполагают сохранение контрольных точек вычислительных узлов в память вычислительных узлов. В [15, 19] вводится понятие многоуровневой схемы сохранения контрольных точек, к ним относятся гибридная (hybrid checkpointing) [15] и двухуровневая (two level recovery scheme) [20] схемы. В работе [15] сохранение контрольных точек основано на инкрементном методе с механизмом скрытого копирования. В механизме скрытого копирования после сохранения контрольных точек в память узлов запускается и продолжение расчета, и процесс сохранения согласованной глобальной контрольной точки на централизованное устройство хранения из памяти вычислительных узлов. В работе [19] предлагается использовать метод создания журналов передачи сообщений для сохранения в память вычислительных узлов и координированный протокол для сохранения на централизованное устройство хранения. В [15, 19] при восстановлении используется последняя согласованная глобальная контрольная точка с централизованного устройства хранения и данные из памяти вычислительных узлов. В работе [20] сохранение контрольных точек осуществляется исключительно в память вычислительных узлов (diskless), при этом кроме узлов осуществляющих расчет, необходимы дополнительные узлы для хранения контрольной суммы. В случае отказа память всех узлов используется для реконструкции образа памяти отказавших узлов, затем осуществляется запуск продолжения вычислений.

2.2. Распространенные программные решения

В России наиболее известными программными решениями, в которых реализованы подходы к обеспечению отказоустойчивости, являются следующие разработки: система метакомпьютинга X-Com [21], разработанная сотрудниками НИВЦ МГУ; программный комплекс «Пирамида» [22] разрабатываемый МСЦ РАН совместно с НИИ «Квант»; программная среда «OpenTS»[23], разработанная в Институте программных систем РАН. Данные комплексы обеспечивают работоспособность системы при выходе из строя некоторых вычислительных узлов. Однако X-Com и Пирамида осуществляют организацию вычислительного параллельного процесса исключительно для задач, в которых отсутствуют информационные обмены между обрабатывающими процессами. «OpenTS» – специализированная среда программирования с поддержкой автоматического динамического распараллеливания программ, на основе функционального программирования.

Система управления HTCondor [24] и специальный модуль BLCR для ядра Линукса [25] предоставляют механизмы для сохранения полного контекста процесса и его перезапуска из сохраненного контекста. Эти программные средства являются автоматическими и не позволяют контролировать процедуру сохранения и восстановления. Разрабатываемые в рамках проектов HTCondor и BLCR средства обеспечения отказоустойчивости вычислений не поддерживают в настоящее время отказоустойчивость при выполнении программ, требующих регулярных обменов данными между процессами (например, программ основанных на принципах Domain Decomposition).

В распространенных открытых библиотеках интерфейсов передачи сообщений MPICH [26], MVAPICH [27], OpenMPI [28] реализованы возможности создания контрольных точек по координированному протоколу восстановления на основе специального модуля BLCR для ядра Linux. Таким образом, эти программные средства позволяют обеспечить отказоустойчивость вычислений, требующих регулярных обменов данными между процессами, но сталкиваются с проблемой нагрузки на сеть при сохранении контрольных точек.

Подробный список зарубежных программных решений обеспечивающих отказоустойчивость можно найти в [29].

2.3. Стандарт MPI и расширение ULFM

Стандарт MPI и его программные реализации играют существенную роль в развитии распределенных вычислений. Отметим, что прикладной программист не работает непосредственно с отказами в вычислительных узлах, он сталкивается с их следствием, то есть с отказами в MPI-процессах. Однако в стандарте MPI 3.0 [30] отсутствует описание каких-либо функций связанных с обеспечением отказоустойчивости в вычислительных системах. В работе [1] представлено расширение ULFM [31] стандарта MPI, которое предназначено для решения задач идентификации сбоев в процессе вычислений, восстановления связи между MPI-процессами, исключения из вычислительного поля отказавших MPI-процессов. В качестве модели сбоя принята модель «поломка» устройства вычислительной системы. Данное расширение предоставляет пользователю возможность реализовать в программном приложении различные техники отказоустойчивости, в том числе варьировать объем и содержание контрольных точек. Появление ULMF в стан-

дарте MPI планируется с версии MPI 3.1. Программные реализации MPI 3.1 окажут существенное влияние на возможность проведения расчетов в распределенных системах с частыми отказами.

3. Обеспечение отказоустойчивости на уровне пользователя с помощью локальных контрольных точек

В работе [15] ссылаются на прогноз работы [32]: «... более чем 83% отказов в системах Петафлопсного уровня могут быть восстановлены с использованием локальных контрольных точек, в то время как оставшиеся 17%, включающие сложные ошибки или потерю вычислительного узла, требуют использования доступной согласованной глобальной контрольной точки». Проблема накладных расходов в сети при сохранении контрольных точек и прогноз, приведенный в [32], говорит о необходимости использования памяти вычислительного узла для хранения согласованной глобальной контрольной точки. Реализация методов восстановления на уровне пользователя позволяет значительно сократить объем контрольных точек, по сравнению со стандартным методом [5]. Однако в текущем стандарте MPI 3.0 отсутствуют механизмы позволяющие работать с отказами на уровне пользователя. Несмотря на это, наличие разрабатываемого расширения ULFM [31] для стандарта MPI и его предшественника FT-MPI [33] ведет к необходимости разработки методов автоматического восстановления на уровне пользователя.

3.1. Память вычислительных узлов

Пусть MPI-процессы сохраняют локальные контрольные точки в память своих вычислительных узлов. Тогда, при отказе хотя бы одного вычислительного узла, MPI-процессы, запущенные на других узлах, не смогут получить доступ к локальным контрольным точкам, расположенным в памяти отказавшего. Таким образом, восстановление процесса вычислений будет невозможно. Для выхода и такого положения следует обеспечить избыточность хранения локальных контрольных точек в системе за счет их дублирования в памяти различных вычислительных узлов. Соответствующая схема записи предлагается в следующем параграфе. В её рамках для каждого MPI-процесса определяются номера вычислительных узлов, в память которых должны быть сохранены копии локальной контрольной точки. Загруженность сети при использовании данной схемы сохранения, будет зависеть от объемов локальных контрольных точек.

3.2. Схема сохранения локальных контрольных точек

Для описания схемы записи локальных контрольных точек будем использовать следующие параметры: N – число узлов в системе; MNF – максимально допустимое число отказов в системе; глубина хранения SD (storage depth) – это количество итераций сохранения, в которые каждая локальная контрольная точка доступна для чтения; коэффициент дублирования DF (duplication factor) – это количество узлов хранящих копию данной локальной контрольной точки, отличных от данного вычислительного узла. Цель данной схемы сохранения состоит в максимизации минимального числа узлов, по-

вреждение которых приведёт к невозможности восстановления с последних SD итераций сохранения.

Предполагаем, что на один вычислительный узел приходится один MPI-процесс. Данное предположение не снижает общности последующих рассуждений. При запуске на одном узле нескольких MPI-процессов необходимо всем MPI-процессам с одного узла осуществлять запись и чтение копий локальных контрольных точек в память вычислительных узлов, определяемых формулой (1). В этом случае в системе будет доступна согласованная глобальная контрольная точка, если число отказов узлов не превосходит значение MNF.

Будем предполагать, что используется координированный протокол. В случае использования других методов, формула (2) может быть неверна.

Введем нумерацию итераций сохранения контрольных точек $k = 0,1,2,\ldots$. На одной итерации сохранения MPI-процесс с каждого i–ого вычислительного узла:

- сохраняет свою локальную контрольную точку в память этого узла;
- осуществляет запись DF копий своей локальной контрольной точки в память вычислительных узлов, определяемых формулой (1).

Введем параметр $j \in \{1, 2, \ldots, DF\}$ для обозначения номера передаваемой копии локальной контрольной точки с i–ого узла. Тогда на k-ой итерации сохранения, запись -ой копии с i–ого узла должна быть осуществлена в память вычислительного узла с номером, определяемым формулой:

$$receiver(i,j,k) = \left[i + j \cdot DF^{k \bmod SD} + k \bmod SD\right] \bmod N. \quad (1)$$

В данной схеме объем сохраняемой на каждом узле информации можно оценить величиной $V = V_0 \cdot SD \cdot (DF + 1)$, где V_0 - средний объем локальных контрольных точек.

Пусть в вычислительной системе достаточно много узлов $N \geq DF^{SD} + SD$ и число отказавших узлов не превосходит

$$MNF = (DF - 1) \cdot SD + 1. \quad (2)$$

Тогда, после первых SD итерации сохранения, в системе будут доступны локальные контрольные точки MPI-процессов со всех узлов одной из последних SD итераций сохранения, то есть согласованная глобальная контрольная точка. Таким образом, используя предлагаемую схему, можно восстановить процесс вычислений, если количество отказов в системе не превышает значения MNF.

3.3. Восстановление вычислений после возникновения отказа

Пусть все множество MPI-процессов делится на рабочие процессы, формирующие рабочее поле, и дополнительные процессы. Предполагаем, что в приложении осуществляется координированное сохранение необходимых пользователю данных по описанной выше схеме на вычислительные узлы рабочих процессов. В общем случае, можно осуществлять сохранение в память всех доступных узлов. После возникновения отказа и оповещения системы о нем должны быть выполнены действия по восстановлению:

1. MPI библиотеки и среды выполнения программы;
2. расчетного поля (перераспределение рабочих и дополнительных процессов);
3. промежуточных данных (для процессов в новом рабочем поле);
4. расчета на основе содержимого локальных контрольных точек доставленных на каждый из узлов нового рабочего поля.

Оповещение об отказе в системе, восстановление MPI библиотеки, среды выполнения программы и рабочее поля предполагается осуществлять с помощью стандартных средств, таких как ULFM для MPI. Восстановление расчета на основе данных доставленных на каждый из узлов нового рабочего поля должно осуществляться пользователем.

Код реализующий алгоритм поиска линии восстановления

```
1:  int receiver(int i, int j, int k, int DF, int SD, int N)
2:  {
3:    return (i+j*(int)pow((double)DF,k%SD)+ k%SD)%N;
4:  }
5:  bool is_element_of_Y(int rank, int *fault_nodes, int N_fault){
6:    for(int i = 0; i < N_fault; i++){
7:      if(rank == fault_nodes[i])
8:        return false;
9:    }
10:   return true;
11: }
    int accessibility(int n, int k, int *fault_nodes, int N_fault,int
12: DF, int SD, int N){
13:   int Receiver;
14:   for(int j = 1; j <= DF; j++){
15:     Receiver = receiver(fault_nodes[n], j, k, DF, SD, N);
16:     if(is_element_of_Y(Receiver, fault_nodes, N_fault))
17:       return Receiver;
18:   }
19:   return -1;
20: }
    int RLFFN( int k_end, int *fault_nodes, int N_fault, int
21: *recovery_nodes, int DF, int SD, int N){
22:   int number_of_saving;
23:   for (int k = k_end; k>(k_end-SD); k--){
24:     bool recovery_is_imposible = false;
25:     for(int n = 0; n < N_fault; n++ ){
          if((recovery_nodes[n] = accessibility ( n, k, fault_nodes,
26: N_fault, DF, SD, N))==-1){
27:         recovery_is_imposible = true;
28:         break;
29:       }
30:     }
31:     if(!recovery_is_imposible )
32:       return (number_of_saving = k);
33:   }
34:   return -1;
35: }
```

Перед описанием алгоритма восстановления промежуточных данных введем следующие уточнения. Введем обозначения: U – множество MPI-процессов запускаемого приложения, среди которых $U_{\text{раб}}$ – множество рабочих, $U_{\text{доп}}$ – множество дополнительных процессов. Допустим, что произошли отказы на всех процессах из множества X. Пусть X подмножество $U_{\text{раб}}$, что не ограничивает общность рассуждений, в противном случае удалим из X и $U_{\text{доп}}$ общие процессы. При $|X| \leq |U_{\text{доп}}|$ возможно создание нового расчетного поля $\widehat{U}_{\text{раб}}$ с помощью замены отказавших процессов на нормально функционирующие дополнительные процессы. Пусть Y – множество нормально функционирующих

процессов из начального расчетного поля $U_{\text{раб}}$ (до введения дополнительных процессов) ($Y = U_{\text{раб}} \backslash X$), Z – множество дополнительных процессов, вводимых в новое рабочее поле ($|Z| = |X|$, $\widehat{U}_{\text{раб}} = Y \cup Z$). В этом случае в памяти узлов, соответствующих процессам множества Z, будет отсутствовать информация о каких-либо контрольных точках. Восстановление возможно только из данных сохраненных в памяти узлов, соответствующих процессам множества Y.

Для восстановления промежуточных данных необходимо определить линию восстановления (последнюю согласованную глобальную контрольную точку [5]) и произвести копирование соответствующих локальных контрольных точек из памяти узлов, соответствующих процессам множества Y, в память узлов, соответствующих процессам множества $\widehat{U}_{\text{раб}}$. Предполагаем, что после восстановления MPI библиотеки, среды выполнения и расчетного поля верно следующие:

- номера рабочих процессов из Y совпадают для $U_{\text{раб}}$ и $\widehat{U}_{\text{раб}}$, то есть должны остаться прежними при замене отказавших процессов на нормально функционирующие дополнительные процессы;
- все процессы в новом расчетном поле должны знать параметры восстановления: глубину сохранения контрольной точки, коэффициент дублирования, номер последней итерации сохранения, число отказавших процессов и их номера.

Линия восстановления будет зависеть от того, какую именно согласованную глобальную контрольную точку можно извлечь из памяти узлов, соответствующих процессам множества Y. Для этого требуется:

- определить номер последней доступной согласованной глобальной контрольной точки;
- определить, для каждого процесса из множества $\widehat{U}_{\text{раб}}$, номер процесса из Y, имеющего доступ к соответствующей локальной контрольной точке.

Для линии восстановления каждый процесс из множества Y будет использовать локальную контрольную точку, записанную в память его вычислительного узла. Таким образом, линия восстановления будет определяться номером последней доступной согласованной глобальной контрольной точки и номерами процессов множества Y, имеющих доступ к соответствующим локальным контрольным точкам процессов множества Z.

Определение необходимых для линии восстановления параметров реализовано в функции RLFFN, в коде которой используются следующие переменные и функции:

- k_end – номер последней итерация сохранения;
- N_fault – число отказавших узлов;
- fault_nodes – массив номеров отказавших узлов в прошлом расчетном поле;
- recovery_nodes[i] – номер процесса из Y, который имеет доступ к локальной контрольной точке, необходимой процессу из Z, с номером fault_nodes[i], для линии восстановления;
- number_of_saving – номер последней согласованной глобальной контрольной точки для восстановления;
- bool is_element_of_Y(int rank, int *fault_nodes, int N_fault) – определяет принадлежность данного процесса с рангом rank множеству Y;

- int accessibility(int n, int k, int *fault_nodes, int N_fault,int DF, int SD, int N) – определяет доступность копии локальной контрольной точки процесса fault_nodes[n] от процессов из Y.

Таким образом, алгоритм восстановления промежуточных данных состоит в определении линии восстановления, т.е. запуска на всех MPI-процессах функции RLFFN, и в копировании недостающих локальных контрольных точек согласно линии восстановления. Копирование заключается в том, что процесс с номером recovery_nodes[i] записывает процессу с номером fault_nodes[i] соответствующую локальную контрольную точку, сохраненную на number_of_saving итерации сохранения. После осуществления копирования для всех процессов из Z должна следовать фаза восстановления расчета пользователем.

Заключение

На данный момент остается актуальным развитие методов автоматического восстановления программ, позволяющих снизить нагрузку на сеть во время сохранения контрольных точек. Одно из возможных решений заключается в предоставлении пользователю возможности реализации в его программе различных методов отказоустойчивости, в том числе варьирования объема, содержания и стратегии сохранения контрольных точек. Для реализации методов на уровне пользователя необходима поддержка механизмов работы с отказами, что связанно с внедрением расширения ULFM в стандарт MPI и его реализацией. Привлекательной является стратегия, основанная на использовании для хранения локальных контрольных точек памяти вычислительных узлов. В работе представлена схема сохранения локальных контрольных точек, в рамках которой осуществляется их дублирование, что позволяет восстановить вычислительный процесс, если число отказов не превосходит допустимого числа отказов для данной схемы. Схема может быть использована, в том числе на уровне пользователя, при разработке различных методов автоматического восстановления вычислений, требующих регулярных обменов данными между процессами.

Работа выполнена при поддержке Российского фонда фундаментальных исследований по гранту 13-01-12073 офи_м.

Литература

1. Bland, W. Post-failure recovery of MPI communication capability: Design and rationale / W. Bland, A. Bouteiller, T. Hérault, G. Bosilca, J. Dongarra // International Journal of High Performance Computing Applications. — 2013. — Vol. 27, No. 3. — P. 244–254.
2. Cappello, F. Fault tolerance in petascale/exascale systems: Current knowledge, challenges and research opportunities / Cappello F. // International Journal of High Performance Computing Applications. — 2009. — Vol. 23, No. 3. — P. 212–226.
3. Hsu, C.-H. A power-aware run-time system for high-performance computting / C.-H. Hsu, W.-C. Feng. // Proceedings of SC|05: The ACM/IEEE International Conference on High-Performance Computing, Networking, and Storage (Seattle, Washington USA November 12 – 18, 2005). — IEEE Press, 2005. — P. 1–9.

4. Sorin, D. Fault Tolerant Computer Architecture. Synthesis Lectures on Computer Architecture / D. Sorin — Morgan&Claypool, 2009. — 104 p.
5. Elnozahy, E.N. A Survey of Rollback-Recovery Protocols in Message-Passing Systems / E.N. Elnozahy, L. Alvisi, Y. Wang, D.B. Johnson // ACM Computing Surveys. — 2002. — Vol.34, No. 3 — P. 375–408.
6. Koren, I. Fault-Tolerant Systems / I. Koren, C. M. Krishna — San Francisco, CA: Morgan Kaufmann Publishers Inc., 2007. — 378 p.
7. Таненбаум, Э. Распределенные системы: принципы и парадигмы / Э. Таненбаум, М. Ван Стеен — Санкт-Петербург: Изд-во Питер, 2003. — 877 с.
8. Kogge, P.M. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems — Tech. Report TR-2008-13. — Univ. of Notre Dame, CSE Dept. — 2008. / P.M. Kogge, et al. URL: http://www.cse.nd.edu/Reports/2008/TR-2008-13.pdf (дата обращения: 25.07.2014).
9. Avizienis, A. Basic Concepts and Taxonomy of Dependable and Secure Computing / A. Avizienis, J.C. Laprie, B. Randell, C. Landwehr // IEEE Transactions on Dependable and Secure Computing. — 2004. — Vol. 1, — P. 11–33.
10. Jalote, P. Fault Tolerance in Distributed Systems / P. Jalote — New Jersey, Prentice Hall, 1994 — 448 p.
11. Тель, Ж. Введение в распределенные алгоритмы / Ж. Тель — Москва.: МЦМНО, 2009. — 616 с.
12. The computer failure data repository URL: https://www.usenix.org/cfdr (дата обращения: 25.07.2014).
13. Addressing the challenges of petascale computing for scientific discovery on information storage capacity, performance, concurrency, reliability, availability, and manageability URL: http://pdsi.nersc.gov/ (дата обращения: 25.07.2014).
14. Yuan, Y. Job failures in high performance computing systems: A large-scale empirical study / Y. Yuan, Y. Wu, Q. Wang, G. Yang, W. Zheng // Computers & Mathematics with Applications. — 2012. — Vol. 63, No 2. — P. 365–377.
15. Dong, X. A Case Study of Incremental and Background Hybrid In-Memory Checkpointing / X. Dong, N. Muralimanohar, N.P. Jouppi, Y. Xie // Proceedings of the 2010 Exascale Evaluation and Research Techniques Workshop (Pittsburgh, PA, USA March 13 – 14, 2010), — ACM, 2010 — P. 119–147.
16. Schroeder, B. Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You? / B. Schroeder, G.A. Gibson // Proceedings of the 5th USENIX Conference on File and Storage Technologies (San Jose, CA, USA February 13–16 2007) — USENIX, 2007. — P. 1–16.
17. Ferreira, K.B. Accelerating incremental checkpointing for extreme-scale computing / K.B. Ferreira, R. Riesen, P.G. Bridges, D. Arnold, R. Brightwell // Future Generation Computer Systems. — 2014. — Vol. 30, No 1. — P. 66–77.
18. Поляков, А.Ю. Оптимизация времени создания и объёма контрольных точек восстановления параллельных программ / А.Ю. Поляков, А.А. Данекина // Вестник СибГУТИ. – Новосибирск: СибГУТИ — 2010. — № 2. — С. 87–100.
19. Vaidya, N.H. A Case for Two-Level Distributed Recovery Schemes / N.H. Vaidya // Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement

and Modeling of Computer Systems (Ottawa, Canada, May 15-19 1995) — ACM, 1995, — P. 64–73.
20. Plank, J.S. Diskless Checkpointing / J.S. Plank, K. Li, M.A, Puening // IEEE Transanctions on Parallel Distributed Systems. — 1998. — Vol. 9, No 10. — P. 972–986.
21. X-COM parallel.ru URL: http://x-com.parallel.ru/node/10 (дата обращения: 25.07.2014).
22. Баранов, А.В. Программный комплекс «Пирамида» организации параллельных вычислений с распараллеливанием по данным / Баранов А.В., Киселёв А.В., Киселёв Е.А., Корнеев В.В., Семёнов Д.В. URL: http://agora.guru.ru/abrau2010/pdf/299.pdf (дата обращения: 25.07.2014).
23. OpenTS - технология и программное обеспечение поддержки распараллеливания программ URL: http://skif.pereslavl.ru/psi-info/rcms-open.ts/index.ru.html (дата обращения: 25.07.2014).
24. HTCondor high throughput computing URL: http://research.cs.wisc.edu/htcondor/index.html (дата обращения: 25.07.2014).
25. Berkeley Lab Checkpoint/Restart (BLCR) for LINUX URL: http://crd.lbl.gov/groups-depts/ftg/projects/current-projects/BLCR/ (дата обращения: 25.07.2014).
26. Open MPI: Open Source High Performance Computing URL: http://www.open-mpi.org (дата обращения: 25.07.2014).
27. MPICH URL: http://www.mpich.org (дата обращения: 25.07.2014).
28. MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE URL: http://mvapich.cse.ohio-state.edu (дата обращения: 25.07.2014).
29. Egwutuoha, I.P. A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. / I.P. Egwutuoha, D. Levy, B. Selic, S. Chen // The Journal of Supercomputing. — 2013. — Vol. 65, No. 3. — P. 1302–1326.
30. Message Passing Interface Forum URL: http://www.mpi-forum.org/ (дата обращения: 25.07.2014).
31. ICL Fault Tolerance URL: http://fault-tolerance.org/ulfm/ulfm-specification (дата обращения: 25.07.2014).
32. Dong, X. Leveraging 3D PCRAM technologies to reduce checkpoint overhead for future exscale systems, / X. Dong, N. Muralimanohar, N. Jouppi, R. Kaufmann, Y. Xie // Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (Portland, Oregon USA November 14-20, 2009). — ACM, 2009. — P. 57-68.
33. FT-MPI URL: http://icl.cs.utk.edu/ftmpi/people/index.html (дата обращения: 25.07.2014).

Бондаренко Алексей Алексеевич, к.ф.-м.н., научный сотрудник, Институт прикладной математики им. М.В. Келдыша РАН (Москва, Российская Федерация), bondaleksey@gmail.com.

Якобовский Михаил Владимирович, д.ф.-м.н., заведующий сектором «Программное обеспечение вычислительных систем и сетей», Институт прикладной математики им. М.В. Келдыша РАН (Москва, Российская Федерация), lira@imamod.ru

Поступила в редакцию 5 августа 2014 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2014, vol. 3, no. 3, pp. 20–36

FAULT TOLERANCE FOR HPC BY USING LOCAL CHECKPOINTS

A.A. Bondarenko, Keldysh Institute of Applied Mathematics (Moscow, Russian Federation),
M.V. Iakobovski, Keldysh Institute of Applied Mathematics (Moscow, Russian Federation)

One of the main problems that occur in the area of high-performance computing is to continue computations despite of failures. In this paper, we consider the main definitions relating to dependability, briefly review the failure rates for distributed systems and also survey the rollback-recovery approaches. The classic fault-tolerance technique used in parallel applications is the coordinated checkpointing protocol. This protocol takes a consistent global checkpoint snapshot by capturing the local state of each process node simultaneously and saves it on a parallel file system via I/O nodes. However, as the number of compute nodes increases and the size of applications grow, the performance overhead of this protocol can reach an unacceptable level. A solution to this problem is to use local storage for checkpointing. To provide protection, it is necessary to duplicate checkpoints to other local storages. In this work, we develop user level approach and present scheme for checkpointing to the local storages. We proof that, if the number of failures is less than the maximum allowable value for the scheme then it is possible to recover from consistent global checkpoint.

Keywords: parallel computing, fault tolerance, checkpoint, MPI.

References

1. Bland W., Bouteiller A., Hérault T., Bosilca G., Dongarra J. Post-failure recovery of MPI communication capability: Design and rationale. International Journal of High Performance Computing Applications. 2013. Vol. 27, No. 3. P. 244–254.
2. Cappello, F. Fault tolerance in petascale/exascale systems: Current knowledge, challenges and research opportunities. International Journal of High Performance Computing Applications. 2009. Vol. 23, No. 3. P. 212–226.
3. Hsu C.-H., Feng W.-C. A power-aware run-time system for high-performance computing. Proceedings of SC|05: The ACM/IEEE International Conference on High-Performance Computing, Networking, and Storage (Seattle, Washington USA November 12 – 18, 2005). IEEE Press, 2005. P. 1–9.
4. Sorin, D. Fault Tolerant Computer Architecture. Synthesis Lectures on Computer Architecture. Morgan&Claypool, 2009. 104 p.
5. Elnozahy E.N., Alvisi L., Wang Y.,Johnson D.B. A Survey of Rollback-Recovery Protocols in Message-Passing Systems. ACM Computing Surveys. 2002. Vol.34, No. 3 P. 375–408.
6. Koren I., Krishna C.M. Fault-Tolerant Systems. San Francisco, CA: Morgan Kaufmann Publishers Inc., 2007. 378 p.
7. Tanenbaum A.S., Steen M. Distributed Systems: Principles and Paradigms. New Jersey, Prentice Hall PTR, 2002. 803 p.

8. Kogge P.M. et al. ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems Tech. Report TR-2008-13. Univ. of Notre Dame, CSE Dept. 2008. URL: http://www.cse.nd.edu/Reports/2008/TR-2008-13.pdf (accessed: 25.07.2014).
9. Avizienis A., Laprie J.C., Randell B., Landwehr C. Basic Concepts and Taxonomy of Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing. 2004. Vol. 1, P. 11–33.
10. Jalote P. Fault Tolerance in Distributed Systems. New Jersey, Prentice Hall, 1994. 448 p.
11. Tel G. Introduction to Distributed Algorithms. Cambridge University Press, 2000. 596 p.
12. The computer failure data repository URL: https://www.usenix.org/cfdr (accessed: 25.07.2014).
13. Addressing the challenges of petascale computing for scientific discovery on information storage capacity, performance, concurrency, reliability, availability, and manageability URL: http://pdsi.nersc.gov/ (accessed: 25.07.2014).
14. Yuan Y., Wu Y., Wang Q., Yang G., Zheng W. Job failures in high performance computing systems: A large-scale empirical study. Computers & Mathematics with Applications. 2012. Vol. 63, No 2. P. 365–377.
15. Dong X., Muralimanohar N., Jouppi N.P., Xie Y. A Case Study of Incremental and Background Hybrid In-Memory Checkpointing. Proceedings of the 2010 Exascale Evaluation and Research Techniques Workshop (Pittsburgh, PA, USA March 13 – 14, 2010), ACM, 2010. P. 119–147.
16. Schroeder B., Gibson G.A. Disk Failures in the Real World: What Does an MTTF of 1,000,000 Hours Mean to You? Proceedings of the 5th USENIX Conference on File and Storage Technologies (San Jose, CA, USA February 13–16 2007) USENIX, 2007. P. 1–16.
17. Ferreira K.B., Riesen R., Bridges P.G., Arnold D., Brightwell R. Accelerating incremental checkpointing for extreme-scale computing. Future Generation Computer Systems. 2014. Vol. 30, No 1. P. 66–77.
18. Polyakov A.Yu. Optimizatsiya vremeni sozdaniya i objema kontrolnykh tochek vosstanovleniya parallelnykh program [Optimization of time creation and checkpoint's volume for parallel programs] // Vestnik SibGUTI [Bulletin of Siberian State University of Telecommunications and Information Sciences]. 2010. No. 2. P. 87–100.
19. Vaidya N.H. A Case for Two-Level Distributed Recovery Schemes. Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems (Ottawa, Canada, May 15-19 1995) ACM, 1995. P. 64–73.
20. Plank J.S., Li K., Puening M.A. Diskless Checkpointing. IEEE Transanctions on Parallel Distributed Systems. 1998. Vol. 9, No 10. P. 972–986.
21. X-COM parallel.ru URL: http://x-com.parallel.ru/node/10 (accessed: 25.07.2014).
22. Baranov A.V. Programnyj kompleks «Piramida» organizatsii parallelnykh vychislenij s rasparallelivaniem po dannym [software package «Pyramid» for organization of parallel computing with parallelization of data]. URL: http://agora.guru.ru/abrau2010/pdf/299.pdf (accessed: 25.07.2014).
23. OpenTS – tekhnologiya i programmnoe obespechenie podderzhki rasparallelivaniya programm [Technology and Software Support for Parallelization of Data-Parallel Applica-

tions] URL: http://skif.pereslavl.ru/psi-info/rcms-open.ts/index.ru.html (accessed: 25.07.2014).
24. HTCondor high throughput computing URL: http://research.cs.wisc.edu/htcondor/index.html (accessed: 25.07.2014).
25. Berkeley Lab Checkpoint/Restart (BLCR) for LINUX URL: http://crd.lbl.gov/groups-depts/ftg/projects/current-projects/BLCR/ (accessed: 25.07.2014).
26. Open MPI: Open Source High Performance Computing URL: http://www.open-mpi.org (accessed: 25.07.2014).
27. MPICH URL: http://www.mpich.org (accessed: 25.07.2014).
28. MVAPICH: MPI over InfiniBand, 10GigE/iWARP and RoCE URL: http://mvapich.cse.ohio-state.edu (accessed: 25.07.2014).
29. Egwutuoha I.P., Levy D., Selic B., Chen S. A survey of fault tolerance mechanisms and checkpoint/restart implementations for high performance computing systems. The Journal of Supercomputing. 2013. Vol. 65, No. 3. P. 1302–1326.
30. Message Passing Interface Forum URL: http://www.mpi-forum.org/ (accessed: 25.07.2014).
31. ICL Fault Tolerance URL: http://fault-tolerance.org/ulfm/ulfm-specification (accessed: 25.07.2014).
32. Dong X., Muralimanohar N., Jouppi N., Kaufmann R., Xie Y.. Leveraging 3D PCRAM technologies to reduce checkpoint overhead for future exscale systems. Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (Portland, Oregon USA November 14-20, 2009). ACM, 2009. P. 57–68.
33. FT-MPI URL: http://icl.cs.utk.edu/ftmpi/people/index.html (accessed: 25.07.2014).

Received 5 August 2014.

УДК 004.45, 004.272

МЕТОД ДЛЯ СОГЛАСОВАННОГО ВЫПОЛНЕНИЯ СЕМЕЙСТВА РАСПРЕДЕЛЕННЫХ АСИНХРОННО ВЗАИМОСВЯЗАННЫХ ТРАНЗАКЦИЙ

И.Г. Данилов

В работе предлагается метод обнаружения RW-конфликтов по разделяемым данным, возникающих во время конкурентного выполнения набора распределенных транзакций, который предназначен для предотвращения связанных с таким типом конфликтов аномалий выполнения.

Ключевые слова: распределенная транзакционная память, разделенное глобальное адресное пространство.

1. Введение

В 90-х годах прошлого века M. Herlihy и J. E. B. Moss описали оригинальную аппаратную реализацию [1] синхронизации доступа нескольких процессоров к разделяемой памяти: конкурентные операции чтения/записи разделяемой памяти процессора могут объединяться в некоторый атомарный (неделимый) набор действий — *транзакцию* — и выполняться совершенно независимо от операций других процессоров. Предложенный механизм синхронизации, который в настоящее время является объектом активных научных исследований, был назван *транзакционной памятью, ТП* (англ. *transactional memory, TM*). Однако можно отметить, что сама идея не так уж и нова: впервые применять подобный механизм синхронизации для любых вычислительных процессов вообще, а не только для управления доступом к данным в *базах данных* (БД), предложил D. B. Lomet. В своей работе [2] Lomet описал концепцию *атомарных действий* (англ. *atomic actions*), позже реализованную Liskov и Scheifler в языке распределенного программирования Argus [3]. Появление идеи транзакционной памяти стало результатом обработки и осмысления многолетних исследований в области традиционной синхронизации вычислительных процессов и моделей согласованности памяти с одной стороны: схожие принципы и соответствующие конструкции можно встретить у Хоара — концепция *мониторов* [4], и с другой стороны — в области теории транзакционной обработки данных в базах данных.

В настоящее время актуальными являются исследования возможностей и преимуществ применения механизмов транзакционной памяти для масштабируемых вычислительных систем с распределенной памятью, в первую очередь для кластерных вычислительных систем. В силу своих особенностей ТП может оказаться более эффективным и масштабируемым по сравнению с традиционными решениями на основе алгоритмов распределенного взаимного исключения подходом к синхронизации в таких системах, что уже подтверждает ряд имеющихся исследований [5–7].

2. Обзор работ по тематике исследования

2.1. Свойства алгоритмов ТП

Свойством *живости* [8] для алгоритмов транзакционной обработки данных является критерий *согласованного выполнения транзакций* или просто *критерий согласованности.*

Примером может служить классический критерий *сериализуемости* транзакций БД [9]. Однако, из-за существующих различий между транзакциями в БД и транзакциями памяти, прежде всего из-за большей автономности последних, традиционного критерия сериализуемости, который является основополагающим для транзакций БД, не хватает для определения корректности выполнения набора транзакций памяти. В нем рассматриваются лишь *зафиксированные* транзакции и не затрагивается вопрос корректности выполнения *"живых"*, еще не зафиксированных транзакций. При использовании оптимистичных методов синхронизации такой критерий подходит только для полностью контролируемых сред или *песочниц*, какой является для транзакций СУБД, и не является удовлетворительным для транзакций памяти. Действительно, при конкурентном выполнении набора транзакций возможны ситуации гонок и, как следствие, несогласованные чтения: как из-за чередования конкурентных операций, так и, возможно, из-за несогласованного состояния памяти в случае использования клонирования (кэширования) данных, либо механизма откатов (см. классические проблемы конкурентного выполнения транзакций [10]), а валидация (обнаружение конфликтов по данным) происходит только перед самой фиксацией транзакции. Транзакции, выполняемые в еще не обнаруженном состоянии гонок, называются *транзакциями-зомби* [11] или *обреченными* (англ. *doomed*).

С целью избежания подобных ситуаций для алгоритмов ТП был предложен более строгий в плане корректности выполнения критерий, который является вариантом строгой сериализуемости, названный критерием *скрытности* (англ. *opacity*) [12]. Он предполагает, что: а) все операции, выполненные каждой зафиксированной транзакцией, представляются так, как если бы они были выполнены в одной неделимой точке жизненного цикла транзакции; б) ни одна операция любой прерванной транзакции не должна быть видна другим, в том числе выполняющимся транзакциям; в) каждая транзакция в любой момент времени наблюдает *согласованное* состояние системы (невозможны несогласованные чтения). Можно сказать, что критерий скрытности последовательно упорядочивает не только зафиксированные, но и прерванные транзакции, без наблюдения результатов их выполнения другими транзакциями набора. Таким образом, при возникновении конфликт должен быть сразу обнаружен (например, с помощью проверки на наличие изменений версий прочитанных ранее значений), и одна из конфликтующих транзакций прервана и перевыполнена заново.

2.2. Обнаружение конфликтов конкурентного выполнения распределенных транзакций

Одним из самых важных вопросов помимо выбора *стратегии обнаружения и разрешения* конфликтов конкурентного выполнения набора транзакций является реализация **метода обнаружения конфликтов**. При использовании *отложенной* стратегии обновления данных существует необходимость в обнаружении только RW-конфликтов, т.к. для записи используется промежуточный буфер (WR-конфликты невозможны), а сама память изменяется только в момент фиксации транзакции. Возникающие при этом "отложенные" WW-конфликты зачастую разрешаются на основе правила *"первая фиксация побеждает"* (англ. *first committer wins*, FCW): в момент фиксации транзакция проверяет перезаписываемые ею данные на наличие обновлений, сделанных конкурентными транзакциями, и, при обнаружении таких обновлений, откатывается. Для оптимистичных механизмов синхронизации [13] основной процедурой методов обнаружения конфликтов является валидация (проверка) множества считанных транзакцией значений $\mathcal{R}set$. Но то, как и когда данную

процедуру использовать, зависит от реализуемого для отслеживания конкурентных обновлений разделяемых данных метода обнаружения конфликтов. Подходы, которые применяются в алгоритмах для мультипроцессорных систем, нельзя напрямую использовать для мультикомпьютерной или *распределенной транзакционной памяти, РТП* (англ. *distributed software transactional memory*, DSTM).

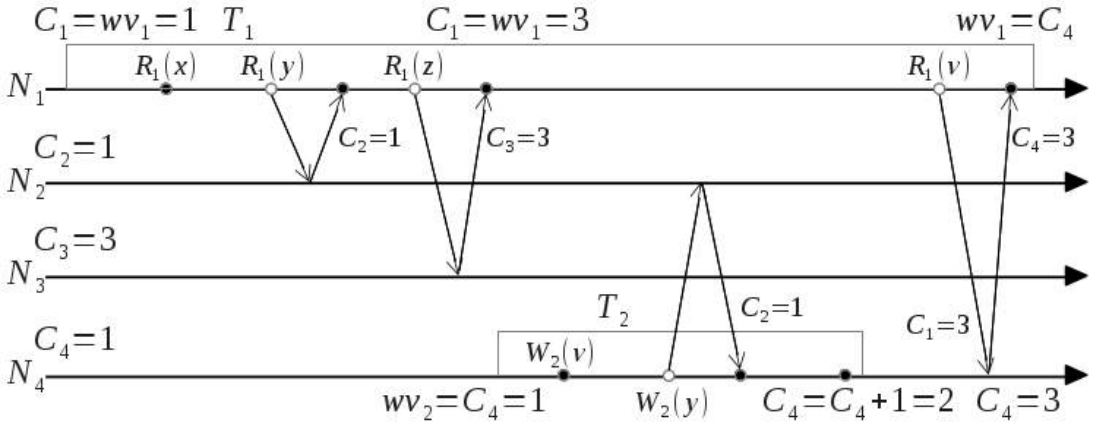
В работе [6] был предложен метод обнаружения RW-конфликтов с использованием версий для данных и валидации множества $\mathcal{R}set$ путем отслеживания причинно-следственного порядка между операцией чтения транзакции и операциями записи остальных транзакций системы. Для этого авторы предложили специального вида логические часы [14], которые реализованы в виде целочисленного счетчика и подчиняются следующим правилам:

1. если a — успешное событие фиксации транзакции $T_i \in T$, выполняемой на узле системы $\mathcal{N}_k$, то часы
$$C_k(a) \leftarrow C_k \leftarrow C_k + 1 \tag{1}$$
2. пусть a — событие приема на узле системы $\mathcal{N}_k$ сообщения φ, с назначенной меткой времени, равной значению часов узла системы $\mathcal{N}_m$ в момент события b отправления сообщения φ: $t_q = C_m(b)$; тогда после события a часы C_k устанавливаются в значение большее среди текущего значения часов C_k и метки t_q:
$$C_k \leftarrow \max(C_k, t_q) \tag{2}$$

Сам метод обнаружения конфликтов, названный методом *продвижения транзакции* (англ. *transaction forwarding*) заключается в следующем:

- при старте транзакция T_i, выполняемая на узле $\mathcal{N}_k$ считывает текущее значение часов C_k и сохраняет его в переменной wv;
- транзакция T_i *продвигает свое стартовое время* wv до значения $wv' > wv$, т.е. делает присваивание $wv = wv'$, только в случае успешной валидации $\mathcal{R}set$: для всех объектов в $\mathcal{R}set$ их текущая версия сравнивается с wv и, если версия какого-либо объекта превышает wv, то валидация заканчивается неуспешно, а транзакция откатывается;
- для чтения *удаленного* объекта транзакция T_i посылает сообщение-запрос на узел расположения объекта $\mathcal{N}_m$; при получении ответного сообщения φ к посланному ранее сообщению-запросу с текущей версией объекта транзакции необходимо продвинуть свое стартовое время wv до значения t_q в случае, если $wv < t_q$ (см. выражение 2 для логических часов); если же $wv \geq t_q$, то объект может быть считан безопасно;
- при чтении *локального* объекта, расположенного на узле выполнения транзакции, его версия проверяется и, если она больше, чем wv, то транзакция откатывается;
- в момент фиксации транзакция наращивает часы C_k (см. выражение 1) и назначает новую версию для всех перезаписываемых объектов равную новому значению часов C_k.

На базе предложенного метода авторами разработан алгоритм TFA, который по производительности превосходит все существующие конкурентные решения [6] и, как заявлено авторами, соответствуют критерию скрытности транзакций. Однако это не совсем так. Представим ситуацию, иллюстрация к которой изображена на рисунке. В системе из четырех узлов: $(\mathcal{N}_1, \mathcal{N}_2, \mathcal{N}_3, \mathcal{N}_4)$ выполняются две транзакции T_1 и T_2. При этом транзакция T_2 перезаписывает между $\mathcal{R}_1(z)$ и $\mathcal{R}_1(v)$ считанный ранее T_1 объект y, но в момент чтения $\mathcal{R}_1(v)$ продвижения транзакции и, следовательно валидации $\mathcal{R}set$ не происходит, в силу того, что $wv_1 = C_4$. В результате чтение $\mathcal{R}_1(v)$ получается несогласованным.



Ситуация несогласованного чтения при использовании метода продвижения транзакции

Кроме этого, недостатком предложенного подхода является его ориентированность на распределенные протоколы когерентности кэша (например Relay [15]), которые плохо масштабируются, и предоставление всего одной перезаписываемой копии объекта транзакциями системы. В работе [7] авторами был предложен подход и ряд принципов для эффективной реализации программно-организуемой транзакционной памяти на основе модели PGAS и односторонних коммуникаций. Предложенный подход был реализован в системе Cluster-STM, хорошо масштабируемой на кластерной системе с сотнями вычислительных узлов. Основной недостаток Cluster-STM — отсутствие поддержки динамического параллелизма на уровне потоков, ограничение "один процессор, одна транзакция" — был устранен в системе GTM [16], также построенной на основе модели PGAS и SPMD-подхода, но вместе с этим предоставляющей поддержку динамического параллелизма на уровне потоков и удаленного вызова процедур. Недостатком обоих систем является предоставление только лишь гарантии сериализуемости; вопрос возможного возникновения гонок и, как следствие некорректное выполнение программы, не рассматривается.

Кроме вышерассмотренных существует ряд подходов, основанных на репликации и многоверсионности данных и предоставляющих более слабые по сравнению со скрытностью гарантии согласованности: GMU [17], SCORe [18] и др. Репликация позволяет только читающим транзакциям выполняться локально, без генерации сетевого трафика и валидации удаленных данных. В GMU применяется многоверсионная схема данных, *векторные логические часы* [19] для отслеживания причинно-следственной связи между событиями, а также *частичная истинная* схема репликации данных. GMU предоставляет слабый критерий согласованности EUS (англ. *extended update-serializability*), согласно которому различные только читающие транзакции могут видеть результат обновления неконфликтующих пищущих транзакций в различном порядке. Улучшенная версия протокола частичной истинной репликации данных, который назван SCORe, гарантирует более строгий критерий *выполнимой сериализуемости одной копии* (англ. *executing one-copy serializability*, 1CS) [18]. Протокол SCORe достаточно хорошо масштабируем и эффективен за счет использования локальной мультиверсионной схемы данных, позволяющей только читающим транзакциям всегда выполняться без отката, и за счет масштабируемой схемы синхронизации логических часов, в которой связанные транзакции обмениваются лишь одним скалярным числом — меткой часов.

Таким образом, можно сделать вывод, что в настоящий момент не разработано методов и алгоритмов РТП, гарантирующих выполнение критерия скрытности. Разработанные ал-

горитмы, дающие более слабые гарантии, реализуются на основе динамических, интерпретируемых языков, таких как Java, которые позволяют обнаружить и, возможно, устранить исключительные ситуации, возникающие из-за несогласованных чтений. Для более популярных в области выскопроизводительных вычислений не динамических языков, таких как C/C++ представлено всего несколько программных средств: Cluster-STM и DMV [20], гарантирующие сериализуемость, и HyflowCPP [21] — программный фреймворк, реализующий алгоритм TFA, который как было показано выше не предоставляет гарантию выполнения критерия скрытности.

3. Разработка метода обнаружения RW-конфликтов по данным в распределенной системе

3.1. Модель системы

Пусть $\mathbb{N} = \{n_1, n_2, \ldots, n_{\mathcal{N}}\}$ — множество узлов *распределенной асинхронной вычислительной системы без сбоев* [8], а $\mathbb{T} = \{T_1, T_2, \ldots, T_{\mathcal{M}}\}$ — множество распределенных в системе *транзакций*. Кроме транзакций система также включает в себя *память* с **последовательной** моделью согласованности, с помощью которой транзакции взаимодействуют. Будем подразделять память на два вида:

1. Plm_k — ***приватная***, доступная только транзакции $T_k \in \mathbb{T}$, $k = \overline{1, \mathcal{M}}$ и относящаяся к или *выделенная* на узле выполнения T_k: $n_j \in \mathbb{N}$, $j = \overline{1, \mathcal{N}}$ память. Множество L представляет собой совокупность всей локальной памяти в системе: $L = Plm_1 \cup Plm_2 \cup \ldots \cup Plm_{\mathcal{M}}$;
2. G — ***разделяемая***, доступная всем транзакциям системы память такая, что $G = Pgm_1 \cup Pgm_2 \cup \ldots \cup Pgm_{\mathcal{N}}$, где Pgm_j — часть разделяемой памяти, выделенная на узле $n_j \in \mathbb{N}$, $j = \overline{1, \mathcal{N}}$.

Будем считать, что сами транзакции неподвижны и выполняются на том же самом узле, где были инициированы, а удаленные данные при этом могут перемещаться и кэшироваться на узле выполнения транзакции. Подобная модель, которая была предложена в работе [22], называется моделью с *потоком данных*. Также будем считать, что транзакции используют *отложенную* стратегию обновления данных.

Память представляет из себя множество специальных объектов — *ячеек*. Каждой ячейке памяти x соответствует некоторое определенное *значение*, задаваемое отображением $V(x)$. Обозначим символом $\mathcal{V}$ множество всех возможных значений, которые могут принимать ячейки памяти. Совокупность всех ячеек системы будем обозначать символом $\mathcal{M}$: $\mathcal{M} = L \cup G$. Будем говорить, что множество ячеек $\mathcal{M}_1$ равно множеству $\mathcal{M}_2$, когда:

$$|\mathcal{M}_1| = |\mathcal{M}_2| \wedge (\forall x \in \mathcal{M}_1 \; \exists! y \in \mathcal{M}_2 : x = y \wedge V(x) = V(y))$$

Для описания системы взаимодействующих распределенных транзакций воспользуемся по аналогии рядом базирующихся на понятии *машины состояний* определений, предложенных в [8], и адаптируем их для распределенной системы:

Определение 3.1. ***Локальным алгоритмом*** *транзакции будем называть пятерку* $(Z, I, \vdash^i, \vdash^r, \vdash^w)$, *в которой* Z *— множество состояний,* I *— это некоторое подмножества множества* Z, $\vdash^i$ *— отношение на множестве* $Z \times Z$, $\vdash^r$ *— отношение на множестве* $Z \times \mathcal{M} \times Z$, $\vdash^w$ *— отношение на множестве* $Z \times \mathcal{M} \times \mathcal{V} \times Z$. *Двуместное отношение* $\vdash$ *на*

множестве Z определяется соотношением:

$$c \vdash d \Longleftrightarrow (c,d) \in \vdash^i \vee \exists x \in \mathcal{M}((c,x,d) \in \vdash^r) \vee \exists x \in \mathcal{M}, \exists v \in \mathcal{V}((c,x,v,d) \in \vdash^w)$$

Отношение $\vdash^i$ представляет собой переход между состояниями, связанный с *внутренними* событиями транзакции $T_k \in \mathbb{T}$, отношение $\vdash^r$ — событиями *чтения* значения из ячейки памяти, а $\vdash^w$ — событиями *записи* значения в ячейку памяти. Транзакция $T_k \in \mathbb{T}$ может выполняться на любом узле $n_j \in \mathbb{N}$ вычислительной системы (ВС). Выполнение транзакции T_k — выполнение системы переходов $(Z_k, I_k, \vdash^i_k, \vdash^r_k, \vdash^w_k)$. Состояние транзакции S_{T_k} описывается собственно состоянием транзакции c_{T_k} и состоянием или *слепком* доступной ей *локальной памяти* Plm_k: $S_{T_k} = (c_{T_k}, Plm_k)$. Внутренние события и события чтения, которые описываются с помощью отношений $\vdash^i_k, \vdash^r_k$ соответственно, могут изменять собственно состояние транзакции, а события записи в том числе и состояние доступной ей памяти: как локальной, так и глобальной.

Для описания распределенной системы в целом введем понятие *распределенного алгоритма*, составленного из соответствующих компонент транзакций, и системы переходов, которую данный алгоритм порождает:

Определение 3.2. ***Распределенным алгоритмом*** *семейства транзакций $\mathbb{T} = \{T_1, T_2, \ldots, T_{\mathcal{M}}\}$ будем называть совокупность локальных алгоритмов, каждый из которых соответствует в точности одной транзакции из $\mathbb{T}$.*

Состояние семейства транзакций $\mathbb{T}$ будем называть *конфигурацией*. Каждая конфигурация системы определяется состояниями всех транзакций и состоянием разделяемой памяти $\mathcal{G}$.

Поведение распределенного алгоритма семейства транзакций $\mathbb{T}$ описывается с помощью системы переходов, определяемой следующим образом:

Определение 3.3. *Пусть задано семейство транзакций $\mathbb{T} = \{T_1, T_2, \ldots, T_{\mathcal{M}}\}$, а локальный алгоритм каждой транзакции T_k представлен пятеркой $(Z_{T_k}, I_{T_k}, \vdash^i_{T_k}, \vdash^r_{T_k}, \vdash^w_{T_k})$. Будем говорить, что система переходов $\mathcal{S} = (\mathcal{C}, \to, \mathcal{I})$* ***порождена распределенным алгоритмом для семейства асинхронно взаимосвязанных транзакций*** *$\mathbb{T}$, если выполнены следующие соотношения:*

1. $\mathcal{C} = \{(S_{T_1}, \ldots, S_{T_{\mathcal{M}}}, \mathcal{G}) : (\forall T \in \mathbb{T} : S_T \in Z_T) \wedge (\forall x \in \mathcal{L}, \forall y \in \mathcal{G} : V(x), V(y) \in \mathcal{V})\}$, *где $\mathcal{C}$ — множество конфигураций семейства транзакций;*
2. $\to = (\bigcup_{T \in \mathbb{T}} \to_T)$, *где символом $\to_T$ обозначаются переходы, которые соответствуют изменениям состояния транзакции T, т.е. $\to_{T_i}$ — множество всех пар вида*

$$(S_{T_1}, \ldots, S_{T_k}, \ldots, S_{T_{\mathcal{M}}}, \mathcal{G}_1),\ (S_{T_1}, \ldots, S'_{T_k}, \ldots, S_{T_{\mathcal{M}}}, \mathcal{G}_2),$$

   таких, что выполняется одно из следующих условий:
   - $(S_{T_k}, S'_{T_k}) \in \vdash^i_{T_k}$ *и* $\mathcal{M}_1 = \mathcal{M}_2$;
   - *для некоторой ячейки* $x \in \mathcal{M}_1$ *имеется событие чтения* $(S_{T_k}, x, S'_{T_k}) \in \vdash^r_{T_k}$ *и* $\mathcal{M}_1 = \mathcal{M}_2$;
   - *для некоторой ячейки* $x \in \mathcal{M}_1$ *и значения* $v \in \mathcal{V}$ *имеется событие записи* $(S_{T_k}, x, v, S'_{T_k}) \in \vdash^w_{T_k}$ *так, что* $|\mathcal{M}_1| = |\mathcal{M}_2|$ *и* $\exists! y \in \mathcal{M}_2 : x = y \wedge V(y) = v$;
3. $\mathcal{I} = \{(S_{T_1}, \ldots, S_{T_{\mathcal{M}}}, \mathcal{G}) : (\forall T \in \mathbb{T} : S_T \in I_T)$
   $\wedge (\forall x \in \mathcal{L}, \forall y \in \mathcal{G} : V(x) = V_0(x), V(y) = V_0(y), V(x), V(y) \in \mathcal{V})\}$, *где $\mathcal{I}$ — множество начальных конфигураций семейства транзакций, а $V_0(x) : \forall x \in \mathcal{L} \cup \mathcal{G}$ — задает начальные значения ячеек памяти.*

Соответственно выполнением распределенного алгоритма называется *всякое* выполнение системы переходов, которая порождена этим алгоритмом. Конкретное выполнение E задается в видет последовательности конфигураций, через которые система проходит в данном выполнении: $E = (\gamma_0, \gamma_1, \dots)$. Выполнение может быть либо конечным, если для него имеется *заключительная* конфигурация, т.е. такая конфигурация $\gamma \in \mathcal{C} : \nexists \delta \,|\, \gamma \to \delta$, либо бесконечным, если для него заключительной конфигурации не существует. С выполнением E связана последовательность событий $\bar{E} = (e_0, e_1, \dots)$, где e_i — некоторое *допустимое* событие, обозначающее преобразование конфигурации γ_i в γ_{i+1}: $e_i(\gamma_i) = \gamma_{i+1}$.

Очевидно, что если два последовательных события в выполнении влияют на не пересекающиеся подмножества конфигураций, то эти события *независимы* и могут "выполняться" в другом порядке. В противном случае, если события нельзя поменять местами, то между ними существует *причинно-следственная зависимость*. Для любой последовательности событий $\bar{E}$, связанной с некоторым выполнением E выделим три вида такой зависимости.

Определение 3.4. *Будем говорить, что между двумя событиями $e_i, e_j \in \bar{E}$ существует:*

1. *истинная зависимость: $e_i \prec^{wr} e_j \iff$ событие e_i — событие записи в некоторую ячейку $x \in \mathcal{M}$ значения $v \in \mathcal{V}$, а событие e_j — последующее после e_i событие чтения из ячейки x значения v;*
2. *выходная зависимость: $e_i \prec^{ww} e_j \iff$ событие e_i — событие записи в некоторую ячейку $x \in \mathcal{M}$ значения $v_1 \in \mathcal{V}$, а событие e_j — последующее после e_i событие записи в эту же ячейку x значения $v_2 \in \mathcal{V}$;*
3. *антизависимость: $e_i \prec^{rw} e_j \iff$ событие e_i — событие чтения из некоторой ячейки $x \in \mathcal{M}$ значения $v_1 \in \mathcal{V}$, а событие e_j — последующее после e_i событие записи в эту же ячейку значения $v_2 \in \mathcal{V}$.*

По аналогии с отношением причинно-следственного порядка, которое Лэмпорт ввел для систем с асинхронным обменом сообщениями [14], зададим такое же отношение для описанной выше асинхронной системы с распределенной общей памятью.

Определение 3.5. *Причинно-следственное отношение частичного порядка $\prec$ на множестве событий системы $\bar{E}$ выполнения E — это такое наименьшее отношение, удовлетворяющее следующим трем условиям:*

1. *если e и f — два события одной транзакции и e произошло раньше, чем f, тогда $e \prec f$;*
2. *пусть e и f два разных события, тогда если $e \prec^{wr} f \vee e \prec^{ww} f \vee e \prec^{rw} f$, то $e \prec f$;*
3. *отношение $\prec$ транзитивно, то есть если $e \prec g$ и $g \prec f$, то $e \prec f$.*

Определим два вида специальных событий, допустимых при выполнении распределенного алгоритма семейства транзакций. Первый вид событий — событие *отката* $\mathcal{A}_k$: такое допустимое событие в последовательности событий транзакции T_k $\bar{E}_{T_k} = (e_0, \dots, e_i, \mathcal{A}_k, e'_0, \dots) \,|\, \bar{E}_{T_k} \subseteq \bar{E}$, представленное парой $\mathcal{A}_k = (S^i_{T_k}, S'_{T_k}) \in \vdash^i_{T_k}$, что $\mathcal{A}_k(\gamma_{i+1}) = \gamma'_0$, где $\gamma'_0 = (S_{T_1}, \dots, S'_{T_k}, \dots, S_{T_N}, G') : S'_{T_k} \in I_{T_k} \wedge (\forall x \in Plm_k : V(x) = V_0(x) \in \mathcal{V})$. Второй вид специальных событий — событие *фиксации* C_k транзакции T_k: такое допустимое событие в последовательности событий $\bar{E}_{T_k} = (e_0, \dots, e_i, C_k)$, представленное парой $C_k = (S^i_{T_k}, S'_{T_k}) \in \vdash^i_{T_k}$, что S'_{T_k} — *заключительное* состояние T_k в выполнении E.

3.2. Метод обнаружения RW-конфликтов по данным в распределенной системе

Будем считать, что с каждой ячейкой $x : x \in \mathcal{O}$ связана некоторая *версия* или *метка* $ts_x^k : ts_x^k \in \mathcal{TS}set$, присвоенная x транзакцией $\mathcal{T}_k$. Определим на множестве $\mathcal{TS}set$ функцию $\mathcal{TS}$ такую, что:

$$\mathcal{TS} : \mathcal{TS}set \to \{\langle j, tsn\rangle : j \in \mathbb{Z}_+, tsn \in \mathbb{Z}_{\geqslant 0}\},$$

где j — целое положительное число, номер узла ВС $n_j \in \mathbb{N}$, а tsn — целое неотрицательное число, которое является отметкой времени часов узла n_j.

Определим, аналогично тому, как это сделано в работе [6], логические часы $\mathcal{CLK}_m$ для узла ВС $n_m \in \mathbb{N}$:

1. если C_k — успешное событие фиксации транзакции $\mathcal{T}_k \in \mathbb{T}$, выполняемой на узле системы n_m, то
$$\mathcal{CLK}_m(C_k) \leftarrow \mathcal{CLK}_m \leftarrow \mathcal{CLK}_m + 1 \tag{3}$$
2. пусть e — событие чтения ячейки $x \in \mathcal{O}$ транзакцией $\mathcal{T}_k \in \mathbb{T}$, выполняемой на узле системы n_m; при этом если ячейке назначена версия $ts_x : \mathcal{TS}(ts_x) = \langle j, tsn\rangle$, тогда после события e часы $\mathcal{CLK}_m$ устанавливаются в значение большее среди текущего значения часов $\mathcal{CLK}_m$ и отметки tsn версии ts_x:
$$\mathcal{CLK}_m \leftarrow \max(\mathcal{CLK}_m, tsn) \tag{4}$$

Установим следующие соотношения для всех возможных значений версий ячеек $ts_i \in \mathcal{TS}set$. Пусть $\forall ts_1, ts_2 \in \mathcal{TS}set : \mathcal{TS}(ts_1) = \langle j_1, tsn_1\rangle \wedge \mathcal{TS}(ts_2) = \langle j_2, tsn_2\rangle$, тогда выполняется:

$$\begin{gathered} ts_1 =, <, > ts_2 \iff tsn_1 =, <, > tsn_2, \\ ts_1 \leq, \geq ts_2 \iff tsn_1 \leq, \geq tsn_2, \\ ts_1 \equiv ts_2 \iff tsn_1 = tsn_2 \wedge j_1 = j_2, \\ ts_1 \leqq, \geqq ts_2 \iff ts_1 <, > ts_2 \vee ts_1 \equiv ts_2 \end{gathered} \tag{5}$$

Определим *множество прочитанных ячеек* транзакции $\mathcal{T}_k$ как: $\mathcal{RS}et_k = \{x \,|\, x \in \mathcal{O} \wedge \exists e : e \in \bar{E}_{T_k} \wedge e$ — событие чтения ячейки $x\}$. На множестве $\mathcal{RS}et_k$ зададим функцию $\mathcal{RS}_k : \mathcal{RS}et_k \to \{ts \,|\, ts \in \mathcal{TS}set\}$ — версия ячейки, которой соответствует некоторое значение $v \in \mathcal{V}$. Таким образом $\mathcal{RS}_k(x) = ts_x$.

Определим для каждой транзакции $\mathcal{T}_k$ вектор $VC_k : |VC_k| = |\mathbb{N}| = \mathcal{N}$ так, что $j-$й элемент вектора равен $VC_k[j] = tsn$ — некоторой отметке времени логических часов узла с номером равным j. Предлагаемый метод обнаружения RW-конфликтов в распределенной системе предполагает использование отложенной стратегия обновления данных и заключается в следующем:

- при старте транзакции $\mathcal{T}_k$, выполняемой на узле n_m, вектор VC_k инициализируется нулевыми значениями: $VC_k[i] \leftarrow 0, \forall i = \overline{1, \mathcal{N}}$;
- при чтении значения ячейки $x \in \mathcal{O}$ с соответствующей версией $ts_x \in \mathcal{TS}set : \mathcal{TS}(ts_x) = \langle j, tsn\rangle$ проверяется, если $VC_k[j] < tsn$, то производится валидация объектов $\mathcal{RS}et_k$ и присваивание $VC_k[j] \leftarrow tsn$, а также изменяются часы $\mathcal{CLK}_m$ (см. выражение 4 для логических часов); конфликты отсутствуют и объект может быть считан безопасно только в случае *успешной* валидации после чего в $\mathcal{RS}et_k$ добавляется x: $\mathcal{RS}_k(x) = ts_x$;

- при записи ячейки $x \in \mathcal{G}$ производится определение ее текущей версии и при необходимости изменяются часы $CL\mathcal{K}_m$ согласно выражению 4, а новое значение v' сохраняется в буфере для последующей записи во время фиксации (отложенная стратегия обновления данных);
- валидация производится для всех объектов x чьи версии сохранены ранее в $\mathcal{R}set_k$ путем сравнения текущей версии объекта ts'_x относительно сохраненной в $\mathcal{R}set_k$ версии ts_x: $\mathcal{RS}_k(x) = ts_x$ и если $ts'_x > ts_x$, то валидация заканчивается *неуспешно*;
- в момент события C_k фиксации транзакции $\mathcal{T}_k$, выполняемой на узле n_m, наращиваются часы $CL\mathcal{K}_m$ (см. выражение 3), а все объекты *атомарно* перезаписываются новым значением v' с новой версией равной $ts^k_x = \langle m, CL\mathcal{K}_m(C_k)\rangle$.

Сформулируем и докажем следующее утверждение:

Теорема 3.1. *Предложенный метод при условии атомарной перезаписи новыми значениями ячеек памяти во время фиксации транзакций позволяет гарантированно обнаруживать во время выполнения распределенного семейства асинхронно взаимосвязанных транзакций RW-конфликты по данным, которые могут привести к ситуации несогласованного чтения.*

Доказательство. Допустим, что это не так. Тогда для некоторой транзакции $\mathcal{T}_k$ узла ВС n_m в выполнении E распределенного алгоритма, реализующего предложенный метод, существует событие несогласованного чтения e'' ячейки $y \in \mathcal{G}$ с версией $ts^s_y \in \mathcal{T}Sset : \mathcal{T}S(ts^s_y) = \langle j, tsn^s\rangle$ и $VC_k[j] \geq tsn^s$, т.о. после e'' валидация $\mathcal{R}set_k$ не производится. При этом в выполнении E для $\mathcal{T}_k$ есть также предшествующее событие чтения e' ячейки $x \in \mathcal{G}$ с версией, созданной транзакцией $\mathcal{T}_r$, $ts^r_x \in \mathcal{T}Sset : \mathcal{T}S(ts^r_x) = \langle i, tsn^r\rangle$ такое, что $e' \prec e''$.

Существование события несогласованного чтения e'' ячейки $y \in \mathcal{G}$ означает то, что $\exists$ два события записи f' и f'' транзакции $\mathcal{T}_s$ узла n_j: f' — событие записи ячейки $x \in \mathcal{G}$, а f'' — событие записи ячейки $y \in \mathcal{G}$ с соответствующими версиями $ts^s_x, ts^s_y \in \mathcal{T}Sset : \mathcal{T}S(ts^s_x) = \mathcal{T}S(ts^s_y) = \langle j, tsn^s\rangle \wedge ts^s_x > ts^r_x$. Причем при условии атомарности перезаписи новыми значениями ячеек памяти во время фиксации транзакций выполняется: 1) $e' \prec^{rw} f'$, $e'' \prec^{rw} f''$, либо 2) $e' \prec^{rw} f'$, $f'' \prec^{wr} e''$. При этом $f' \prec f'' \vee f'' \prec f'$. В первом случае событие e'' не является событием несогласованного чтения, что противоречит первоначальному условию.

Во втором случае, так как после события чтения e'' выполняется соотношение $VC_k[j] \geq tsn^s$, то для транзакции $\mathcal{T}_k$ должно $\exists$ событие чтения g ячейки $z \in \mathcal{G}$ с версией $ts^q_z \in \mathcal{T}Sset$: $\mathcal{T}S(ts^q_z) = \langle j, tsn^q\rangle \wedge VC_k[j] = tsn^q \geq tsn^s$. Т.е. ts^q_z произведена некоторой транзакцией $\mathcal{T}_q$. Тогда, если $tsn^q = tsn^s$, то $\mathcal{T}_q$ — это и есть транзакция $\mathcal{T}_s$, поэтому при чтении ячейки $z \in \mathcal{G}$ валидация $\mathcal{R}set_k$ должна отследить изменение версии ячейки $x \in \mathcal{G}$ и дальнейшее событие несогласованного чтения e'' невозможно: т.о. приходим к противоречию. Если $tsn^q > tsn^s$, то $\mathcal{T}_q$ и $\mathcal{T}_s$ разные транзакции одного узла ВС n_j, но $\mathcal{T}_q$ завершилась позже $\mathcal{T}_s$, следовательно, валидация $\mathcal{R}set_k$ после события чтения g также должна отследить изменение версии и ячейки $x \in \mathcal{G}$, следовательно дальнейшее событие несогласованного чтения e'' невозможно. Опять приходим к противоречию. □

Стоит отметить, что условие атомарной перезаписи ячеек памяти при фиксации транзакции обычно выполняется в силу использования специальных протоколов фиксации, таких как двухфазный протокол фиксации (2PC).

4. Программная реализация предложенного метода

На основе разработанного метода предложены алгоритмы и создана программная система распределенной транзакционной памяти DSTM_P1 [23, 24]. Данная система реализована на языке C++ и предназначена для запуска и контроля над выполнением многопоточных приложений, написанных на языке Си с использованием метода синхронизации на основе распределенной кэшируемой программно-организуемой транзакционной памяти, на кластерных вычислительных системах под управлением операционной системы Linux.

DSTM_P1 можно условно разделить на две основные части: *среду выполнения приложений* и *прикладной программный интерфейс* (API), доступный для написания многопоточных приложений. Среда выполнения является своего рода программной *"песочницей"* для приложений пользователя и, по выбору пользователя. API системы DSTM_P1 подразделяется на:

- интерфейс для работы с распределенной разделяемой памятью *Idstm_malloc*;
- интерфейс для запуска и управления выполнением распределенных потоков *Idstm_pthread*;
- интерфейс для барьерной синхронизации распределенных потоков *Idstm_barrier*.

Общая схема работы DSTM_P1 может быть описана следующим образом: (1) пользователь системы загружает исходные коды многопоточного приложения, написанного на языке C с использованием библиотеки Pthread и применением атомарных конструкций (__tm_atomic) в местах доступа к разделяемым данным; (2) приложение компилируется с помощью DTMC [8] в язык промежуточного представления; (3) далее полученное представление трансформируется в представление, содержащее соответствующие вызовы функций библиотеки транзакционной памяти для всех инструкций доступа к памяти атомарного блока; (4) представление, полученное на предыдущем шаге линкуется с необходимыми библиотеками (транзакционной памяти, «обертками» системных библиотек pthread и malloc); (5) на лету компилируется и исполняется main-функция приложения; (6) все вызовы функций библиотеки pthread и malloc переадресуются в соответствующие вызовы функций библиотек «оберток». Во время исполнения main-функции при вызове pthread_create определяется код функции потока, который распределяется в системе с помощью модуля балансировки нагрузки и исполняется в отдельном потоке на менее загруженном узле кластера.

С предварительными результатами экспериментальных исследований можно ознакомиться в [24].

5. Заключение

В работе представлено описание метода, который может быть использован для разработки алгоритмов синхронизации конкурентного выполнения набора распределенных асинхронно взаимосвязанных транзакций. Отличительной особенностью метода является его соответствие строгому критерию согласованности транзакций: критерию скрытности.

Литература

1. Herlihy M., Moss J. E. B. Transactional memory: architectural support for lock-free data structures // SIGARCH Comput. Archit. News. 1993. Vol. 21, No. 2. P. 289–300.
2. Lomet D. B. Process structuring, synchronization, and recovery using atomic actions // SIGOPS Oper. Syst. Rev. 1977. Vol. 11, No. 2. P. 128–137.
3. Liskov B., Scheifler R. Guardians and Actions: Linguistic Support for Robust, Distributed Programs // ACM Trans. Program. Lang. Syst. 1983. Vol. 5, No. 3. P. 381–404.
4. Hoare C. A. R. Monitors: an operating system structuring concept // Commun. ACM. 1974. Vol. 17, No. 10. P. 549–557.
5. Saad M. M., Ravindran B. HyFlow: a high performance distributed software transactional memory framework // Proceedings of the 20th international symposium on High performance distributed computing. HPDC '11. 2011. P. 265–266.
6. Saad M. M., Ravindran B. Transactional Forwarding Algorithm: Tech. rep.: ECE Dept., Virginia Tech, 2011.
7. Bocchino R. L., Adve V. S., Chamberlain B. L. Software transactional memory for large scale clusters // Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming. PPoPP '08. 2008. P. 247–258.
8. Тель Ж. Введение в распределенные алгоритмы. М.: Изд-во МЦНМО, 2009. 616 с.
9. Eswaran K. P., Gray J. N., Lorie R. A., Traiger I. L. The notions of consistency and predicate locks in a database system // Commun. ACM. 1976. Vol. 19, No. 11. P. 624–633.
10. Bernstein P. A., Goodman N. Concurrency Control in Distributed Database Systems // ACM Comput. Surv. 1981. Vol. 13, No. 2. P. 185–221.
11. Dice D., Shalev O., Shavit N. Transactional locking II // Proceedings of the 20th international conference on Distributed Computing. DISC'06. 2006. P. 194–208.
12. Guerraoui R., Kapalka M. On the correctness of transactional memory // Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming. PPoPP '08. 2008. P. 175–184.
13. Kung H. T., Robinson J. T. On optimistic methods for concurrency control // ACM Trans. Database Syst. 1981. Vol. 6, No. 2. P. 213–226.
14. Lamport L. Time, clocks, and the ordering of events in a distributed system // Commun. ACM. 1978. Vol. 21, No. 7. P. 558–565.
15. Zhang B., Ravindran B. Brief Announcement: Relay: A Cache-Coherence Protocol for Distributed Transactional Memory // Proceedings of the 13th International Conference on Principles of Distributed Systems. OPODIS '09. 2009. P. 48–53.
16. Sridharan S., Vetter J. S., Kogge P. M. Scalable Software Transactional Memory for Global Address Space Architectures: Tech. Rep. FTGTR-2009-04: Future Technologies Group, Oak Ridge National Lab, 2009.
17. Peluso S., Ruivo P., Romano P. et al. When Scalability Meets Consistency: Genuine Multiversion Update-Serializable Partial Data Replication // Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference. 2012. P. 455–465.

18. Peluso S., Romano P., Quaglia F. SCORe: a scalable one-copy serializable partial replication protocol // Proceedings of the 13th International Middleware Conference. Middleware '12. 2012. P. 456–475.

19. Mattern F. Virtual Time and Global States of Distributed Systems // Proc. Workshop on Parallel and Distributed Algorithms / Ed. by C. M. et al. North-Holland / Elsevier: 1989. P. 215–226.

20. Manassiev K., Mihailescu M., Amza C. Exploiting distributed version concurrency in a transactional memory cluster // Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming. PPoPP '06. 2006. P. 198–208.

21. Mishra S., Turcu A., Palmieri R., Ravindran B. HyflowCPP: A Distributed Transactional Memory Framework for C++ // 12th IEEE International Symposium on Network Computing and Applications. NCA 2013. Boston, USA: IEEE Computer Society, 2013.

22. Herlihy M., Sun Y. Distributed transactional memory for metric-space networks // Proceedings of the 19th international conference on Distributed Computing. DISC'05. 2005. P. 324–338.

23. Данилов И.Г. Прототип распределенной программной транзакционной памяти DSTM_P1 // Высокопроизводительные параллельные вычисления на кластерных системах. Материалы XI Всероссийской конференции (Н. Новгород, 2–3 ноября 2011 г.) / Под ред. проф. В.П. Гергеля. - Нижний Новгород: Изд-во Нижегородского госуниверситета. - 2011. - С. 102-107.

24. Данилов И.Г. Об одном подходе к реализации программной транзакционной памяти для распределённых вычислений // Известия ЮФУ. Технические науки. Тематический выпуск «Проблемы математического моделирования, супервычислений и информационных технологий». - Таганрог: Изд-во ТТИ ЮФУ, 2012. - № 6 (131), С. 91-95.

Данилов Игорь Геннадьевич, программист, Научно-исследовательский центр супер-ЭВМ и нейрокомпьютеров (Таганрог, Российская Федерация), vainamon@gmail.com.

Поступила в редакцию 4 августа 2014 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2014, vol. 3, no. 3, pp. 37–50

METHOD FOR DISTRIBUTED AND CONSISTENT TRANSACTIONS EXECUTION

I.G. Danilov, Supercomputers and Neurocomputers Research Center (Taganrog, Russia)

This paper proposes a method for detecting RW shared data conflicts, which encountered during execution of competitive set of distributed transactions. This method is designed to prevent runtime anomalies associated with this type of conflicts in distributed transactional memory systems.

Keywords: distributed transactional memory, partitioned global address space.

References

1. Herlihy M., Moss J. E. B. Transactional memory: architectural support for lock-free data structures // SIGARCH Comput. Archit. News. 1993. Vol. 21, No. 2. P. 289–300.
2. Lomet D. B. Process structuring, synchronization, and recovery using atomic actions // SIGOPS Oper. Syst. Rev. 1977. Vol. 11, No. 2. P. 128–137.
3. Liskov B., Scheifler R. Guardians and Actions: Linguistic Support for Robust, Distributed Programs // ACM Trans. Program. Lang. Syst. 1983. Vol. 5, No. 3. P. 381–404.
4. Hoare C. A. R. Monitors: an operating system structuring concept // Commun. ACM. 1974. Vol. 17, No. 10. P. 549–557.
5. Saad M. M., Ravindran B. HyFlow: a high performance distributed software transactional memory framework // Proceedings of the 20th international symposium on High performance distributed computing. HPDC '11. 2011. P. 265–266.
6. Saad M. M., Ravindran B. Transactional Forwarding Algorithm: Tech. rep.: ECE Dept., Virginia Tech, 2011.
7. Bocchino R. L., Adve V. S., Chamberlain B. L. Software transactional memory for large scale clusters // Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming. PPoPP '08. 2008. P. 247–258.
8. Tel G. Introduction to Distributed Algorithms. MCCME, Moscow. 2009.
9. Eswaran K. P., Gray J. N., Lorie R. A., Traiger I. L. The notions of consistency and predicate locks in a database system // Commun. ACM. 1976. Vol. 19, No. 11. P. 624–633.
10. Bernstein P. A., Goodman N. Concurrency Control in Distributed Database Systems // ACM Comput. Surv. 1981. Vol. 13, No. 2. P. 185–221.
11. Dice D., Shalev O., Shavit N. Transactional locking II // Proceedings of the 20th international conference on Distributed Computing. DISC'06. 2006. P. 194–208.
12. Guerraoui R., Kapalka M. On the correctness of transactional memory // Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming. PPoPP '08. 2008. P. 175–184.
13. Kung H. T., Robinson J. T. On optimistic methods for concurrency control // ACM Trans. Database Syst. 1981. Vol. 6, No. 2. P. 213–226.
14. Lamport L. Time, clocks, and the ordering of events in a distributed system // Commun. ACM. 1978. Vol. 21, No. 7. P. 558–565.
15. Zhang B., Ravindran B. Brief Announcement: Relay: A Cache-Coherence Protocol for Distributed Transactional Memory // Proceedings of the 13th International Conference on Principles of Distributed Systems. OPODIS '09. 2009. P. 48–53.
16. Sridharan S., Vetter J. S., Kogge P. M. Scalable Software Transactional Memory for Global Address Space Architectures: Tech. Rep. FTGTR-2009-04: Future Technologies Group, Oak Ridge National Lab, 2009.
17. Peluso S., Ruivo P., Romano P. et al. When Scalability Meets Consistency: Genuine Multiversion Update-Serializable Partial Data Replication // Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference. 2012. P. 455–465.

18. Peluso S., Romano P., Quaglia F. SCORe: a scalable one-copy serializable partial replication protocol // Proceedings of the 13th International Middleware Conference. Middleware '12. 2012. P. 456–475.

19. Mattern F. Virtual Time and Global States of Distributed Systems // Proc. Workshop on Parallel and Distributed Algorithms / Ed. by C. M. et al. North-Holland / Elsevier: 1989. P. 215–226.

20. Manassiev K., Mihailescu M., Amza C. Exploiting distributed version concurrency in a transactional memory cluster // Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming. PPoPP '06. 2006. P. 198–208.

21. Mishra S., Turcu A., Palmieri R., Ravindran B. HyflowCPP: A Distributed Transactional Memory Framework for C++ // 12th IEEE International Symposium on Network Computing and Applications. NCA 2013. Boston, USA: IEEE Computer Society, 2013.

22. Herlihy M., Sun Y. Distributed transactional memory for metric-space networks // Proceedings of the 19th international conference on Distributed Computing. DISC'05. 2005. P. 324–338.

23. Danilov I.G. DSTM_P1: Distributed transactional memory prototype // 11th Conference on high performance computing. HPC 2011. Nizhni Novgorod, Russia, 2011. P. 102-107.

24. Danilov I.G. On one approach to implement software transactional memory for distributed computing. IZVESTIYA SFedU. Series: Engineering sciences. 2012. No. 6(131). P. 91-95.

Received 4 August 2014.

УДК 004.75, 519.172

ОБЗОР ТЕХНОЛОГИЙ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ

А.В. Шамакина

В данной работе представлен обзор технологий распределенных вычислений с точки зрения организации планирования заданий в грид и облачных средах. Рассмотрена архитектура платформы UNICORE, ориентированной на обеспечение прозрачного безопасного доступа к ресурсам распределенной вычислительной среды. Приведена общая классификация алгоритмов планирования в распределенных вычислительных средах, а также классификация алгоритмов планирования для заданий, имеющих зависимости между задачами.

Ключевые слова: распределенные вычисления, проблемно-ориентированные распределенные вычислительные среды, грид, UNICORE, алгоритмы планирования ресурсов, алгоритмы кластеризации, ориентированный ациклический граф, потоки работ.

Введение

Развитие технологий распределенных вычислений в конце 1990-х годов позволило объединить географически-распределенные по всему миру гетерогенные ресурсы. Появились технические возможности для решения масштабных задач в области науки, техники и коммерции на территориально-распределенных ресурсах, принадлежащих разным владельцам. Исследования данной тематики привело к возникновению концепции грид вычислений (grid computing) [1, 2–4], и затем — к новой концепции облачных вычислений (cloud computing) [5–8]. Для раскрытия всех потенциальных возможностей использования распределенных вычислительных ресурсов принципиально важно наличие результативных и эффективных алгоритмов планирования, используемых менеджерами ресурсов.

Управление ресурсами в традиционных гомогенных многопроцессорных системах (вычислительных кластерах) — хорошо изученный и проработанный вопрос. Существует большое количество менеджеров ресурсов для подобных систем [9]. Менеджеры ресурсов включаются в пакетные планировщики, в инструментарий для управления очередями заданий, в операционные системы. Эти менеджеры являются локальными, имеют полный контроль над ресурсами и реализуют механизмы и политики для эффективного использования данных изолированных ресурсов. Алгоритмы планирования для изолированных гомогенных многопроцессорных систем не могут также хорошо работать в распределенных вычислительных средах [4].

Главной задачей, которую решают технологии распределенных вычислений, является обеспечение доступа к глобально распределенным ресурсам с помощью специального инструментария. Сложность управления глобальными ресурсами заключается в том, что запуск, выполнение работы и доступ к необходимым данным могут производиться на различных компьютерах. Глобальные распределенные вычислительные сети формируются из автономных ресурсов, конфигурация которых динамически изменяется. Кроме того, распределенные ресурсы могут принадлежать различным административным доменам, поэтому возникает проблема их администрирования, заключающаяся в согласовании различных политик. Еще одной немаловажной проблемой является гетерогенность ресурсов. Ранние работы [6, 10–12] в области управления ресурсами в распределенных вычислительных средах, фокусирующиеся на гетерогенности ресурсов, привели к созданию

стандартных протоколов управления ресурсами и механизмов описания требований заданий к ресурсам. Однако практика показала, что эффективные методы и алгоритмы планирования для однородных изолированных многопроцессорных систем плохо адаптируются для распределенных гетерогенных систем [13]. Управление ресурсами в неоднородных распределенных вычислительных средах требует принципиально новых моделей вычислений и управления ресурсами.

В настоящее время перспективным является направление, связанное с применением распределенных вычислительных технологий для решения ресурсоемких научных задач в разных предметных областях: медицине, инженерном проектировании, нанотехнологиях, прогнозировании климата и др. Вычислительное задание в подобных предметных областях во многих случаях имеют потоковую структуру и могут быть описаны с помощью модели потока работ (workflow) [14], в соответствии с которой задание представляется в виде ориентированного ациклического графа, узлами которого являются задачи, являющиеся составными частями задания, а дуги соответствуют потокам данных, передаваемых между отдельными задачами. При этом набор задач, из которых строятся задания, является конечным и предопределенным. Проблемно-ориентированная специфика потоков работ в подобных сложных приложениях выражается в том, что в подавляющем большинстве случаев, еще до выполнения задания, для каждой задачи могут быть получены оценки таких качественных характеристик, как время выполнения задачи на одном процессорном ядре, пределы масштабируемости и объем генерируемых данных. Использование подобных знаний о специфике задач в конкретной проблемно-ориентированной области может существенно улучшить эффективность методов управления вычислительными ресурсами. В настоящее время известно несколько программных систем, ориентированных на управление сложными приложениями с потоковой структурой в распределенных вычислительных средах. В качестве примера можно перечислить такие инструменты как Condor DAGMan [15], CoG [16], Pegasus [17], GridFlow [18] и ASKALON [19]. Существуют также алгоритмы планирования, использующие знания о проблемно-ориентированной специфике задач, составляющих вычислительное задание, такие как алгоритм Кима и Брауна [20], алгоритм DSC [21]. Однако данный класс алгоритмов применим для задач, выполняющихся на одном процессорном ядре некоторой многопроцессорной системы. В соответствие с этим актуальной является задача разработки методов и алгоритмов управления ресурсами в проблемно-ориентированных распределенных вычислительных средах, учитывающих специфику предметной области, масштабируемость отдельных задач в задании и использующих возможность параллельного выполнения независимых задач.

Статья организована следующим образом. В разделе 1 рассматриваются основные технологии распределенных вычислений: грид-вычисления и облачные вычисления. Раздел 2 посвящен обзору алгоритмов планирования ресурсов в распределенных вычислительных средах. В заключении изложены выводы, основанные на данном обзоре.

1. Технологии распределенных вычислений

1.1. Грид-вычисления

Вычислительный грид является программно-аппаратной инфраструктурой, которая обеспечивает надежный и прозрачный доступ к высокопроизводительным вычислительным ресурсам [4]. Грид представляет общую среду для развертывания инфраструктуры,

ориентированной на сервисы, поддерживающей создание и совместное использование ресурсов распределенных организаций. Под ресурсами понимаются аппаратное обеспечение, инструментарий, программное обеспечение и данные, а также сервисы, подключенные посредством промежуточного слоя программного обеспечения и обеспечивающие безопасность, мониторинг, управление ресурсами и др.

При рассмотрении проблемы планирования в грид-средах для повышения уровня абстракции часто игнорируют такие компоненты инфраструктуры, как аутентификация, авторизация, обнаружение ресурсов и контроль доступа. В работе [22] приводится следующее адаптированное определение: «Грид — это тип параллельных и распределенных систем, которые обеспечивают совместное использование, выбор и динамическую агрегацию географически-распределенных автономных и гетерогенных ресурсов в зависимости от их доступности, характеристик, производительности, стоимости и требований качества обслуживания, предъявляемых пользователями».

С точки зрения функциональности можно выделить логическую архитектуру подсистемы планирования заданий в грид. В работе [13] предложена общая архитектура данной подсистемы. Процесс планирования в распределенных вычислительных сетях может быть представлен тремя этапами: 1) обнаружение ресурсов и их фильтрация, 2) поиск подходящих ресурсов и планирование в соответствии с определенными целями, 3) выполнение задания [23]. На рис. 1 представлена модель системы планирования в распределенных вычислительных средах, в которой функциональные компоненты связывают два типа потоков данных: прерывистая линия определяет поток ресурсов/поток информации о приложении, прямая линия — поток заданий/поток команд планирования заданий.

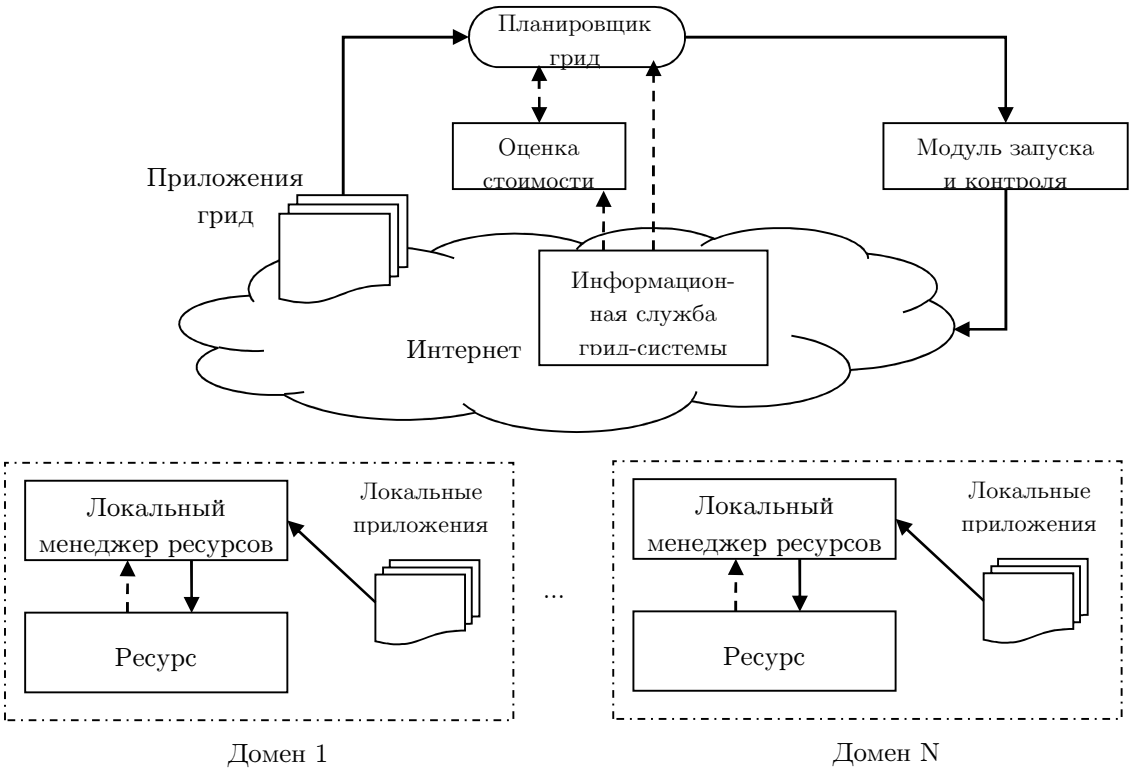


Рис. 1. Логическая архитектура системы планирования в грид

Основная работа *планировщика грид* (GS — Grid Scheduler) заключается в том, что он принимает заявки на выполнение некоторых приложений пользователей, производит поиск подходящих ресурсов в соответствии с полученными от *информационного сервиса*

грид-среды (GIS — Grid Information Service) данными и создает отображение приложений на ресурсы, основанное на целевых функциях и предсказании характеристик ресурсов. В отличие от планировщиков традиционных параллельных и распределенных систем, грид-планировщики обычно не контролируют грид-ресурсы напрямую, а работают как брокеры или агенты [24]. Грид-планировщики не всегда находятся в одном домене с ресурсами, которые им доступны. На рис. 1 показан только один грид-планировщик, однако в действительности может быть развернуто несколько подобных планировщиков для формирования различных структур системы планирования (централизованной, иерархической и децентрализованной [25]) для решения проблем производительности и масштабируемости. Хотя планировщик уровня грид (иначе его называют метапланировщиком [26]) не является необходимым компонентом в инфраструктуре грид-среды (он не включен в Globus Toolkit [27]), но он имеет решающее значение для использования в грид-средах, которые имеют тенденцию быстро расширяться, добавляя в свой состав все новые и новые ресурсы от суперкомпьютеров до настольных компьютеров.

Информация о состоянии имеющихся ресурсов необходима грид-планировщику для составления приемлемого расписания работ, особенно в условиях гетерогенного и динамичного характера грид. Расписание часто представляется в виде диаграммы Ганта, горизонтальная ось которой представляет собой время, вертикальная ось — ресурсы в грид. На каждом из ресурсов прямоугольниками обозначаются задачи, ширина прямоугольника показывает длительность задачи. Роль информационного сервиса грид-среды заключается в предоставлении информации о состоянии имеющихся ресурсов планировщикам грид. Информационный сервис ответственен за сбор и предсказание информации о состоянии ресурса. Информационный сервис грид-среды также может отвечать на запросы, предоставляя информацию о ресурсе, или передавать информацию подписчикам. Примерами информационных сервисов в грид являются Globus Monitoring и Discovery System (MDS) [28].

Для составления приемлемого расписания помимо информации о ресурсе необходимо знать свойства приложений (приблизительное количество инструкций, требования к памяти и хранению, зависимости подзадач в задании) и производительность ресурсов для различных видов приложений. Информация о свойствах приложения может быть получена с помощью *профилирования* (AP — Application profiling), а измерение производительности ресурса для данного типа задания — с помощью *компоненты тестирования* (AB — Analogical Benchmarking) [29, 30]. На основе информации, полученной при профилировании приложений, и информации от компоненты тестирования, а также используемой модели производительности [4], производится оценка стоимости планирования узлов-кандидатов на выполнение приложения, из которых планировщик выбирает те, которые оптимизируют целевые функции.

Модуль запуска и контроля (LM — Launching and Monitoring) формирует окончательное расписание, предоставляет приложениям соответствующие ресурсы, поставляет входные данные и исполняемые файлы, если это необходимо, и выполняет мониторинг исполнения приложений. Модуль запуска и контроля иногда называют *компановщиком* [31]. Примером модуля запуска и контроля является Globus GRAM (Grid Resource Allocation and Management) [32].

Локальный менеджер ресурсов (LRM — Local Resource Manager) несет основную ответственность за составление локального расписания внутри домена, где присутствуют

задания не только от внешних пользователей грид, но и выполняются задания локальных пользователей домена, а также предоставляет отчетную информацию для информационного сервиса грид-среды. Внутри домена могут работать сразу несколько локальных планировщиков каждый со своей локальной политикой управления ресурсами. В качестве примеров подобных локальных планировщиков можно привести OpenPBS [33] и Condor [34]. LRM собирает информацию о локальных ресурсах с помощью таких инструментальных средств, как Network Weather Service [35], Hawkeye [34] и Ganglia [36], и формирует отчет с информацией о состоянии ресурсов для информационного сервиса.

1.2. Облачные вычисления

Концепция предоставления вычислительных ресурсов, названная облачными вычислениями (cloud computing), сформировалась в 2008 г. В 2011 г. Национальный институт стандартов и технологий США (The National Institute of Standards and Technology, NIST) опубликовал 16-е, окончательное определение данного понятия. Согласно NIST, *облачные вычисления* — это модель обеспечения удобного повсеместного сетевого доступа по требованию к совместно используемому пулу конфигурируемых вычислительных ресурсов, которые можно быстро предоставить и внедрить с минимумом административных усилий или взаимодействия с сервис-провайдером [37].

NIST зафиксированы следующие пять обязательных характеристик облачных вычислений:

- самообслуживание по требованию;
- универсальный доступ по сети;
- объединение ресурсов;
- эластичность;
- учет потребления.

Существует три основных модели обслуживания облачных вычислений:

- программное обеспечение как услуга (SaaS — Software-as-a-Service),
- платформа как услуга (PaaS — Platform-as-a-Service),
- инфраструктура как услуга (IaaS — Infrastructure-as-a-Service),

а также появляются дополнительные модели:

- аппаратное обеспечение как услуга (HaaS — Hardware as a Service),
- безопасность как сервис (SECaaS — Security as a Service),
- данные как услуга (DaaS — Data as a Service).

Модель обслуживания SaaS предоставляется возможность использования прикладного программного обеспечения провайдера, работающего в облачной инфраструктуре и доступного из различных клиентских устройств или посредством тонкого клиента, например, из браузера (например, почта Gmail) или интерфейса программы.

Модель обслуживания PaaS предоставляет потребителю возможность использования облачной инфраструктуры для размещения базового программного обеспечения для последующего размещения на нем новых или существующих приложений. В состав платформ входят инструментальные средства создания, тестирования и выполнения прикладного программного обеспечения, предоставляемые облачным провайдером. Примерами подобных платформ являются Google App Engine [8] и Windows Azure [5].

Модель обслуживания IaaS предоставляет возможность использования облачной инфраструктуры для самостоятельного управления ресурсами обработки, хранения, сетей и

другими фундаментальными вычислительными ресурсами. Примером облаков, предоставляющих инфраструктуру как услугу, является Nimbus [6].

Эталонная архитектура облачных вычислений NIST содержит пять главных действующих субъектов – *актеров (actors)*. Каждый актер выступает в некоторой *роли (role)* и выполняет *действия (activities)* и *функции (functions)*. Эталонная архитектура представляется в виде последовательности диаграмм с увеличивающимся уровнем детализации. Виды актеров облачных вычислений представлены в таблице.

Таблица

Виды узлов логического плана решения задач

Актер	**Определение**
Облачный потребитель Cloud Consumer	Лицо или организация, поддерживающая бизнес-отношения и использующая услуги *Облачных провайдеров.*
Облачный провайдер Cloud Provider	Лицо, организация или сущность, отвечающая за доступность облачной услуги для *Облачных потребителей.*
Облачный аудитор Cloud Auditor	Участник, который может выполнять независимую оценку облачных услуг, обслуживания информационных систем, производительности и безопасности реализации облака.
Облачный брокер Cloud Broker	Сущность, управляющая использованием, производительностью и предоставлением облачных услуг, а также устанавливающая отношения между *Облачными провайдерами* и *Облачными потребителями.*
Облачный оператор связи Cloud Carrier	Посредник, предоставляющий услуги подключения и транспорт (услуги связи) для доставки облачных услуг от *Облачных провайдеров* к *облачным потребителям.*

Обобщенная облачная среда содержит три концептуальных уровня.

– *Уровень сервиса (Service Layer)* определяет базовые сервисы, предоставляемые облачным провайдером.
– *Уровень абстракции и контроля ресурсов (Resource Abstraction and Control Level)* назначает/предоставляет элементы программного обеспечения, такие как гипервизор, виртуальные хранилища данных и поддерживающие программные компоненты, используемые для реализации облачной инфраструктуры, поверх которой может быть определен/установлен облачный сервис. Кроме того, данный уровень назначает/предоставляет ассоциированные функциональные модули, которые управляют абстрагированными ресурсами для обеспечения эффективного, безопасного и надежного использования.
– *Уровень физических ресурсов (Physical Resource Level)* включает все физические ресурсы: компьютерное оборудование и инженерную инфраструктуру.

На рис. 2 представлена концептуальная диаграмма эталонной архитектуры облачных вычислений согласно NIST.

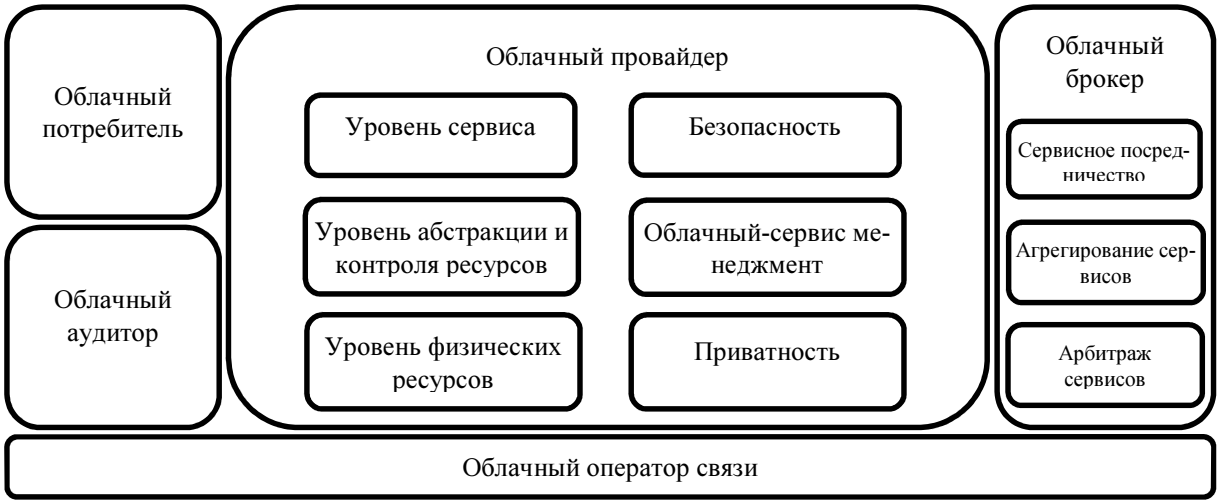


Рис. 2. Концептуальная диаграмма эталонной архитектуры облачных вычислений

1.3. Платформа UNICORE

Проект UNICORE (Uniform Interface to Computing Resources — единый интерфейс к вычислительным ресурсам) появился в 1997 году, и к настоящему моменту представляет собой комплексное решение, ориентированное на обеспечение прозрачного безопасного доступа к ресурсам распределенной вычислительной среды [38].

Архитектура UNICORE 6 [39] формируется из пользовательского, сервисного и системного слоев (рис. 3).

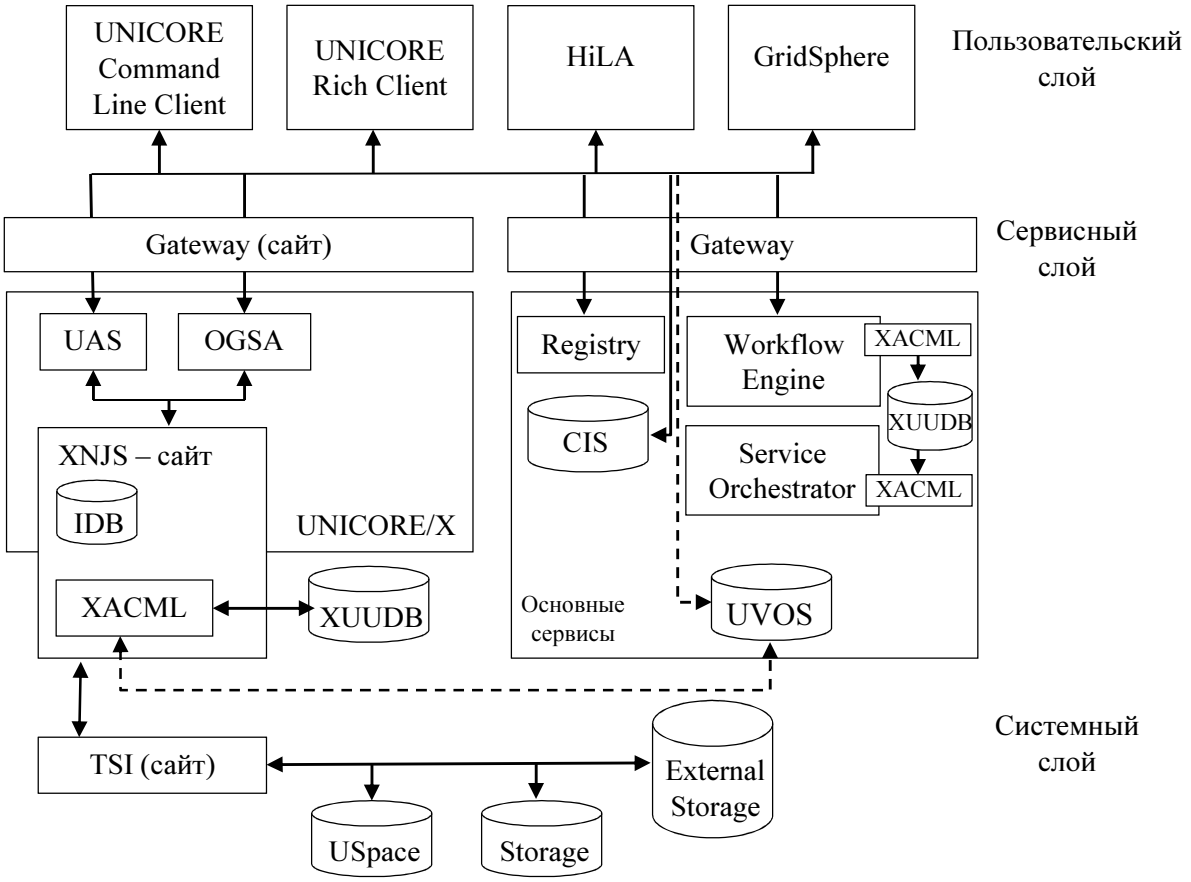


Рис. 3. Архитектура UNICORE 6

Верхним слоем в архитектуре является *пользовательский слой*. В нем располагаются различные клиенты, обеспечивающие взаимодействие пользователей с распределенной вычислительной средой:

На пользовательском уровне доступ организуется с использованием графического интерфейса и интерфейса командой строки. Задания могут быть запущены на любой из платформ UNICORE в распределенной вычислительной среде. Пользователь может осуществлять мониторинг и управление запущенными заданиями, используя часть интерфейса, называемую *монитором заданий*. Сейчас UNICORE позволяет использовать различные клиенты по требованию: Command Line Client (UCC), Eclipse-based UNICORE Rich Client (URC), High Level API for Grid Applications (HiLA) и Portal Clients (например, GridSphere).

Средний слой или слой сервисов включает все сервисы и компоненты сервис-ориентированной архитектуры UNICORE, основанные на WS-RF 1.2 и SOAP, а также на стандартах WS-I.

Компонент *Gateway* является точкой входа на сайт UNICORE и выполняет аутентификацию всех входящих запросов.

Сервер *UNICORE/X* является главным компонентом сайта UNICORE 6. Посредством WSRF сервер UNICORE/X обеспечивает доступ к ресурсам хранилища, сервису передачи файлов и сервисам выполнения заданиями и мониторинга.

Набор собственных интерфейсов веб-сервисов называется *UNICORE Atomic Services (UAS)*. UAS предоставляет набор базовых услуг для сервисов более высокого уровня, клиентов и пользователей.

Компонент, отвечающий за механизм выполнения заданий в UNICORE, называется *XNJS*. Он обеспечивает ресурсы хранения, сервисы передачи файла и сервисы управления заданием.

База данных *IDB (Incarnation Data Base)* используется для отображения описания абстрактного задания на языке JSDL (Job submission description language) в описание конкретного задания для определенного ресурса. Информация о доступных приложениях и характеристиках ресурсов должна быть определена в базе данных IDB.

Компонент *XUUDB* представляет собой веб-сервис и базу данных для отображения сертификатов X.509 на логины фактических пользователей и их роли. Управление доступом основывается на политике XACML.

Компонент *UNICORE VO Service (UVOS)* используется в качестве альтернативы XUUDB, а также для авторизации пользователей с помощью стандарта SAML.

Компонент *Registry* предназначен для регистрации сервисов, доступных в UNICORE. Единая сервисная регистрация необходима для построения распределенной инфраструктуры UNICORE и управления ей.

Common Information Service (CIS) является информационным сервисом UNICORE. CIS собирает статическую и динамическую информацию от всех компонентов XNJS, которые с ним связаны.

Компонент *Workflow Engine* обеспечивает выполнение потока работ. Поток работ создается/запускается с помощью графического интерфейса UNICORE Rich Client или из командной строки Command Line Client.

Компонент *Service Orchestrator* отвечает за выполнение отдельных задач в потоке работ, обработку выполнения задач и мониторинг в распределенной вычислительной

среде. В Service Orchestrator реализованы различные стратегии планирования ресурсов. Существует возможность подключения пользовательских стратегий.

В основании архитектуры платформы UNICORE находится *системный слой*. Компонент *Target System Interface (TSI)* обеспечивает взаимодействие между UNICORE и отдельным ресурсом распределенной вычислительной сети. TSI обеспечивает трансляцию команд, поступающих из распределенной вычислительной среды, в команды для локальной системы.

Целевая система (Target System, TS) — это совокупность программного и аппаратного обеспечения, доступного в распределенной вычислительной среде. Описание приложений и аппаратных ресурсов хранится в файле simpleidb директории TSI.

USpace представляет собой директорию для хранения заданий пользователей. Существует отдельные директории для каждого задания, в которых XNJS и TSI хранят исходные данные, стандартный вывод и стандартный поток ошибок. Для передачи данных между сайтами, например, для передачи данных с/на внешние хранилища, используется протокол GridFTP.

External Storage служит для передачи данных между сайтами. Для работы с внешними хранилищами компонент External Storage использует протокол GridFTP.

UNICORE 6 позволяет использовать *потоки работ* и поддерживает следующие возможности:

- представление задания в виде ориентированного графа;
- использование переменных в потоках работ;
- использование циклов и управляющих конструкций:
  - while, for-each, if-else;
- использование в потоках работ следующих условий:
  - код выхода;
  - существование файла;
  - размер файла.

Основным достоинством использования платформы UNICORE 6 для разработки распределенных вычислительных систем является наличие большого количества различных клиентов, обеспечивающих взаимодействие пользователя с ресурсами распределенной вычислительной сети, а также развитых средств обеспечения безопасности при разработке приложений в распределенных вычислительных средах.

2. Обзор алгоритмов планирования ресурсами в распределенных вычислительных средах

2.1. Общая классификация алгоритмов планирования ресурсами

Существующие алгоритмы планирования ресурсов в распределенных вычислительных средах могут быть классифицированы по разным признакам: с точки зрения архитектуры компонентов, участвующих в планировании; используемых политик; целевых функций; моделей приложений; ограничений качества обслуживания (QoS — Quality of Service); стратегий, применяемых для ресурсов с динамическим поведением и др. [9].

На рис. 4 представлена иерархическая классификация алгоритмов планирования общего назначения для параллельных и распределенных вычислительных систем [40].

На верхнем уровне иерархической классификации выделяют *статические и динамические алгоритмы* планирования ресурсов. Статическое планирование и расчет стоимостной оценки вычислений осуществляется до начала выполнения задания, когда информация относительно всех ресурсов в распределенных вычислительных средах и всех задачах задания уже доступна [41–43]. Одно из главных преимуществ статической модели — это простота реализации планировщика. Однако, стоимостная оценка, основанная на статической информации, не адаптивна к ситуациям, когда один из вычислительных узлов выходит из строя, становится изолированным для системы из-за сетевого отказа или из-за высокой загрузки на систему время отклика становится более длительным, чем ожидалось. Для решения проблемы используют вспомогательные механизмы, такие как механизм перепланирования [31].

Динамическое планирование обычно применяется, когда трудно оценить вычислительную стоимость приложений, поступающих на выполнение динамически в режиме online [44–47]. Примером использования динамического планирования является управление очередью заданий в системах метавычислений Condor [48] и Legion [49]. Динамическое планирование задач включает в себя два важных компонента: оценка состояния системы и принятие решения о связывании задачи из очереди с выбранным ресурсом [50]. Для сохранения оптимального состояния вычислительной системы используется балансировка загрузки всех ее ресурсов. Преимущество динамической балансировки загрузки над статической состоит в том, что система не обязана знать о поведении приложения во время его выполнения до его запуска. Особенно это подход полезен в системе, где основной целью является максимизация утилизации ресурса, а не минимизация времени выполнения отдельных заданий [51].

Динамические алгоритмы планирования, описанные в работах [26] и [52], рассматривают случай резервирования ресурса, которое популярно в распределенных вычислениях. Резервирование ресурсов используется для получения некоторой степени уверенности в производительности ресурса. Алгоритмы в этих двух работах стремятся минимизировать время исполнения входящих заданий, которые состоят из набора задач.

В динамических сценариях планирования ответственность за принятие глобальных решений планирования ресурсов может лежать на одном централизованном планировщике или нескольких распределенных планировщиках. *Централизованная стратегия* имеет преимущество, заключающееся в простоте реализации, но она плохо масштабируется, не отказоустойчива и часто становится узким местом для производительности системы. Например, в работе [53] предлагается централизованный метапланировщик, который использует алгоритм обратного заполнения (Backfill) для планирования параллельных заданий на сложных гетерогенных сайтах. Аналогично, в работе [54] представлен полностью *децентрализованный, динамический и инициируемый отправителем алгоритм* планирования и балансировки загрузки для распределенных вычислительных сред. Главное свойство этого алгоритма заключается в том, что он использует интеллектуальную стратегию поиска узлов-партнеров, на которые могут быть перенесены задачи.

В том случае, когда вся информация относительно состояния ресурсов и заданий известна, *оптимальная привязка* заданий к ресурсам может быть сделана на основании некоторой целевой функции, такой как минимизация времени выполнения заданий и максимальная утилизация ресурсов.

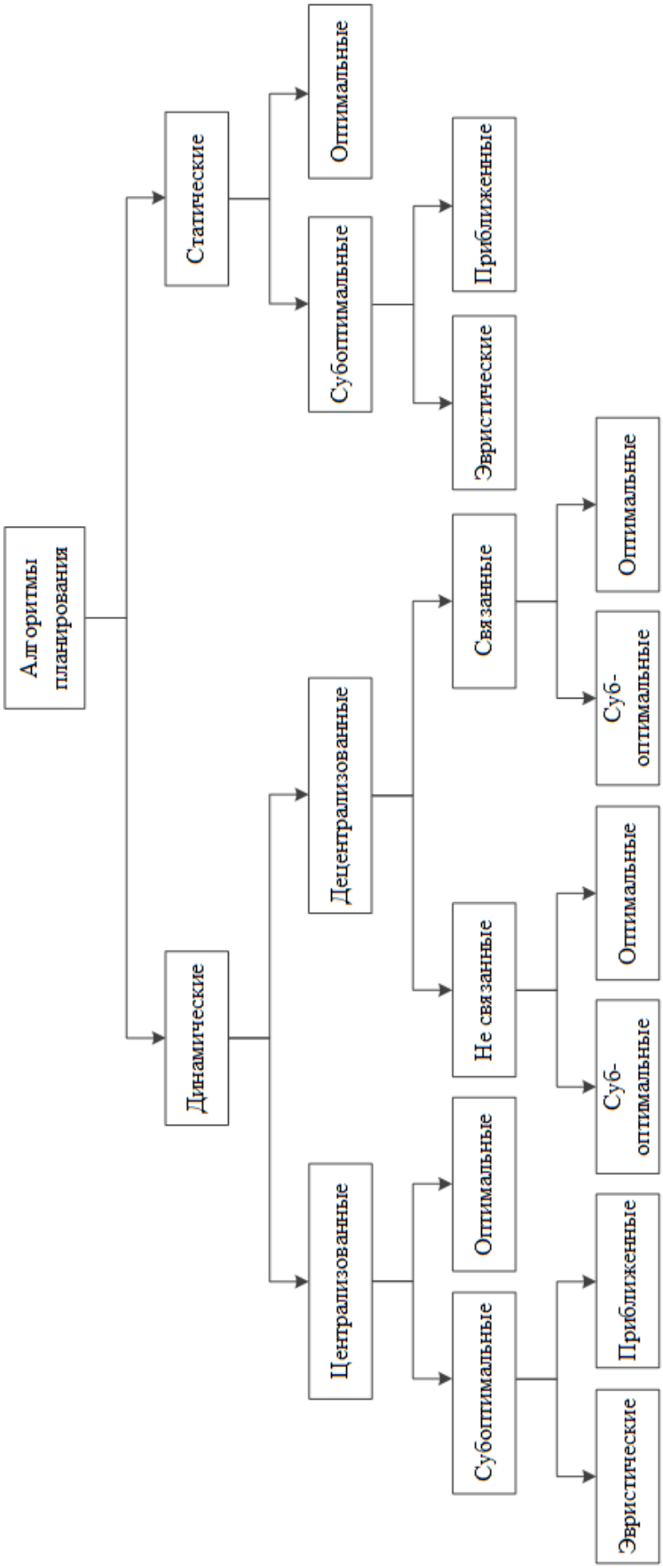


Рис. 4. Иерархическая классификация алгоритмов планирования

Однако, доказать оптимальность алгоритма или сделать некоторые разумные предположения об оптимальности представляется невозможным из-за того, что общая задача планирования является NP-полной [55]. Текущие исследования пытаются найти *субоптимальные решения*, которые могут быть далее разделены на следующие основные категории. *Приближенные алгоритмы* используют формальные вычислительные модели, вместо поиска всего пространства решений и выбора из него оптимального решения. Данные алгоритмы осуществляют поиск приемлемого решения, близкого к оптимальному. В случае, когда существует метрика пригодная для оценки решения, этот метод может использоваться для уменьшения времени, потраченного на поиск приемлемого расписания.

Эвристики — представляют собой класс алгоритмов, которые делают наиболее реалистичные предположения об априорном знании относительно характеристик загрузки системы и выполнения. Оценка этого вида решения обычно основана на экспериментах в реальном мире или на моделировании. Наиболее популярными в настоящее время являются экономические подходы [56–62] и эвристики, основанные на природных явлениях: генетический алгоритм (GA — Genetic Algorithm) [41, 63–65], моделируемый отжиг (SA — Simulated Annealing) [41, 62, 66,], запрещенный поиск (TS — Tabu Search) [41] и комбинированная эвристика [67].

Распределенные алгоритмы планирования можно разделить на *связанные или несвязанные*, в зависимости от того, как работают узлы, использующиеся при планировании заданий, совместно или независимо (несовместно). В несовместном случае локальные планировщики действуют как автономные сущности и принимают решения, с учетом их собственных целевых функций. В совместном случае каждый планировщик в распределенной вычислительной среде несет ответственность за выполнение его собственной части задачи планирования, но при этом все планировщики работают с одной общей целью в масштабе всей системы [68].

2.2. Классификация на основе зависимости задач в задании

При рассмотрении отношений между задачами в задании обычно используют следующую дихотомию: есть ли между ними зависимость или же задачи независимы [9]. Зависимость означает, что у задач есть приоритеты, то есть, задача не может запуститься до того, как были выполнены все ее родителя. Зависимость оказывает решающее влияние на разработку алгоритмов планирования. На рис. 5 представлена классификация алгоритмов планирования, основанная на наличии/отсутствии связей между задачами.

Главная *стратегия планирования независимых задач* заключается в назначении независимых задач на ресурсы в зависимости от их загрузки с целью обеспечения высокой пропускной способности вычислительной системы. Примерами статических алгоритмов с оценкой производительности являются: алгоритм минимального времени выполнения (MET — Minimum Execution Time), алгоритм минимального времени завершения (MCT — Minimum Completion Time), эвристики Min-Min [69, 70] и Max-min, Suffrage [12] и XSuffrage [42]. Данные алгоритмы обычно используются для планирования заданий, которые состоят из множества независимых задач с большими модулями и интенсивными вычислениями. Muthuvelu и др. [47] предлагают динамический алгоритм планирования сгруппированных задач, который позволяет уменьшить среднее значение издержек при планировании и запуске задания и увеличить использование ресурса.

В гетерогенных средах на производительность вышеупомянутых алгоритмов также влияет уровень неоднородности задач и ресурсов. Исследование в работе [12] показало,

что не существует ни одного алгоритма, который бы имел преимущества во всех аспектах. Для достижения максимальной высокой производительности в вычислительном грид планировщик должен уметь адаптироваться под неоднородные приложения/ресурсы.

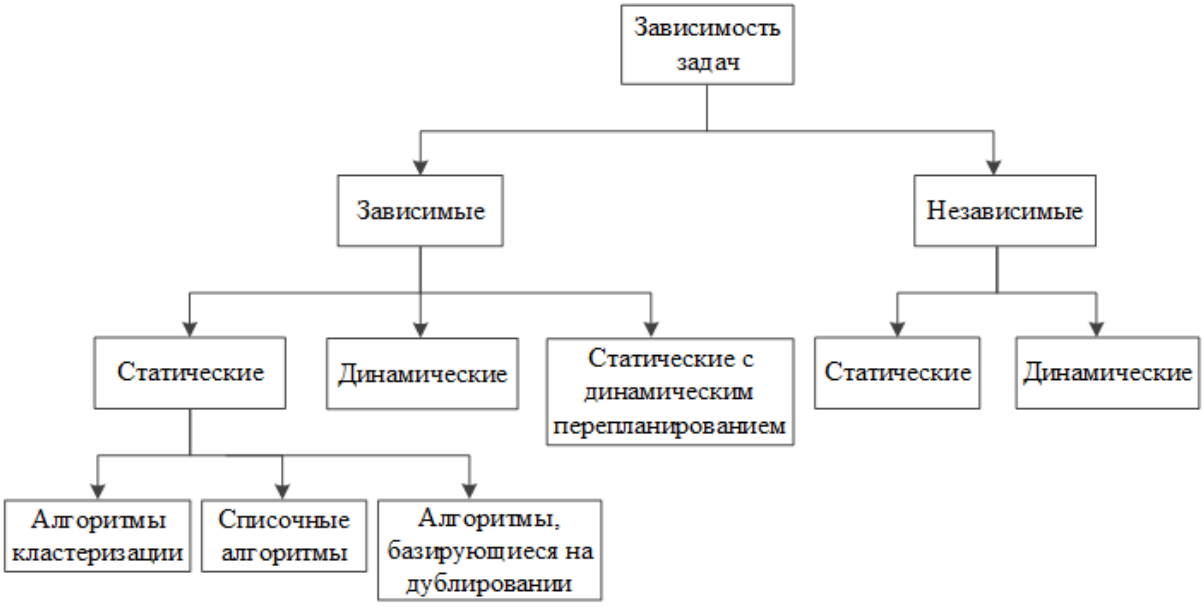


Рис. 5. Классификация зависимых и независимых задач в алгоритмах планирования ресурсами

Алгоритмы, представленные выше, предсказывают производительность для отображения задачи на ресурс. В [71] и [72] предложены два алгоритма, которые не используют оценку производительности, но основываются на идее дублирования, которое выполнимо в распределенной вычислительной среде, где вычислительные ресурсы присутствуют в достаточном количестве, но непостоянны.

В случае *планирования задач, имеющих зависимости*, задание обычно представляется в виде ориентированного ациклического графа, в котором каждая вершина представляет собой задачу, ориентированное ребро обозначает порядок приоритета между двумя вершинами. В некоторых случаях к вершинам и ребрам могут быть добавлены веса, показывающие вычислительную стоимость и коммуникационную стоимость соответственно.

Важнейшей проблемой при планировании заданий с потоковой структурой является нахождение компромисса между использованием максимального параллелизма задач в задании и минимизации коммуникационных задержек. Для решения данной проблемы (также известной как проблема максимина [55]) в гетерогенных вычислительных системах ранее были предложены три вида эвристических алгоритмов: эвристики списка [73, 74]; алгоритмы, базирующиеся на дублировании; алгоритмы кластеризации (рис. 5).

Планирование списком — это класс эвристик планирования, в котором задачам присваиваются приоритеты, задачи помещаются в список, упорядоченный по мере уменьшения величины приоритета. Решение о выборе задачи из списка для ее выполнения осуществляется на основе приоритета. Сначала осуществляется привязка к ресурсам задач с высоким приоритетом [55]. Классическими примерами эвристик списка являются HEFT (Heterogeneous Earliest-Finish-Time) [75] и FCP (Fast Critical Path) [73].

Критической проблемой в эвристике списка для DAG является вычисление ранга узла. Собственные значения, используемые для принятия решения об упорядочивании, могут быть усредненными значениями (как в оригинальном алгоритме HEFT в [75]), средним значением [11], худшем значением, оптимальным значением и так далее. Zhao и др. [76] показали, что различные варианты могут влиять на производительность эвристики списка, которые существенны для HEFT (время исполнения может изменяться на 47,2 % для определенного графа). Sakellariou и др. [77] предложил гибридный алгоритм, который менее чувствителен к разным подходам для ранжирования узлов.

Предыдущие алгоритмы планирования для ориентированного ациклического графа были рассмотрены применительно к использованию в распределенных вычислительных средах. Списочный алгоритм планирования, предложенный в работе [43], подобен алгоритму HEFT, но модифицирует его для вычисления уровня задачи в графе и учитывает входящую коммуникационную стоимость его родительских задач. В работе [78] Ma и др. предлагают новый списочный алгоритм для распределенных вычислительных сред под названием «расширенный динамический предельный путь» (xDCP — Extended Dynamic Critical Path), который представляет собой модификацию алгоритма DCP для однородной среды.

Альтернативным способом сокращения времени исполнения является копирование задачи на различные ресурсы. Основная идея дублирования состоит в том, что планирование использует время простоя ресурсов для копирования предшествующей задачи. Это позволяет избежать передачи результатов от предшествующей задачи последующей, таким образом, уменьшается коммуникационная стоимость. Дублирование также может решить проблему максимина.

Алгоритмы, базирующиеся на дублировании, отличаются стратегиями выбора задач для дублирования. Первоначально, алгоритмы этой группы применялись для неограниченного числа идентичных процессоров, таких как многопроцессорные системы с распределенной памятью. Также они имеют более высокую сложность, чем алгоритмы, осаждавшиеся выше. Например, Darbha и др. [79] представляют алгоритм под названием «алгоритм планирования, основанный на дублировании задач» (TDS — Task Duplication-based Scheduling Algorithm) для машин с распределенной памятью, имеющий сложность $O(v^2)$ для гомогенных сред.

Для применения дублирования в гетерогенных средах был предложен новый алгоритм под названием «алгоритм планирования, основанный на дублировании задач, для гетерогенных систем» (TANH — Task duplication-based scheduling Algorithm for Network of Heterogeneous systems), который представлен в работах [80] и [81]. Дублирование уже получило некоторое признание [71, 72], но на сегодняшний день алгоритмы, основанные на дублировании, в распределенных вычислительных средах имеют дело только с независимыми заданиями.

Применение *алгоритмов кластеризации* в параллельных и распределенных системах — это эффективный способ уменьшить коммуникационную задержку в ориентированном ациклическом графе. Основная идея данных алгоритмов заключается в кластеризации взаимосвязанных задач в маркированные группы для дальнейшего их присвоения некоторой группе ресурсов. Примерами алгоритмов кластеризации являются DSC (Dominant Sequence Clustering) [21], CASS-II [83]. Экспериментальные исследования алгоритмов кластеризации приводятся в работах [83, 84].

Рассмотрим более подробно следующие алгоритмы кластеризации: алгоритм KB/L, алгоритм Саркара, алгоритм DSC и алгоритм POS.

Алгоритм KB/L

В работах [20, 85] рассматривается алгоритм линейной кластеризации Кима и Брауна, также называемый алгоритмом KB/L. Алгоритм KB/L предназначен для кластеризации графа задания и предполагает, что количество вычислительных узлов неограниченно.

Первоначально все дуги графа задания маркируются как «не рассмотренные». На первом шаге выполнения алгоритма KB/L происходит поиск критического пути графа задания, который включает только «нерассмотренные» дуги, с помощью стоимостной функции веса (1). Все вершины найденного критического пути объединяются в один кластер, коммуникационные стоимости дуг обнуляются. На втором шаге дуги, инцидентные вершинам критического пути, маркируются как «рассмотренные». Описанные выше шаги алгоритма KB/L повторятся до тех пор, пока все дуги не будут рассмотрены.

Ким в работе [20] использует стоимостную функцию

$$Cost\ function = w_1 * \sum \tau_i + (1 - w_1)\Big(w_2 * \sum c_{ij} + (1 - w_2) * \sum c_{ij}^{adj}\Big) \qquad (1)$$

для определения длины критического пути графа задания, где w_1 и w_2 — нормализующие множители, $\sum \tau_i$ — сумма вычислительных стоимостей всех вершин критического пути, c_{ij}^{adj} — коммуникационная стоимость дуг между вершиной критического пути и всеми его смежными вершинами, не входящими в критический путь. Ким не приводит систематического способа для определения нормализующих множителей. Положим, $w_1 = \frac{1}{2}$ и $w_2 = \frac{1}{2}$ для стоимостной функции, уменьшающей длину критического пути. Целевой функцией алгоритма KB/L является минимизация параллельного времени графа.

Рассмотрим алгоритм KB/L на примере графа задания на рис. 6а. Результат кластеризации с помощью алгоритма KB/L с параллельным временем 12,5 приведен на рис. 6б.

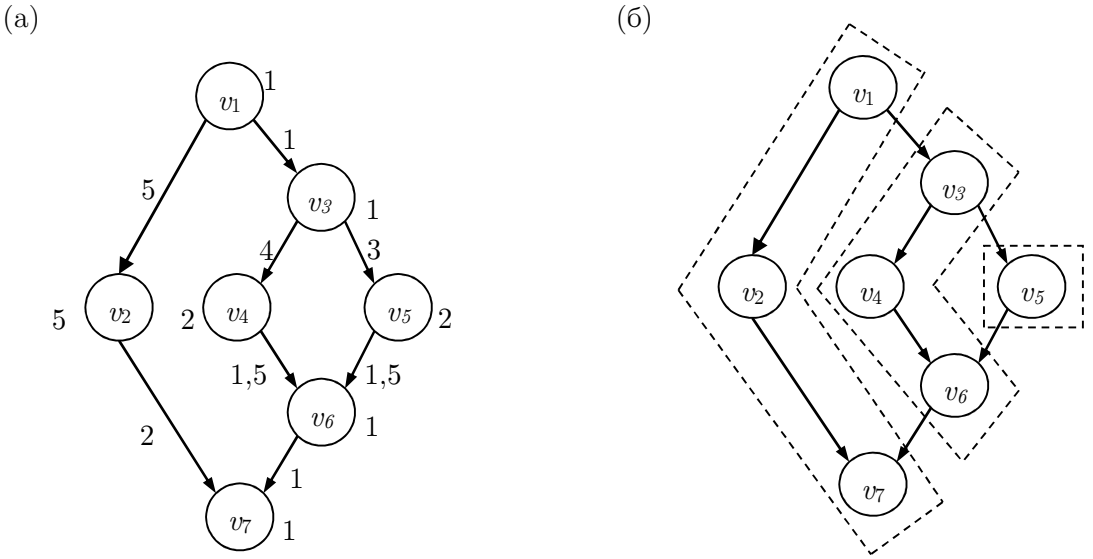


Рис. 6. (а) Граф с весами и разметкой;
(б) кластеризация графа с помощью алгоритма KB/L

На первом шаге критический путь состоит из вершин v_1, v_2 и v_7. Объединяем данные вершины в один кластер и исключаем их из дальнейшего рассмотрения. Оставшийся граф содержит вершины v_3, v_4, v_5, v_6. Новый критический путь состоит из вершин v_3, v_4, v_6. Данные вершины также группируются в один кластер. Окончательно, имеем три кластера $W_0 = \{v_1, v_2, v_7\}$, $W_1 = \{v_3, v_4, v_6\}$ и $W_2 = \{v_5\}$.

Алгоритм KB/L имеет сложность $O\big(v(v+e)\big)$, где v — число вершин, e — число дуг графа.

Алгоритм Саркара

В работе [86] рассматривается алгоритм кластеризации Саркара (Sarkar). Суть алгоритма может быть описана следующим образом. Все дуги графа задания сортируются в порядке убывания их коммуникационных стоимостей. Начиная с дуги с большей коммуникационной стоимостью, производится процесс их обнуления. Коммуникационная стоимость дуги обнуляется только в том случае, если параллельное время не увеличится на следующем шаге. Алгоритм Саркара завершается, когда рассмотрены все дуги графа задания. Целевой функцией алгоритма также является минимизация параллельного времени графа.

Рассмотрим алгоритм Саркара на примере графа задания на рис. 6а. Результат кластеризации с помощью алгоритма Саркара с параллельным временем равным 10 приведен на рис. 7.

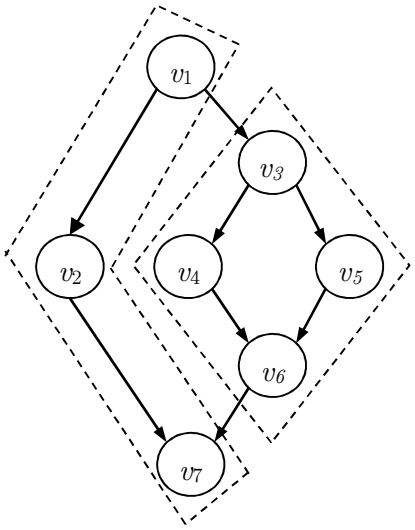


Рис. 7. Нелинейная кластеризация графа с помощью алгоритма Саркара

На первом шаге алгоритма Саркара выполним сортировку коммуникационных стоимостей дуг графа задания по убыванию. Получим следующий список дуг $\{(v_1, v_2), (v_3, v_4), (v_3, v_5), (v_2, v_7), (v_4, v_6), (v_5, v_6), (v_1, v_3), (v_6, v_7)\}$. В начальный момент времени каждая вершина графа задания находится в отдельном кластере. Параллельное время равно 14. На втором шаге обнуляем две первые дуги из списка (v_1, v_2) и (v_3, v_4). При этом параллельное время уменьшится до 12,5. На третьем шаге обнуляем коммуникационную стоимость дуги (v_3, v_5) и т.д. Окончательно, имеем два кластера $W_0 = \{v_1, v_2, v_7\}$ и $W_1 = \{v_3, v_4, v_5, v_6\}$ и параллельное время равное 10.

Алгоритм Саркара имеет сложность $O\big(e(v+e)\big)$, где v — число вершин графа, e — число дуг графа.

Алгоритм DSC

Алгоритм кластеризации доминирующей последовательности DSC (Dominant Sequence Clustering) [21] относится к эвристикам кластеризации. Главная идея алгоритма DSC состоит в выполнении последовательности шагов по обнулению коммуникационных стоимостей дуг графа задания с целью сокращения длины доминирующей последовательности, т.е. минимизации параллельного времени. Здесь *доминирующая последовательность* — это критический путь распланированного графа (рис. 8в). Заметим, что обычно под критическим путем графа понимают путь наибольшей длины, включающий вычислительные стоимости его вершин и ненулевые коммуникационные стоимости дуг (рис. 8а).

В начальный момент времени до выполнения алгоритма DSC каждая вершина содержится в отдельном кластере, все дуги графа задания помечаются как «нерассмотренные». После рассмотрения некоторой дуги на предмет возможности ее обнуления, дуга обозначается как «рассмотренная», а вершина, из которой исходит дуга, помечается как «распланированная». Распланированные вершины образуют множество S, не распланированные вершины составляют множество U. Вершина называется «свободной», если все ее предшествующие вершины распланированы.

На первом шаге алгоритма DSC из дуг, принадлежащих доминирующей последовательности графа задания, выбирается первая нерассмотренная дуга. Поиск дуги в доминирующей последовательности производится сверху вниз. На втором шаге дуга обнуляется, а ее вершины объединяются в один кластер, если параллельное время не увеличивается. Порядок выполнения задач в кластере определяется наибольшим значением $bot\_level(n_x, i)$, которое рассчитывается как сумма всех коммуникационных стоимостей дуг и вычислительных стоимостей вершин между вершиной n_x и нижней вершиной графа в доминирующей последовательности. Алгоритм DSC завершается, когда все дуги рассмотрены.

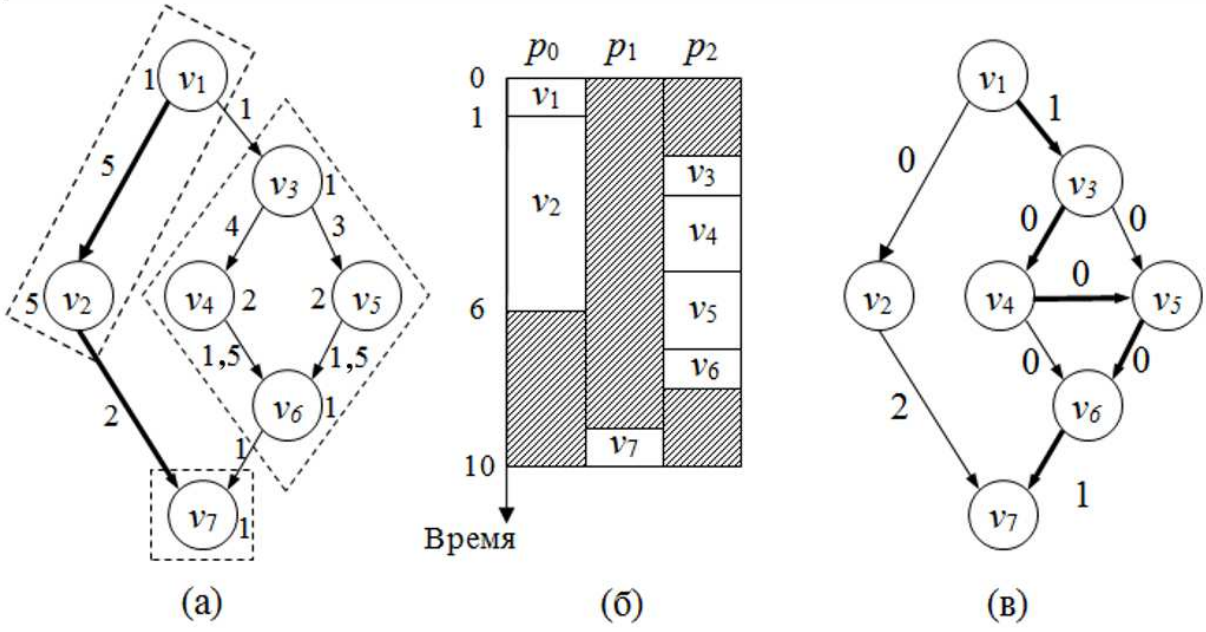


Рис. 8. (а) Кластеризованный граф и его критический путь; (б) диаграмма Ганта; (в) распланированный граф и его доминирующая последовательность

Рассмотрим алгоритм DSC на примере графа задания на рис. 6а. Результат кластеризации с помощью алгоритма DSC с параллельным временем равным 10.5 приведен на рис. 9.

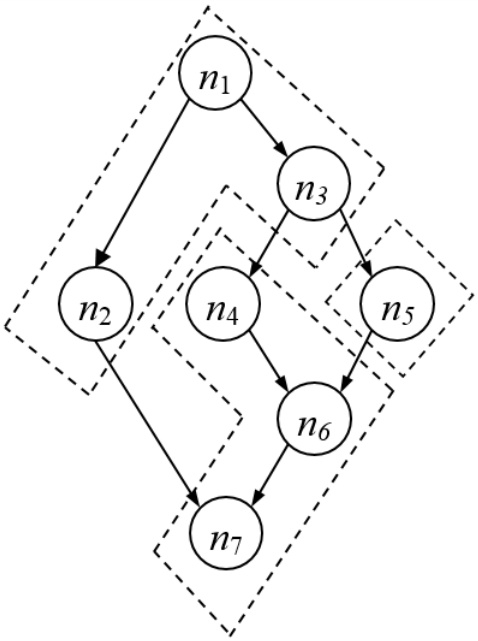


Рис. 9. Нелинейная кластеризация графа с помощью алгоритма DSC

На первом шаге алгоритма DSC определим доминирующую последовательность $DS_1 = \langle v_1, v_2, v_7 \rangle$ и параллельное время $PT_1 = 14$. Рассмотрим первую дугу из доминирующей последовательности — (v_1, v_2). Обнуление коммуникационной стоимости дуги уменьшит параллельное время до 13,5, поэтому произведем объединение вершин v_1, v_2 в один кластер.

Находим новую доминирующую последовательность $DS_2 = \langle v_1, v_3, v_4, v_6, v_7 \rangle$ и параллельное время $PT_1 = 13{,}5$. Рассмотрим первую дугу (v_1, v_3) из этой доминирующей последовательности. Ее обнуление уменьшит критический путь до 12,5. Следовательно, обнуляем данную дугу.

На третьем шаге алгоритма доминирующая последовательность не именилась $DS_3 = DS_2$. Рассмотрим дугу (v_3, v_4). Ее обнуление приведет к увеличению параллельного времени до 13,5. Следовательно, вершины v_3 и v_4 остаются в разных кластерах.

Аналогично, рассматриваем дуги (v_4, v_6), (v_3, v_5), (v_5, v_6) и (v_6, v_7) соответственно. Окончательно, имеем три кластера $M_0 = \{v_1, v_2, v_3\}$, $M_1 = \{v_4, v_6, v_7\}$ и $M_2 = \{v_5\}$ параллельное время равное 10,5.

Алгоритм DSC в общем случае имеет сложность $O\big((e + v)\log v\big)$, где v — число вершин графа, e — число дуг графа.

Алгоритм POS

Алгоритм планирования POS (Problem-Oriented Scheduling) [87, 88] для распределенных кластерных вычислительных сред в отличие от алгоритма DSC позволяет планировать запуск одной задачи на нескольких процессорных ядрах с учетом ограничений по масштабируемости данной задачи. Вычислительное задание, представляющее собой поток работ, моделируется в виде нагруженного, помеченного ориентированного графа задания. Каждая задача-вершина помечается парой натуральных чисел, первое из которых задает

время выполнения задачи на одном процессорном ядре, а второе — максимальное количество ядер, до которого задача демонстрирует ускорение близкое к линейному. Веса дуг задают объем данных, передаваемый между задачами. Предложенный алгоритм планирования ресурсов предусматривает построение последовательности конфигураций графа задания с целью минимизации критического пути в данном графе. В начальной конфигурации граф задания разбивается в каноническую ярусно-параллельную форму (ЯПФ). Задаются начальные кластеризация задач по вычислительным кластерам и расписание их выполнения. На следующем шаге происходит переход от начальной конфигурации графа к новой конфигурации с помощью перемещения одной из задач по ярусам ЯПФ и ее присоединения к определенному кластеру. Таким образом, изменяются кластеризация и расписание. Переход к новой конфигурации происходит только в том случае, когда сложность критического пути не увеличивается. Процесс построения последовательности конфигураций завершается, когда все вершины зафиксированы. Данный алгоритм реализован в брокере ресурсов [89–91] системы DiVTB [92–98] на платформе UNICORE на языке программирования Java.

Заключение

В статье были рассмотрены основные технологии распределенных вычислений: грид-вычисления и облачные вычисления. Дан обзор алгоритмов планирования ресурсов в распределенных вычислительных средах. На сегодняшний день предложено большое количество алгоритмов планирования, ориентированных на использование в распределенных вычислительных средах. Некоторые из этих алгоритмов производят планирование с учетом потоков работ в сложных приложениях. Однако часто алгоритмы не учитывают проблемно-ориентированную специфику, что может существенно повысить эффективность планирования. В связи с этим, перспективным является направления, связанное с разработкой алгоритмов планирования ресурсами в проблемно-ориентированных средах, которые позволяют создать эффективную и действенную систему планирования ресурсов.

Работа выполнялась при поддержке Российского фонда фундаментальных исследований (проект № 14-07-31159мол_а «Мой первый грант»).

Литература

1. Buyya, R. Economic Models for Resource Management and Scheduling in Grid Computing / R. Buyya, D. Abramson et al. // Journal of Concurrency and Computation: Practice and Experience. — 2002. — Vol. 14, — Issue 13–15. — P. 1507–1542.
2. Foster, I. A Quality of Service Architecture That Combines Resource Reservation and Application Adaptation / I. Foster, A. Roy, V. Sander // Proceedings 8th Int. Workshop on Quality of Service (Pittsburgh, PA, USA, June 5–7, 2000). — Carnegie Mellon University, 2000. — P. 181–188.
3. Foster, I. The Anatomy of the Grid: Enabling Scalable Virtual Organizations / I. Foster, C. Kesselman, S. Tuecke // International Journal of Supercomputer Applications and High Performance Computing. — 2001. — Vol. 15, — No 3. — P. 200–222.
4. Foster, I. The Grid. Blueprint for a new computing infrastructure / I. Foster, C. Kesselman — San Francisco: Morgan Kaufman, 1999. — 677 p.

5. Jennings, R. Cloud Computing with the Windows Azure Platform / R. Jennings, — Indianapolis, Indiana: Wiley Publishing, Inc., 2009. — 360 p.
6. Marshall, P. T. Improving Utilization of Infrastructure Clouds / P. Marshall, K. Keahey, T. Freeman // Cluster, Cloud and Grid Computing (CCGrid 2011): Proceedings of the IEEE/ACM International Symposium (Newport Beach, CA, USA. May 23–26, 2011). — IEEE Computer Society, 2011. — P. 205–214.
7. NIST Special Publication 800-145. A NIST Definition of Cloud Computing. URL: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf (дата обращения: 01.02.2014).
8. Sanderson, D. Programming Google App Engine: Build and Run Scalable Web Apps on Google's Infrastructure / D. Sanderson, — O'Reilly Media, 2009. — 400 p.
9. Dong, F. Scheduling algorithms for grid computing: State of the art and open problems. Technical Report No. 2006-504 / F .Dong, S.G. Akl, — Queen's University, Canada, 2006. — P. 55.
10. Berman, F. Application-Level Scheduling on Distributed Heterogeneous Networks / F. Berman, R. Wolski et al. // Supercomputing: Proceedings of the ACM/IEEE conference (Pittsburgh, Pennsylvania USA, May 25–28, 1996). — IEEE Computer Society, 1996. — P. 39–39.
11. Iverson, M. Dynamic, Competitive Scheduling of Multiple DAGs in a Distributed Heterogeneous Environment / M. Iverson, F. Ozguner // Proceedings of Seventh Heterogeneous Computing Workshop (Orlando, Florida USA, March 30, 1998). — IEEE Computer Society, 1998. — P. 70–78.
12. Maheswaran, M. Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems / M. Maheswaran, S. Ali et al. // Journal of Parallel and Distributed Computing. — 1999. — Vol. 59, — No. 2. — P. 107–131.
13. Zhu, Y. A Survey on Grid Scheduling Systems. — Technical Report SJTU_CS_TR_200309001. — Department of Computer Science and Engineering, Shanghai Jiao. — 2013. / Y. Zhu, L.M. Ni. URL: http://www.cs.sjtu.edu.cn/~yzhu/reports/SJTU_CS_TR_200309001.pdf (дата обращения: 01.08.2014).
14. Belhajjame, K. A flexible workflow model for process-oriented applications / K. Belhajjame, G. Vargas-Solar, C. Collet // Web Information Systems Engineering (WISE'01): Proceedings of the Second International Conference (Kyoto, Japan, December 3–6, 2001). — IEEE Computer Society. — Vol. 1, — No. 1. — P. 72–80.
15. Condor. URL: http://www.cs.wisc.edu/condor (дата обращения: 04.06.2014).
16. Laszewski, G. CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids / G. Laszewski, I. Foster et al. // Proceedings of the ACM Java Grande 2000 Conference (San Francisco, CA, USA, June 3–5, 2000). — ACM Press. — P. 97–106.
17. Deelman, E. Pegasus: Mapping Scientific Workflows onto the Grid / E. Deelman, J. Blythe et al. // AcrossGrids Conference (AxGrids 2004): Proceedings of the Second European Conference (Nicosia, Cyprus, January 28–30, 2004). — Springer. — P. 11–26.
18. Cao, J. GridFlow: Workflow Management for Grid Computing / J. Cao, S.A. Jarvis et al. // Cluster Computing and the Grid (CCGrid'03): Proceedings of the 3rd International Symposium (Tokyo, Japan, May 12–15, 2003). — IEEE Computer Society, 2003. — P. 198–205.

19. Wieczorek, M. Scheduling of Scientific Workflows in the ASKALON Grid Environment / M. Wieczorek, R. Prodan, T. Fahringer // ACM SIGMOD Record. — 2005. — Vol. 34, — No. 3. — P. 56–62.
20. Kim, S.J. A general approach to multiprocessor scheduling. — Report TR-88-04. — Department of Computer Science, University of Texas at Austin. — 1988. / S.J. Kim. URL: ftp://ftp.cs.utexas.edu/pub/techreports/tr88-04.pdf (дата обращения: 01.08.2014).
21. Yang, T. DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors / T. Yang, A. Gerasoulis // IEEE Transactions on Parallel and Distributed Systems. — 1994. — Vol. 5, — No. 9. — P. 951–967.
22. Baker, M. Grids and Grid Technologies for Wide-area Distributed Computing / M. Baker, R. Buyya, D. Laforenza // Journal of Software-Practice & Experience. — 2002. — Vol. 32, — No. 15. — P. 1437–1466.
23. Schopf, J. Ten Actions When SuperScheduling, document of Scheduling Working Group, Global Grid Forum. / J. Schopf. URL: http://www.ggf.org/documents/GFD.4.pdf (дата обращения: 12.06.2014).
24. Berman, F. Adaptive Computing on the Grid Using AppLeS / F. Berman, R. Wolski et al. // IEEE Transactions on Parallel and Distributed Systems. — 2003. — Vol. 14, — No. 4. — P. 369–382.
25. Hamscher, V. Evaluation of Job-Scheduling Strategies for Grid Computing / V. Hamscher, U. Schwiegelshohn et al. // GRID 2000: Proceedings of the First IEEE/ACM International Workshop (Bangalore, India, December 17, 2000). — IEEE Computer Society, 2000. — P. 191–202.
26. Mateescu, G. Quality of Service on the Grid via Metascheduling with Resource Co-Scheduling and Co-Reservation / G. Mateescu // International Journal of High Performance Computing Applications. — 2003. — Vol. 17, — No. 3. — P. 209–218.
27. The Globus Toolkit. URL: http://www.globus.org (дата обращения: 04.01.2013).
28. Czajkowski, K. Grid Information Services for Distributed Resource Sharing / K. Czajkowski, S. Fitzgerald et al. // High-Performance Distributed Computing (HPDC-10): Proceedings the 10th IEEE International Symposium (San Francisco, California, USA, August 7–9, 2001). — IEEE Computer Society, 2001. — P. 181–194.
29. Khokhar, A.A. Heterogeneous Computing: Challenges and Opportunities / A.A. Khokhar, V.K. Prasanna et al. // IEEE Computer. — 1993. — Vol. 26, — No. 6. — P. 18–27.
30. Siegel, H.J. Software Support for Heterogeneous Computing / H.J. Siegel, H.G. Dietz, J.K. Antonio // ACM Computing Surveys. — 1996. — Vol. 28, — No. 1. — P. 237–239.
31. Cooper, K. New Grid Scheduling and Rescheduling Methods in the GrADS Project / K. Cooper, A. Dasgupta et al. // International Parallel and Distributed Processing Symposium (IPDPS'04): Proceedings of the 18th International Symposium (Santa Fe, New Mexico USA, April 26–30, 2004). — IEEE Computer Society, 2004. — P. 199–206.
32. Czajkowski, K. A Resource Management Architecture for Metacomputing Systems / K. Czajkowski, I. Foster et al. // Job Scheduling Strategies for Parallel Processing (JSSPP 1998): Proceedings of the 4th Workshop (Orlando, Florida USA, March 30, 1998). — Springer, Lecture Notes in Computer Science, 1998. — Vol. 1459. — P. 62–82.
33. OpenPBS. URL: http://www.openpbs.org (дата обращения: 14.12.2013).
34. Condor. URL: http://www.cs.wisc.edu/condor (дата обращения: 04.12.2013).

35. Wolski, R. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing / R. Wolski, N.T. Spring, J. Hayes // Future Generation Computing Systems. — 1999. — Vol. 15, — No. 5–6. — P. 757–768.
36. Sacerdoti, F.D. Wide area cluster monitoring with Ganglia / F.D. Sacerdoti, M.J. Katz et al. // Cluster Computing (CLUSTER 2003): Proceedings of IEEE International Conference (Hong Kong, December 1–4, 2003). — IEEE Computer Society, 2003. — P. 289-298.
37. NIST Special Publication 800-145. A NIST Definition of Cloud Computing. URL: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf (дата обращения: 01.08.2014).
38. Streit, A. UNICORE: Getting to the heart of Grid technologies / A. Streit // eStrategies. — 2009. — Vol. 3. — P. 8–9.
39. Streit, A. UNICORE - What lies beneath Grid functionality? / A. Streit // eStrategies. — 2008. — Vol. 7. — P. 38–39.
40. Casavant, T. A Taxonomy of Scheduling in General-purpose Distributed Computing Systems / T. Casavant, J. Kuhl // IEEE Transactions on Software Engineering. — 1988. — Vol. 14, — No. 2. — P. 141–154.
41. Braun, R. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems / R. Braun, H. Siegel et al. // Parallel and Distributed Computing. — 2001. — Vol. 61, — No. 6, — P. 810–837.
42. Casanova, H. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments / H. Casanova, A. Legrand et al. // Heterogeneous Computing Workshop (HCW'00): Proceedings of the 9th Workshop (Cancun, Mexico, May 1, 2000). — IEEE Computer Society, 2000. — P. 349–363.
43. You, S.Y. Task Scheduling Algorithm in GRID Considering Heterogeneous Environment / S.Y. You, H.Y. Kim et al. // Parallel and Distributed Processing Techniques and Applications (PDPTA '04): Proceedings of the International Conference (Nevada, USA, June 21–24, 2004). — CSREA Press, 2004. — Vol. 1. — P. 240–245.
44. Kurowski, K. Improving Grid Level Throughput Using Job Migration And Rescheduling / K. Kurowski, B. Ludwiczak et al. // Scientific Programming. — 2004. — Vol. 12, — No. 4. — P. 263–273.
45. Takefusa, A. A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid / A. Takefusa, S. Matsuoka et al. // High Performance Distributed Computing (HPDC-10): Proceedings of the 10th IEEE International Symposium (San Francisco, California, USA, August 7–9, 2001). — IEEE Computer Society, 2001. — P. 406–415.
46. Chen, H. Distributed Dynamic Scheduling of Composite Tasks on Grid Computing Systems / H. Chen, M. Maheswaran // International Parallel and Distributed Processing Symposium (IPDPS 2002): Proceedings of the 16th International Symposium (Fort Lauderdale, FL, USA, April 15-19, 2002). — IEEE Computer Society, 2002. — P. 88–97.
47. Muthuvelu, N. A Dynamic Job Grouping-Based Scheduling for Deploying Applications with Fine-Grained Tasks on Global Grids / N. Muthuvelu, J. Liu et al. // Grid Computing and e-Research (AusGrid 2005): Proceedings of the 3rd Australasian Workshop (Newcastle, NSW, Australia, January 30 – February 4, 2005). — Australian Computer Society, 2005. — P. 41–48.

48. Wright, D. Cheap Cycles from the Desktop to the Dedicated Cluster: Combining Opportunistic and Dedicated Scheduling with Condor / D. Wright URL: http://www.linuxclustersinstitute.org/conferences/archive/2001/PDF/wright_wisc.pdf (дата обращения: 04.08.2014).
49. Chapin, S.J. The Legion Resource Management System / S.J. Chapin, D. Katramatos et al. // Job Scheduling Strategies for Parallel Processing (JSSPP '99): Proceedings of the 5th Workshop (San Juan, Puerto Rico, April 16, 1999). — Springer, Lecture Notes in Computer Science, 1999. — Vol. 1659. — P. 162–178.
50. Rotithor, H.G. Taxonomy of Dynamic Task Scheduling Schemes in Distributed Computing Systems / H.G. Rotithor // Computer and Digital Techniques. — 1994. — Vol. 141, — No. 1. — P. 1–10.
51. James, H.A. Scheduling in Metacomputing Systems. — Ph.D. Thesis. — The Department Of Computer Science, University of Adelaide, Australia. — 1999. / H.A. James URL: http://digital.library.adelaide.edu.au/dspace/bitstream/2440/19450/1/09phj274.pdf (дата обращения: 04.08.2014).
52. Aggarwal, A.K. An Adaptive Generalized Scheduler for Grid Applications / A.K. Aggarwal, R.D. Kent // High Performance Computing Systems and Applications (HPCS'05): Proceedings of the 19th Annual International Symposium (Guelph, Ontario Canada, May 15–18, 2005). — IEEE Computer Society, 2005. — P. 15–18.
53. Sabin, G. Scheduling of Parallel Jobs in a Heterogeneous Multi-Site Environment / G. Sabin, R. Kettimuthu et al. // Job Scheduling Strategies for Parallel Processing (JSSPP'03): Proceedings of the 9th International Workshop (Seattle, WA, USA, June 24, 2003). — Springer, Lecture Notes in Computer Science, 2003. — Vol. 2862. — P. 87–104.
54. Arora, M. A Decentralized Scheduling and Load Balancing Algorithm for Heterogeneous Grid Environments / M. Arora, S.K. Das, R. Biswas // International Conference on Parallel Processing Workshops (ICPPW'02): Proceedings of International Conference (Vancouver, British Columbia Canada, August 20-23, 2002). — IEEE Computer Society, 2002. — P. 499–505.
55. El-Rewini, H. Task Scheduling in Parallel and Distributed Systems / H. El-Rewini, T. Lewis, H. Ali — Prentice Hall, 1994. — 290 p.
56. Buyya, R. The Grid Economy / R. Buyya, D. Abramson, S. Venugopal // Proceedings of the IEEE. — 2005. — Vol. 93, — No. 3. — P. 698–714.
57. Buyya, R. An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications / R. Buyya, J. Giddy, D. Abramson // Active Middleware Services (AMS 2000): Proceedings of the 2nd International Workshop (Pittsburgh, USA, August 1, 2000). — Kluwer Academic Press, 2000. — Vol. 583. — P. 221–230.
58. Yu, J. QoS-based Scheduling of Workflow Applications on Service Grids / J. Yu, R. Buyya, C.K. Tham // e-Science and Grid Computing (e-Science'05): Proceedings of the 1st IEEE International Conference (Melbourne, Australia, December 5–8, 2005). — IEEE Computer Society, 2005. — P. 1–9.
59. Venugopal, S. A Deadline and Budget Constrained Scheduling Algorithm for eScience Applications on Data Grids / S. Venugopal, R. Buyya // International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-2005): Proceedings of 6th

International Conference (Melbourne, Australia, October 2-3, 2005). — Springer, Lecture Notes in Computer Science, 2005. — P. 60–72.

60. Ernemann, C. Economic Scheduling in Grid Computing / C. Ernemann, V. Hamscher, R. Yahyapour // Job Scheduling Strategies for Parallel Processing (JSSPP'02): Proceedings of 8th Workshop (Edinburgh, Scotland, UK, July 24, 2002). — Springer, Lecture Notes in Computer Science, 2002. — P. 128–152.
61. Zhu, Y. Incentive-based P2P Scheduling in Grid Computing / Y. Zhu, L. Xiao et al. // Grid and Cooperative Computing (GCC'04): Proceedings of the 3rd International Conference (Wuhan, China, October 21-24, 2004). — Springer, Lecture Notes in Computer Science, 2004. — P. 209–216.
62. Young, L. Scheduling Architecture and Algorithms within the ICENI Grid Middleware / L. Young, S. McGough et al. // Proceedings of UK e-Science All Hands Meeting (Nottingham, UK, September 2003). — IOP Publishing Ltd., 2003. — P. 5–12.
63. Kim, S. A Genetic Algorithm Based Approach for Scheduling Decomposable Data Grid Applications / S. Kim, J.B. Weissman // International Conference on Parallel Processing (ICPP'04): Proceedings of the International Conference (Montreal, Quebec Canada, August 15–18, 2004). — IEEE Computer Society, 2004. — P. 406–413.
64. Song, S. Security-Driven Heuristics and A Fast Genetic Algorithm for Trusted Grid Job Scheduling / S. Song, Y. Kwok, K. Hwang // International Parallel and Distributed Processing Symposium (IPDPS'05): Proceedings of 19th IEEE International Symposium (Denver, Colorado USA, April 25–29, 2005). — IEEE Computer Society, 2005. — P. 65–74.
65. Spooner, D.P. Localised Workload Management using Performance Prediction and QoS Contracts / D.P. Spooner, J. Cao et al. // UK Performance Engineering Workshop (UKPEW 2002): Proceedings of the 18th Annual Workshop (Glasgow, UK, July 10–11, 2002). — University of Glasgow, 2002. — P. 69–80.
66. Liu, Y. Survey on Grid Scheduling. — Department of Computer Science, University of Iowa. — 2004.
67. Abraham, A. Nature's Heuristics for Scheduling Jobs on Computational Grids / A. Abraham, R. Buyya and B. Nath // Advanced Computing and Communications (ADCOM 2000): Proceedings of 8th IEEE International Conference (Cochin, India, December 14–16, 2000). — IEEE Computer Society, 2000. — P. 45–52.
68. Shan, H. Scheduling in Heterogeneous Grid Environments: The Effects of Data Migration / H. Shan, L. Oliker et al. // Advanced Computing and Communication (ADCOM 2004): Proceedings of the 12th IEEE International Conference (Ahmedabad Gujarat, India, December 15–18, 2004). — IEEE Computer Society, 2004. — P. 1–8.
69. He, X. A QoS Guided Min-Min Heuristic for Grid Task Scheduling / X. He, X. Sun, G. Laszewski // Computer Science and Technology: Special Issue on Grid Computing. — 2003. — Vol. 18, — No. 4. — P. 442–451.
70. Wu, M. Segmented Min-Min: A Static Mapping Algorithm for Meta-Tasks on Heterogeneous Computing Systems / M. Wu, W. Shu, H. Zhang // Heterogeneous Computing Workshop (HCW'00): Proceedings of the 9th Workshop (Cancun, Mexico, May 1, 2000). — IEEE Computer Society, 2000. — P. 375–385.
71. Subramani, V. Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests / V. Subramani, R. Kettimuthu et al. // High Performance Distributed

Computing (HPDC 2002): Proceedings of 11th IEEE Symposium (Edinburgh, Scotland, July 23–26, 2002). — IEEE Computer Society, 2002. — P. 359–366.

72. Silva, D.P. Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids / D.P. Silva, W. Cirne, F.V. Brasileiro // Euro-Par 2003: Proceedings of the 9th International Conference (Klagenfurt, Austria, August 26–29, 2003). — Springer, Lecture Notes in Computer Science, 2003. — P. 169–180.
73. Radulescu, A. On the Complexity of List Scheduling Algorithms for Distributed Memory Systems / A. Radulescu, A.J.C. Gemund // Supercomputing (SC'99): Proceedings of 13th International Conference (Portland, Oregon, USA, November 13–19, 1999). — IEEE Computer Society, 1999. — P. 68–75.
74. Sakellariou, R. A Low-cost Rescheduling Policy for Efficient Mapping of Workflows on Grid Systems / R. Sakellariou, H. Zhao // Scientific Programming. — 2004. — Vol. 12, — No. 4. — P. 253–262.
75. Topcuoglu, H. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing / H. Topcuoglu, S. Hariri, M.Y. Wu // IEEE Transactions on Parallel and Distributed Systems. — 2002. — Vol. 13, — No. 3. — P. 260–274.
76. Zhao, H. An Experimental Investigation into the Rank Function of the Heterogeneous Earliest Finish Time Scheduling Algorithm / H. Zhao, R. Sakellariou // Euro-Par 2003: Proceedings of the 9th International Conference (Klagenfurt, Austria, August 26–29, 2003). — Springer, Lecture Notes in Computer Science, 2003. — P. 189–194.
77. Sakellariou, R. A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems / R. Sakellariou, H. Zhao // International Parallel and Distributed Processing Symposium (IPDPS'04): Proceedings of the 18th International Symposium (Santa Fe, New Mexico USA, April 26–30, 2004). — IEEE Computer Society, 2004. — P. 111–123.
78. Ma, T. Critical-Path and Priority based Algorithms for Scheduling Workflows with Parameter Sweep Tasks on Global Grids / T. Ma, R. Buyya // Computer Architecture and High Performance Computing (SBAC-PAD 2005): Proceedings of the 17th International Symposium (Rio de Janeiro, Brazil, October 24–27, 2005). — IEEE Computer Society, 2005. — P. 251–258.
79. Darbha, S. Optimal Scheduling Algorithm for Distributed Memory Machines / S. Darbha, D.P. Agrawal // IEEE Transactions on Parallel and Distributed Systems. — 1998. — Vol. 9, — No. 1. — P. 87–95.
80. Ranaweera, S. A Task Duplication Based Scheduling Algorithm for Heterogeneous Systems / S. Ranaweera, D.P. Agrawal // International Parallel and Distributed Processing Symposium (IPDPS'00): Proceedings of 14th International Symposium (Cancun, Mexico, May 1–5, 2000). — IEEE Computer Society, 2005. — P. 445–450.
81. Bajaj, R. Improving Scheduling of Tasks in A Heterogeneous Environment / R. Bajaj, D.P. Agrawal // IEEE Transactions on Parallel and Distributed Systems. — 2004. — Vol. 15, — No. 2. — P. 107–118.
82. Gerasoulis, A. A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors / A. Gerasoulis, T. Yang // Parallel and Distributed Computing. — 1992. — Vol. 16, — No. 4. — P. 276–291.
83. Liou, J. An Efficient Task Clustering Heuristic for Scheduling DAGs on Multiprocessors / J. Liou, M.A. Palis // Parallel and Distributed Processing (SPDP 1996): Proceedings of

the Eighth IEEE Symposium (New Orleans, Louisiana, USA, October 23–26, 1996). — IEEE Computer Society, 1996. — P. 152–156.

84. Liou, J. A Comparison of General Approaches to Multiprocessor Scheduling / J. Liou, M.A. Palis // International Parallel Processing Symposium (IPPS '97): Proceedings the 11th International Symposium (Geneva, Switzerland, April 1–5, 1997). — IEEE Computer Society, 1996. — P. 152–156.
85. Kim, S.J. A general approach to mapping of parallel computation upon multiprocessor architectures / S.J. Kim, J.C. Browne // International Conference on Parallel Processing, (ICPP'88): Proceedings of the International Conference (The Pennsylvania State University, University Park, PA, USA, August 1988). — Pennsylvania State University Press., 1988. — Vol. 3. — P. 1-8.
86. Sarkar, V. Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors / V. Sarkar — Cambridge, MA: The MIT Press, 1989. — P. 215.
87. Шамакина, А.В. Формальная модель задания в распределенных вычислительных средах / А.В. Шамакина, Л.Б. Соколинский // Параллельные вычислительные технологии (ПаВТ'2014): Труды международной научной конференции (Ростов-на-Дону, 1–3 апреля 2014 г.). — Челябинск: Издательский центр ЮУрГУ, 2014. — С. 343–354.
88. Радченко, Г.И. Модели и методы профилирования и оценки времени выполнения потоков работ в суперкомпьютерных системах / Г.И. Радченко, Л.Б. Соколинский, А.В. Шамакина // Вычислительные методы и программирование: Новые вычислительные технологии. — 2013. — Т. 14. — Вып 4. — С. 96–103.
89. Шамакина. А.В. Брокер ресурсов для поддержки проблемно-ориентированных сред / А.В. Шамакина // Вестник ЮУрГУ. Серия "Вычислительная математика и информатика". — 2012. — № 46(305). — Вып. 1. — С. 88–98.
90. Шамакина. А.В. Организация брокера ресурсов в системе CAEBeans / А.В. Шамакина // Вестник Южно-Уральского государственного университета. Серия "Математическое моделирование и программирование". — 2008. — № 27(127). — Вып. 2. — С. 110–116.
91. Шамакина. А.В. CAEBeans Broker: брокер ресурсов системы CAEBeans / А.В. Шамакина // Вестник ЮУрГУ. Серия "Математическое моделирование и программирование". — 2010. — № 16(192). — Вып. 5. — С. 107–115.
92. Савченко, Д.И. DiVTB Server: среда выполнения виртуальных экспериментов / Д.И. Савченко, Г.И. Радченко // Параллельные вычислительные технологии (ПаВТ'2013): Труды международной научной конференции (Челябинск, 1–5 апреля 2013 г.). — Челябинск: Издательский центр ЮУрГУ, 2013. — С. 532–539.
93. Радченко, Г.И. Распределенные виртуальные испытательные стенды: использование систем инженерного проектирования и анализа в распределенных вычислительных средах / Г.И. Радченко // Вестник Южно-Уральского государственного университета. Серия: Математическое моделирование и программирование. — 2011. — № 37. — С. 108–121.
94. Радченко, Г.И. Технология построения проблемно-ориентированных иерархических оболочек над инженерными пакетами в грид-средах / Г.И. Радченко // Системы управления и информационные технологии. — 2008. — № 4(34). — С. 57–61.
95. Радченко, Г.И. Технология построения виртуальных испытательных стендов в распределенных вычислительных средах / Г.И. Радченко, Л.Б. Соколинский // Научно-

технический вестник информационных технологий, механики и оптики. — 2008. — № 54. — С. 134–139.
96. Радченко, Г.И. Методы организации грид-оболочек системного слоя в технологии CAEBeans / Г.И. Радченко // Вестник Южно-Уральского государственного университета. Серия: Математическое моделирование и программирование. — 2008. — № 15. — С. 69–80.
97. Радченко, Г.И. Грид-система CAEBeans: интеграция ресурсов инженерных пакетов в распределенные вычислительные среды / Г.И. Радченко // Вестник Нижегородского университета им. Н.И. Лобачевского. — 2009. — № 6-1. — С. 192–202.
98. Savchenko, D. Mjolnirr: private PaaS as distributed computing evolution / D. Savchenko, G. Radchenko // MIPRO 2014: Proceedings of the 37th International Convention (Opatija, Croatia, May 26–30, 2014). — IEEE Computer Society, 2014. — P. 386–391.

Шамакина Анастасия Валерьевна, старший преподаватель кафедры системного программирования, Южно-Уральский государственный университет (Челябинск, Российская Федерация), shamakinaav@susu.ru.

Поступила в редакцию 5 августа 2014 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2014, vol. 3, no. 3, pp. 51–85

SURVEY ON DISTRIBUTED COMPUTING TECHNOLOGIES

A.V. Shamakina, South Ural State University (Chelyabinsk, Russian Federation)

This paper presents an overview of distributed computing technologies from the perspective of the scheduling organization in grid and cloud environments. The architecture of the UNICORE platform, focused on ensuring a transparent secure access to resources of distributed computing environment, is considered. A general classification of scheduling algorithms in distributed computing environments and a classification of scheduling algorithms for independent tasks and jobs with dependencies between tasks are shown.

Keywords: distributed computing, problem-oriented distributed computing environments, Grid, UNICORE, scheduling algorithms, clustering algorithms, a directed acyclic graph, workflow.

References

1. Buyya R., Abramson D. et al. Economic Models for Resource Management and Scheduling in Grid Computing // Journal of Concurrency and Computation: Practice and Experience. 2002. Vol. 14, Issue 13–15. P. 1507–1542.
2. Foster I., Roy A., Sander V. A Quality of Service Architecture That Combines Resource Reservation and Application Adaptation // Proceedings 8th Int. Workshop on Quality

of Service, Pittsburgh, PA, USA, June 5–7, 2000. Carnegie Mellon University P. 181–188.

3. Foster I., Kesselman C., Tuecke S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations // International Journal of Supercomputer Applications and High Performance Computing. 2001. Vol. 15, No 3. P. 200–222.
4. Foster I., Kesselman C. The Grid. Blueprint for a new computing infrastructure. San Francisco: Morgan Kaufman, 1999. 677 p.
5. Jennings R. Cloud Computing with the Windows Azure Platform. Indianapolis, Indiana: Wiley Publishing, Inc., 2009. 360 p.
6. Marshall, P., Keahey K., Freeman, T. Improving Utilization of Infra–structure Clouds // Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, Newport Beach, CA, USA. May 23–26, 2011. IEEE Computer Society, 2011. P. 205–214.
7. NIST Special Publication 800–145. A NIST Definition of Cloud Computing. URL: http://csrc.nist.gov/publications/nistpubs/800–145/SP800–145.pdf (дата обращения: 01.02.2014).
8. Sanderson D. Programming Google App Engine: Build and Run Scalable Web Apps on Google`s Infrastructure. O'Reilly Media, 2009. 400 p.
9. Dong F, Akl S.G. Scheduling algorithms for grid computing: State of the art and open problems. Technical Report No. 2006–504, Queen's University, Canada, 2006. P. 55.
10. Berman F., Wolski R. et al. Application–Level Scheduling on Distributed Heterogeneous Networks. // Proceedings of the ACM/IEEE conference on Supercomputing, Pittsburgh, Pennsylvania USA, May 25–28, 1996. IEEE Computer Society, 1996. P. 39–39.
11. Iverson M., Ozguner F. Dynamic, Competitive Scheduling of Multiple DAGs in a Distributed Heterogeneous Environment // Proceedings of Seventh Heterogeneous Computing Workshop, Orlando, Florida USA, March 30, 1998. IEEE Computer Society, 1998. P. 70–78.
12. Maheswaran M., Ali S. et al. Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems // Journal of Parallel and Distributed Computing. 1999. Vol. 59, No. 2. P. 107–131.
13. Zhu Y. A Survey on Grid Scheduling Systems. Technical Report SJTU_CS_TR_200309001. Department of Computer Science and Engineering, Shanghai Jiao. 2013. URL: http://www.cs.sjtu.edu.cn/~yzhu/reports/SJTU_CS_TR_200309001.pdf (дата обращения: 01.08.2014).
14. Belhajjame K., Vargas–Solar G., Collet C. A flexible workflow model for process–oriented applications // Proceedings of the Second International Conference on Web Information Systems Engineering (WISE'01), Kyoto, Japan, December 3–6, 2001. IEEE Computer Society. Vol. 1, No. 1. P. 72–80.
15. Condor. URL: http://www.cs.wisc.edu/condor (дата обращения: 04.06.2014).
16. Laszewski G., Foster I. et al. CoG Kits: A Bridge between Commodity Distributed Computing and High–Performance Grids // Proceedings of the ACM Java Grande 2000 Conference, CA, USA, June 3–5, 2000. ACM Press. P. 97–106.
17. Deelman E., Blythe J. et al. Pegasus: Mapping Scientific Workflows onto the Grid // Proceedings of Grid Computing: Second European AcrossGrids Conference (AxGrids 2004), Nicosia, Cyprus, January 28–30, 2004. Springer. P. 11–26.

18. Cao J., Jarvis S. A. et al. GridFlow: Workflow Management for Grid Computing // Proceedings of the 3rd International Symposium on Cluster Computing and the Grid (CCGrid'03), Tokyo, Japan, May 12–15, 2003. IEEE Computer Society, 2003. P. 198–205.
19. Wieczorek M., Prodan R., Fahringer T. Scheduling of Scientific Workflows in the ASKALON Grid Environment // ACM SIGMOD Record. 2005. Vol. 34, No. 3. P. 56–62.
20. Kim S.J. A general approach to multiprocessor scheduling. Report TR-88-04. Department of Computer Science, University of Texas at Austin. 1988. URL: ftp://ftp.cs.utexas.edu/pub/techreports/tr88-04.pdf (дата обращения: 01.08.2014).
21. Yang T., Gerasoulis A. DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors // IEEE Transactions on Parallel and Distributed Systems. 1994. Vol. 5, No. 9. P. 951–967.
22. Baker M., Buyya R., Laforenza D. Grids and Grid Technologies for Wide–area Distributed Computing // Journal of Software-Practice & Experience. 2002. Vol. 32, No. 15. P. 1437–1466.
23. Schopf J. Ten Actions When SuperScheduling, document of Scheduling Working Group, Global Grid Forum. URL: http://www.ggf.org/documents/GFD.4.pdf (дата обращения: 12.06.2014).
24. Berman F., Wolski R. et al. Adaptive Computing on the Grid Using AppLeS // IEEE Transactions on Parallel and Distributed Systems. 2003. Vol. 14, No. 4. P. 369–382.
25. Hamscher V., Schwiegelshohn U. et al. Evaluation of Job–Scheduling Strategies for Grid Computing // Proceedings of GRID 2000 GRID 2000, First IEEE/ACM International Workshop, Bangalore, India, December 17, 2000. IEEE Computer Society. P. 191–202.
26. Mateescu G. Quality of Service on the Grid via Metascheduling with Resource Co–Scheduling and Co–Reservation // International Journal of High Performance Computing Applications. 2003. Vol. 17, No. 3. P. 209–218.
27. The Globus Toolkit. URL: http://www.globus.org (дата обращения: 04.01.2013).
28. Czajkowski K., Fitzgerald S. et al. Grid Information Services for Distributed Resource Sharing // Proceedings the 10th IEEE International Symposium on High– Performance Distributed Computing (HPDC–10), San Francisco, California, USA, August 7–9, 2001. IEEE Computer Society, 2001. P. 181–194.
29. Khokhar A.A., Prasanna V.K. et al. Heterogeneous Computing: Challenges and Opportunities // IEEE Computer. 1993. Vol. 26, No. 6. P. 18–27.
30. Siegel H. J., Dietz H. G., Antonio J. K., Software Support for Heterogeneous Computing // ACM Computing Surveys. 1996. Vol. 28, No. 1. P. 237–239.
31. Cooper K., Dasgupta A. et al. New Grid Scheduling and Rescheduling Methods in the GrADS Project // Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04), Santa Fe, New Mexico USA, April 26–30, 2004. IEEE Computer Society, 2004. P. 199–206.
32. Czajkowski K., Foster I. et al. A Resource Management Architecture for Metacomputing Systems // Proceedings of the 4th Workshop on Job Scheduling Strategies for Parallel Processing, Orlando, Florida USA, March 30, 1998. Springer, Lecture Notes in Computer Science, 1998. Vol. 1459. P. 62–82.

33. OpenPBS. URL: http://www.openpbs.org (дата обращения: 14.12.2013).
34. Condor. URL: http://www.cs.wisc.edu/condor (дата обращения: 04.12.2013).
35. Wolski R., Spring N.T., Hayes J. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing // Future Generation Computing Systems. 1999. Vol. 15, No. 5–6. P. 757–768.
36. Sacerdoti F.D., Katz M.J. et al. Wide area cluster monitoring with Ganglia // Proceedings of IEEE International Conference on Cluster Computing, Hong Kong, December 1–4, 2003. IEEE Computer Society, 2003. P. 289–298.
37. NIST Special Publication 800–145. A NIST Definition of Cloud Computing. URL: http://csrc.nist.gov/publications/nistpubs/800–145/SP800–145.pdf (дата обращения: 01.02.2012).
38. A. Streit. UNICORE: Getting to the heart of Grid technologies // eStrategies. Vol. 3. 2009. P. 8–9.
39. A. Streit. UNICORE – What lies beneath Grid functionality? // eStrategies. Vol. 7. 2008. P. 38–39.
40. Casavant T., Kuhl J. A Taxonomy of Scheduling in General–purpose Distributed Computing Systems // IEEE Transactions on Software Engineering. 1988.Vol. 14, No. 2. P. 141–154.
41. Braun R., Siegel H. et al. A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems // Journal of Parallel and Distributed Computing. 2001. Vol. 61, No. 6, P. 810–837.
42. Casanova H., Legrand A. et al. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments // Proceedings of the 9th heterogeneous Computing Workshop (HCW'00), Cancun, Mexico, May 1, 2000. IEEE Computer Society, 2000. P. 349–363.
43. You S.Y., Kim H.Y. et al. Task Scheduling Algorithm in GRID Considering Heterogeneous Environment // Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '04), Nevada, USA, June 21–24, 2004. CSREA Press, 2004. P. 240–245.
44. Kurowski K., Ludwiczak B. et al. Improving Grid Level Throughput Using Job Migration And Rescheduling // Scientific Programming. 2004. Vol. 12, No. 4. P. 263–273.
45. Takefusa A., Matsuoka S. et al. A Study of Deadline Scheduling for Client–Server Systems on the Computational Grid // Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC–10'01), San Francisco, California USA, August 7–9, 2001. IEEE Computer Society, 2001. P. 406–415.
46. Chen H., Maheswaran M. Distributed Dynamic Scheduling of Composite Tasks on Grid Computing Systems // Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002), Fort Lauderdale, Florida USA, April 15-19, 2002. IEEE Computer Society, 2002. P. 88–97.
47. Muthuvelu N., Liu J. et al. A Dynamic Job Grouping-Based Scheduling for Deploying Applications with Fine–Grained Tasks on Global Grids // Proceedings of the 3rd Australasian Workshop on Grid Computing and e–Research (AusGrid 2005), Newcastle, Australia, January 30 – February 4, 2005. Australian Computer Society, 2005. — P. 41–48.

48. Wright D. Cheap Cycles from the Desktop to the Dedicated Cluster: Combining Opportunistic and Dedicated Scheduling with Condor. URL: http://www.linuxclustersinstitute.org/conferences/archive/2001/PDF/wright_wisc.pdf (дата обращения: 04.08.2014).
49. Chapin S. J., Katramatos D. et al. The Legion Resource Management System // Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '99), San Juan, Puerto Rico, April 16, 1999. Springer, Lecture Notes in Computer Science, 1999. Vol. 1659. P. 162–178.
50. Rotithor H. G. Taxonomy of Dynamic Task Scheduling Schemes in Distributed Computing Systems // Proceedings on Computer and Digital Techniques, January 1994. Vol. 141, No. 1. P. 1–10.
51. James H. A. Scheduling in Metacomputing Systems, Ph.D. Thesis. The Department Of Computer Science, University of Adelaide, Australia. 1999. URL: http://digital.library.adelaide.edu.au/dspace/bitstream/2440/19450/1/09phj274.pdf (дата обращения: 04.08.2014).
52. Aggarwal A. K., Kent R. D. An Adaptive Generalized Scheduler for Grid Applications // Proceedings of the 19th Annual International Symposium on High Performance Computing Systems and Applications (HPCS'05). Guelph, Ontario Canada, May 15–18, 2005). IEEE Computer Society, 2005. P. 15–18.
53. Sabin G., Kettimuthu R. et al. Scheduling of Parallel Jobs in a Heterogeneous Multi–Site Environment // Proceedings of the 9th International Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'03), Seattle, WA, USA, June 24, 2003). Springer, Lecture Notes in Computer Science, 2003. Vol. 2862. P. 87–104.
54. Arora M., Das S.K., Biswas R. A Decentralized Scheduling and Load Balancing Algorithm for Heterogeneous Grid Environments // Proceedings of International Conference on Parallel Processing Workshops (ICPPW'02). Vancouver, British Columbia Canada, August 20-23, 2002). IEEE Computer Society, 2002. P. 499–505.
55. El-Rewini H., Lewis T., Ali H. Task Scheduling in Parallel and Distributed Systems. Prentice Hall, 1994. 290 p.
56. Buyya R., Abramson D., Venugopal S. The Grid Economy // Proceedings of the IEEE. 2005. Vol. 93, No. 3. P. 698–714.
57. Buyya R., Giddy J., Abramson D. An Evaluation of Economy–based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications // Proceedings of the 2nd International Workshop on Active Middleware Services (AMS 2000), Pittsburgh, USA, August 1, 2000. Kluwer Academic Press, 2000. Vol. 583. P. 221–230.
58. Yu J., Buyya R., Tham C.K. QoS–based Scheduling of Workflow Applications on Service Grids // Proceedings of the 1st IEEE International Conference on e–Science and Grid Computing (e–Science'05), Melbourne, Australia, December 5–8, 2005. IEEE Computer Society, 2005. P. 1–9.
59. Venugopal S., Buyya R. A Deadline and Budget Constrained Scheduling Algorithm for eScience Applications on Data Grids // Proceedings of 6th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP–2005), Melbourne, Australia, October 2-3, 2005. Springer, Lecture Notes in Computer Science, 2005. P. 60–72.

60. Ernemann C., Hamscher V., Yahyapour R. Economic Scheduling in Grid Computing // Proceedings of 8th Workshop on Job Scheduling Strategies for Parallel Processing, in conjunction with HPDC/GGF 5, Edinburgh, Scotland, UK, July 24, 2002. Springer, Lecture Notes in Computer Science, 2002. P. 128–152.
61. Zhu Y., Xiao L. et al. Incentive–based P2P Scheduling in Grid Computing // Proceedings of the 3rd International Conference on Grid and Cooperative Computing (GCC2004), Wuhan, China, October 21-24, 2004. Springer, Lecture Notes in Computer Science, 2004. P. 209–216.
62. Young L., McGough S. et al. Scheduling Architecture and Algorithms within the ICENI Grid Middleware // Proceedings of UK e–Science All Hands Meeting, Nottingham, UK, September 2003). IOP Publishing Ltd., 2003. P. 5–12.
63. Kim S., Weissman J.B. A Genetic Algorithm Based Approach for Scheduling Decomposable Data Grid Applications // Proceedings of the 2004 International Conference on Parallel Processing (ICPP'04), Montreal, Quebec Canada, August 15–18, 2004). IEEE Computer Society, 2004. P. 406–413.
64. Song S., Kwok Y., Hwang K. Security–Driven Heuristics and A Fast Genetic Algorithm for Trusted Grid Job Scheduling // Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), Denver, Colorado USA, April 25–29, 2005. IEEE Computer Society, 2005. P. 65–74.
65. Spooner D.P., Cao J. et al. Localised Workload Management using Performance Prediction and QoS Contracts // UK Performance Engineering Workshop (UKPEW 2002): Proceedings of the 18th Annual Workshop, Glasgow, UK, July 10–11, 2002. University of Glasgow, 2002. P. 69–80.
66. Liu Y. Survey on Grid Scheduling. Department of Computer Science, University of Iowa. 2004.
67. Abraham A., Buyya R. and Nath B. Nature's Heuristics for Scheduling Jobs on Computational Grids // Proceedings of 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000), Cochin, India, December 14–16, 2000. IEEE Computer Society, 2000. P. 45–52.
68. Shan H., Oliker L. et al. Scheduling in Heterogeneous Grid Environments: The Effects of Data Migration // Proceedings of ADCOM2004: International Conference on Advanced Computing and Communication, Ahmedabad Gujarat, India, December 15–18, 2004. IEEE Computer Society, 2004. P. 1–8.
69. He X., Sun X., Laszewski G. A QoS Guided Min–Min Heuristic for Grid Task Scheduling // Journal of Computer Science and Technology: Special Issue on Grid Computing. 2003. Vol. 18, No. 4. P. 442–451.
70. Wu M., Shu W., Zhang H. Segmented Min–Min: A Static Mapping Algorithm for Meta–Tasks on Heterogeneous Computing Systems // Proceedings of the 9th Heterogeneous Computing Workshop (HCW'00), Cancun, Mexico, May 1, 2000. IEEE Computer Society, 2000. P. 375–385.
71. Subramani V., Kettimuthu R. et al. Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests // Proceedings of 11th IEEE Symposium on High Performance Distributed Computing (HPDC 2002), Edinburgh, Scotland, July 23–26, 2002. IEEE Computer Society, 2002. P. 359–366.

72. Silva D.P., Cirne W., Brasileiro F.V. Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids // Proceedings of Euro–Par 2003, Klagenfurt, Austria, August 26–29, 2003. Springer, Lecture Notes in Computer Science, 2003. P. 169–180.
73. Radulescu A., Gemund A.J.C. On the Complexity of List Scheduling Algorithms for Distributed Memory Systems // Portland, Oregon, USA, November 13–19, 1999). IEEE Computer Society, 1999. P. 68–75.
74. Sakellariou R., Zhao H. A Low-cost Rescheduling Policy for Efficient Mapping of Workflows on Grid Systems // Scientific Programming. 2004. Vol. 12, No. 4. P. 253–262.
75. Topcuoglu H., Hariri S., Wu M.Y. Performance–Effective and Low–Complexity Task Scheduling for Heterogeneous Computing // IEEE Transactions on Parallel and Distributed Systems. 2002. Vol. 13, No. 3. P. 260–274.
76. Zhao H., Sakellariou R. An Experimental Investigation into the Rank Function of the Heterogeneous Earliest Finish Time Scheduling Algorithm // Proceedings of Euro–Par 2003, Klagenfurt, Austria, August 26–29, 2003. Springer, Lecture Notes in Computer Science, 2003. P. 189–194.
77. Sakellariou R., Zhao H. A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems // Proceedings of 18th International Parallel and Distributed Processing Symposium (IPDPS'04), Santa Fe, New Mexico USA, April 26–30, 2004. IEEE Computer Society, 2004. P. 111–123.
78. Ma T., Buyya R. Critical–Path and Priority based Algorithms for Scheduling Workflows with Parameter Sweep Tasks on Global Grids // Proceedings of the 17th International Symposium on Computer Architecture and High Performance Computing, Rio de Janeiro, Brazil, October 24–27, 2005. IEEE Computer Society, 2005. P. 251–258.
79. Darbha S., Agrawal D.P. Optimal Scheduling Algorithm for Distributed Memory Machines // IEEE Transactions on Parallel and Distributed Systems. January 1998. Vol. 9, No. 1. P. 87–95.
80. Ranaweera S., Agrawal D.P. A Task Duplication Based Scheduling Algorithm for Heterogeneous Systems // Proceedings of 14th International Parallel and Distributed Processing Symposium (IPDPS'00), Cancun, Mexico, May 1–5, 2000. IEEE Computer Society, 2005. P. 445–450.
81. Bajaj R., Agrawal D. P. Improving Scheduling of Tasks in A Heterogeneous Environment // IEEE Transactions on Parallel and Distributed Systems. 2004. Vol. 15, No. 2. P. 107–118.
82. Gerasoulis A., Yang T. A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors // Journal of Parallel and Distributed Computing, 1992. Vol. 16, No. 4. P. 276–291.
83. Liou J., Palis M.A. An Efficient Task Clustering Heuristic for Scheduling DAGs on Multiprocessors // Proceedings of the Eighth IEEE Symposium on Parallel and Distributed Processing (SPDP 1996), New Orleans, Louisiana, USA, October 23–26, 1996. IEEE Computer Society, 1996. P. 152–156.
84. Liou J., Palis M.A. A Comparison of General Approaches to Multiprocessor Scheduling // Proceedings the 11th International Parallel Processing Symposium (IPPS '97), Geneva, Switzerland, April 1–5, 1997. IEEE Computer Society, 1996. P. 152–156.

85. Kim S.J., Browne J.C. A general approach to mapping of parallel computation upon multiprocessor architectures // Proceedings of the International Conference on Parallel Processing (ICPP'88), The Pennsylvania State University, University Park, PA, USA, August 1988). Pennsylvania State University Press., 1988. Vol. 3. P. 1–8.
86. Sarkar V. Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors. Cambridge, MA: The MIT Press, 1989. P. 215.
87. Shamakina A.V., Sokolinsky L.B. Formal'naja model' zadanija v raspredelennyh vychislitel'nyh sredah [A mathematical model of a job in distributed computing environments]. Parallelnye vychislitelnye tekhnologii (PaVT'2014): Trudy mezhdunarodnoj nauchnoj konferentsii (Rostov-na-Donu, 1–3 aprelya 2014) [Parallel Computational Technologies (PCT'2010): Proceedings of the International Scientific Conference (Rostov-on-Don, Russia, April 1–3, 2014)]. Chelyabinsk, Publishing of the South Ural State University, 2014. P. 343–354.
88. Radchenko G.I., Sokolinsky L.B., Shamakina A.V. Modeli i metody profilirovanija i ocenki vremeni vypolnenija potokov rabot v superkomp'juternyh sistemah [Models and methods of profiling and evaluation of workflow runtime in supercomputing systems] // Vychislitel'nye metody i programmirovanie: Novye vychislitel'nye tehnologii [Numerical Methods and Programming: New computing technologies]. 2013. Vol. 14, No. 4. P. 96–103.
89. Shamakina A.V. Broker resursov dlja podderzhki problemno-orientirovannyh sred [Brokering service for supporting problem-oriented grid environments] // Vestnik JUUrGU. Serija "Vychislitel'naja matematika i informatika" [Bulletin of the South Ural State University. Series "Computational Mathematics and Software Engineering"]. 2012. Vol. 46(305). No. 1. P. 88–98.
90. Shamakina A.V. Organizacija brokera resursov v sisteme CAEBeans [Organization of a resource broker in the CAEBeans system] // Vestnik JUzhno-Ural'skogo gosudarstvennogo universiteta. Serija "Matematicheskoe modelirovanie i programmirovanie" [Bulletin of the South Ural State University. The series "Mathematical Modelling, Programming and Computer Software"]. 2008. Vol. 27(127). No. 2. P. 110–116.
91. Shamakina A.V. CAEBeans Broker: broker resursov sistemy CAEBeans [CAEBeans Broker: a resource broker of the CAEBeans system] // Vestnik JUzhno-Ural'skogo gosudarstvennogo universiteta. Serija "Matematicheskoe modelirovanie i programmirovanie" [Bulletin of the South Ural State University. The series "Mathematical Modelling, Programming and Computer Software"]. 2010. Vol 16(192). No. 5. P. 107–115.
92. Savchenko D.I., Radchenko G.I. DiVTB Server: sreda vypolnenija virtual'nyh jeksperimentov [DiVTB Server: a runtime environment for virtual experiments]. Parallelnye vychislitelnye tekhnologii (PaVT'2013): Trudy mezhdunarodnoj nauchnoj konferentsii (Cheljabinsk, 1–5 aprelya 2013) [Parallel Computational Technologies (PCT'2010): Proceedings of the International Scientific Conference (Chelyabinsk, Russia, April 1–5, 2013)]. Chelyabinsk, Publishing of the South Ural State University, 2013. P. 532–539.
93. Radchenko G.I. Raspredelennye virtual'nye ispytatel'nye stendy: ispol'zovanie sistem inzhenernogo proektirovanija i analiza v raspredelennyh vychislitel'nyh sredah [Distributed virtual test beds: the use of CAE-systems in distributed computing environments] // Vestnik JUzhno-Ural'skogo gosudarstvennogo universiteta. Serija "Matematicheskoe modelirovanie i programmirovanie" [Bulletin of the South Ural State University. The

series "Mathematical Modelling, Programming and Computer Software"]. 2011. Vol. 37. P. 108–121.

94. Radchenko G.I. Tehnologija postroenija problemno-orientirovannyh ierarhicheskih obolochek nad inzhenernymi paketami v grid-sredah [Technology of building problem-oriented hierarchical shells over engineering packages in Grid environments] // Sistemy upravlenija i informacionnye tehnologii [Management systems and information technologies]. 2008. Vol. 4(34). P. 57–61.
95. Radchenko G.I., Sokolinsky L.B. Tehnologija postroenija virtual'nyh ispytatel'nyh stendov v raspredelennyh vychislitel'nyh sredah [Technology of building virtual test beds in distributed computing environments] // Nauchno-Tehnicheskii Vestnik Informatsionnykh Tekhnologii, Mekhaniki i Optiki [Scientific and Technical Journal of Information Technologies, Mechanics and Optics]. 2008. Vol. 54. P. 134–139.
96. Radchenko G.I. Metody organizacii grid-obolochek sistemnogo sloja v tehnologii CAEBeans [Methods of organizing grid shells of a system layer in the CAEBeans technology] // Vestnik JUzhno-Ural'skogo gosudarstvennogo universiteta. Serija "Matematicheskoe modelirovanie i programmirovanie" [Bulletin of the South Ural State University. The series "Mathematical Modelling, Programming and Computer Software"]. 2008. Vol. 15. P. 69–80.
97. Radchenko G.I. Grid-sistema CAEBeans: integracija resursov inzhenernyh paketov v raspredelennye vychislitel'nye sredy [The CAEBeans grid system: an integration of resources of CAE-package in distributed computing environments] // Vestnik Nizhegorodskogo universiteta im. N.I. Lobachevskogo [Bulletin of the Nizhny Novgorod University. N.I. Lobachevsky]. 2009. Vol. 6-1. P. 192–202.
98. Savchenko D., Radchenko G. Mjolnirr: private PaaS as distributed computing evolution // MIPRO 2014: Proceedings of the 37th International Convention (Opatija, Croatia, May 26–30, 2014). IEEE Computer Society, 2014. P. 386–391.

Received 5 August 2014.

УДК 004.432

АВТОМАТИЧЕСКОЕ ОТОБРАЖЕНИЕ ПРОГРАММ НА ЯЗЫКЕ ФОРТРАН НА КЛАСТЕРЫ С ГРАФИЧЕСКИМИ ПРОЦЕССОРАМИ[1]

В.А. Бахтин, М.С. Клинов, А.С. Колганов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула

В статье рассматриваются результаты использования системы автоматизации распараллеливания САПФОР для распараллеливания последовательных программ на кластеры с графическими ускорителями, в том числе программ с регулярными зависимостями по данным. Система переводит программу на языке Fortran в программу на языке Fortran DVMH. Полученная программа запускается на кластере. Язык Fortran DVMH, компиляторы для него и средства отладки входят в состав DVM-системы. Рассмотрены проведенные преобразования исходных программ. Получены параллельные программы, использующие различные технологии параллельного программирования. Приведены характеристики полученных текстов. Приведены экспериментальные данные об эффективности выполнения программ на графических и универсальных процессорах кластера К-100.

Ключевые слова: DVM for Heterogeneous systems, Fortran DVMH, гибридные системы с ускорителями, графические процессоры, Fortran, OpenMP.

Введение

Сложность программирования для кластеров с графическими ускорителями делает актуальными исследования по автоматизации программирования.

Изначально система САПФОР [1] была реализована для кластеров с универсальными процессорами, при этом в качестве промежуточного языка программирования использовался язык Fortran DVM. Для мультипроцессора система САПФОР позволяет получить программы на языке Fortran OpenMP. При реализации отображения на графические ускорители в системе САПФОР используется язык Fortran DVMH [2], который представляет собой расширение языка Fortran DVM.

В настоящее время исследуются возможности расширения языка Fortran DVMH для поддержки ускорителей Intel Xeon Phi. Такая преемственность промежуточных языков существенно упрощает разработку систем для отображения последовательных программ на новые архитектуры ЭВМ, при этом многие блоки системы находят новое применение и используются повторно.

В 2012 году были получены результаты распараллеливания при помощи системы САПФОР двух приложений из области гидродинамики на кластер с графическими ускорителями: двумерная и трехмерная задачи о течении несжимаемой жидкости или слабосжимаемого газа около прямоугольной выемки. В этих задачах в качестве математической модели используется гиперболический вариант квазигазодинамической системы [3].

[1] Статья рекомендована к публикации программным комитетом Международной суперкомпьютерной конференции "Научный сервис в сети Интернет: многообразие суперкомпьютерных миров – 2014".

В ходе данного исследования система САПФОР была опробована на двух приложениях из области лазерных технологий обработки материалов: двумерная и трехмерная задачи моделирования процессов плавления многокомпонентных порошков при селективном лазерном спекании на основе многокомпонентной и многофазной гидродинамической модели [4]. В этих задачах в порошке имеются поры, заполненные газом. В процессе плавления легкоплавкой компоненты порошка образуется жидкость. Будем называть в статье эти задачи Спекание2D и Спекание3D соответственно.

Помимо этого система САПФОР была развита для поддержки распараллеливания программ с регулярными зависимостями по данным между витками циклов и протестирована на примере алгоритма численного решения краевой задачи для уравнения Лапласа методом последовательной верхней релаксации. В этой программе на витках цикла есть зависимость от данных, вычисляемых на соседних витках. Выполнение таких циклов может быть реализовано в виде конвейера. Организация конвейера для графических ускорителей отличается от организации конвейера для универсальных процессоров, потому что ключевым моментом для ускорителей является обеспечение согласованного доступа к памяти устройства. Соответствующая поддержка таких циклов была сделана на уровне языка Fortran DVMH, в компиляторе с этого языка и в библиотеке функций системы поддержки выполнения DVMH-программ. Такому развитию системы была посвящена статья [5]. Без этого развития было бы невозможно осуществить отображение последовательных программ на кластер с графическими ускорителями в системе САПФОР. В этом плане отображение параллельных программ (на языке Fortran DVMH) на кластер с графическими ускорителями является частью отображения последовательных программ на этот кластер.

Для получения результатов распараллеливания этих программ потребовалось развитие системы САПФОР, которое было выполнено. Внесены изменения в блоки анализа, в блок построения вариантов распараллеливания на кластер с графическими ускорителями. Также были выполнены некоторые преобразования распараллеливаемых программ. Имеются в виду такие преобразования, после которых программа остается последовательной (без каких-либо указаний о ее распараллеливании), но которая уже может быть преобразована системой распараллеливания в эффективную параллельную программу. Более подробно преобразования будут рассмотрены далее.

На данный момент такие преобразования делает пользователь системы САПФОР, а не сама система. При этом подразумевается, что пользователь системы обладает навыками программирования на языке Fortran: либо является автором распараллеливаемой программы, либо специалистом по ее преобразованию. В дальнейшем планируется, что подобные преобразования будут частично или полностью автоматизированы.

В состав системы САПФОР входит диалоговая оболочка и автоматически распараллеливающий компилятор, сокращенно АРК. Для данного исследования использовался как раз АРК. Показ причин, по которым не удавалось эффективно распараллелить исходную программу (без преобразований), не был развит для поддержки данных входных программ и не тестировался на них. В рамках данного исследования исследовались возможности системы распараллеливания по эффективному распараллеливанию этих программ как таковые. Улучшение диалоговой оболочки для этих и других программ из этого класса программ (похожих на этих) будет следующим этапом.

1. Распараллеливание приложений Спекание2D и Спекание3D при помощи автоматически распараллеливающего компилятора

Рассмотрим процесс распараллеливания.

Для приложения Спекание2D у программиста имеется последовательная Fortran-программа из 872 строк, которая подается на вход АРК. В процессе работы компилятора создается файл с информацией о ходе распараллеливания. Эта информация является достаточно подробной, но и в силу этого объемной. Она не представляется удобной для конечного пользователя. Данное исследование проводилось разработчиками системы САПФОР, как следствие они были способны разобраться в объемной диагностике процесса распараллеливания программы. По диагностике были определены причины, по которым циклы программы остались не распараллелены или распараллелены неэффективно. По этим причинам были определены фрагменты программы, которые требовалось изменить в последовательной программе, она была изменена и снова подана на вход АРК.

После изменения программа для приложения Спекание2D стала занимать 1007 строк.

После преобразования текст программы для Спекания3D возрос с 901 строки до 1555 строк.

В целом преобразования хоть и объемные, но не сложные. Они не требуют знания о распараллеливании программы, в отличие от распараллеливания программистом программы. Во многих местах преобразования аналогичные, за счет чего выполняются быстро, хоть и требуют определенного навыка. Полученная программа является последовательной, написана на том же языке, что и исходная, на Fortran. Корректность преобразований проверяется по мере их накопления путем сравнения результатов выполнения исходной и преобразованной программы. Это позволяет не ошибиться программисту на данном этапе.

Рассмотрим преобразования одного фрагмента из этих программ. Эти преобразования оказались самыми сложными из тех, с которыми сталкивались авторы статьи (не только на примере этих программ), при подготовке входных программ для АРК.

Речь идет о фрагменте программы, показанной на рис. 1, в котором некоторые детали были опущены (заменены на "...").

С точки зрения распараллеливания во внешнем цикле (по i и по j) есть OUTPUT-зависимость (запись с разных витков в одну ячейку памяти). Ее можно устранить путем заведения дополнительного массива, который будет использоваться для переноса значений, на которые в данном случае надо изменить элемент массива. Дополнительный массив будет привязан своими индексами к значениям переменных цикла на каждом витке. Адресуя его элементы можно брать значения с конкретных витков.

В таком случае в данном цикле не будут записываться элементы целевого массива, а будет заполняться дополнительный массив. Запись в целевой будет сделана в последующих циклах для того, чтобы обеспечить такой же порядок суммирования в один элемент массива, который был в исходной (не преобразованной) программе. Например, заполнение дополнительного массива показано на рис. 2.

А запись в целевой массив будет выглядеть как показано на рис. 3.

```
do j=1,Nz
do i=1,Nx
  do jj=0,1
  do ii=0,1
    iloc(1)=1-2*ii
    jloc(1)=0

    iloc(2)=0
    jloc(2)=1-2*jj

    i0=i-1+ii
    j0=j-1+jj
    do na=1,2
      i1=i0+iloc(na)
      j1=j0+jloc(na)

      bs(na,1)=x(i1,j1)-x(i0,j0)
      bs(na,2)=z(i1,j1)-z(i0,j0)
    end do
    …
    flx(i,j-1+jj)=flx(i,j-1+jj)-iloc(1)*fl(1)
    flz(i-1+ii,j)=flz(i-1+ii,j)-jloc(2)*fl(2)
  end do
  end do

end do
end do
```

Рис. 1. Подлежащий преобразованию фрагмент исходного кода

```
    tmp_arr(i,j,ii,jj,1)=iloc(1)*fl(1)
    tmp_arr(i,j,ii,jj,2)=jloc(2)*fl(2)
```

Рис. 2. Заполнение дополнительного массива

```
do j=1,Nz
do i=1,Nx
  flx(i,j)=flx(i,j)-tmp_arr(i,j,0,1,1)
  flx(i,j)=flx(i,j)-tmp_arr(i,j,1,1,1)
end do
end do
```

Рис. 3. Запись в целевой массив

```
if(na.eq.1) then
  if(ii.eq.0) then
    if(jj.eq.0) then
      x_i1_j1=x(i,j-1)
      …
```

Рис. 4. Способ избавления от косвенной индексации

Вторая особенность рассматриваемого фрагмента связана с косвенной индексацией элементов массивов на чтение. На самом деле выполняются операции чтения соседних элементов, но для системы распараллеливания это должно быть извлекаемым из текста программы. В данном случае это возможно через замену косвенной индексации на явную. Для этого заведены вспомогательные скалярные переменные, которые будут присвоены в начале каждого витка и использованы на нем. В скалярные переменные будут присвоены значения массивов с косвенной индексацией в зависимости от значений ите-

рационных переменных циклов на витке. Например, для обращений x(i1,j1) будет использована переменная x_i1_j1, как показано на рис. 4.

Характеристики исходных и преобразованных программ приведены в табл. 1. Под программой с обозначением Fortran APK подразумевается программа на языке Fortran, подаваемая на вход APK. Процентное соотношение приведено относительно исходной Fortran-программы.

Таблица 1

Характеристики программ Спекание2D и Спекание3D

Программы	Характеристики	Fortran	Fortran APK
Спекание2D	Общее число строк	872	1007 (115%)
	Добавлено строк	–	135 (15%)
	Изменено строк	–	23 (2,6%)
Спекание3D	Общее число строк	901	1555 (173%)
	Добавлено строк	–	654 (73%)
	Изменено строк	–	91 (10%)

После распараллеливания приложений с помощью APK были получены тексты программ на языке Fortran OpenMP, Fortran DVM и Fortran DVMH. Характеристики полученных программ приведены в табл. 2.

Таблица 2

Размер различных версий программ Спекание2D и Спкание3D, в строках

Программы	Fortran APK	Fortran OpenMP	Fortran DVM	Fortran DVMH
Спекание2D	1007	1035	1068	1206
Спекание3D	1555	1617	1648	1845

В табл. 3 и 4 приведены времена выполнения полученных версий программ на одном узле кластера К-100 [6] при использовании различного числа ядер и графических ускорителей(GPU). Для сравнения приведено время работы автоматически распараллеленной программы при помощи компилятора Intel (опция -parallel).

Компилятор Intel не справляется с автоматическим распараллеливанием данных программ. Получаемые параллельные программы практически не ускоряются при увеличении числа используемых ядер. Основная проблема — приватные массивы в циклах (рабочие массивы, зависимость по которым не мешает выполнять цикл параллельно, если для каждого процесса/нити использовать свой экземпляр таких переменных). Установить факт, что тот или иной массив можно сделать приватным, достаточно сложная задача для статического анализатора (как правило, он указывает на наличие зависимости между витками цикла, что приводит к его последовательному выполнению). В системе САПФОР реализован механизм спецкомментариев для передачи информации о свойствах программы, извлечь которые на этапе анализа невозможно или очень трудоемко. Все приватные массивы были указаны в соответствующих спецком-

ментариях, что позволило получить более эффективные варианты параллельных программ.

Таблица 3

Времена выполнения программы Спекание2D (1000x1000, ITMAX=100) на кластере K-100, в секундах

	Последовательная	Intel -parallel	APK OpenMP	APK DVM	APK DVMH
1 ядро	19,23	19,72	20,53	25,96	20,16
6 ядер	–	20,41	4,79	6,52	5,48
12 ядер	–	20,81	3,41	4,49	3,72
1 GPU	–	–	–	–	12,03
2 GPU	–	–	–	–	10,01
3 GPU	–	–	–	–	7,17

Таблица 4

Времена выполнения программы Спекание3D (100x100x100, ITMAX=100) на кластере K-100, в секундах

	Последовательная	Intel -parallel	APK OpenMP	APK DVM	APK DVMH
1 ядро	53,01	58,54	50,03	128,75	56,56
6 ядер	–	51,05	24,11	25,59	13,61
12 ядер	–	50,46	23,80	13,41	8,46
1 GPU	–	–	–	–	6,43
2 GPU	–	–	–	–	4,94
3 GPU	–	–	–	–	4,21

На 1 ядре универсального процессора DVM-варианты программ выполняются медленнее остальных. Например, DVM-вариант программы Спекание3D выполняется в 2.4 раза дольше последовательного варианта. Причина — линеаризация распределенных между процессами массивов, выполняемая компилятором Fortran DVM. Память для элементов таких массивов выделяется системой поддержки выполнения DVM-программ. Для локальной секции массива на каждом процессоре память отводится в соответствии с форматом распределения данных и с учетом теневых граней. Все ссылки к элементам распределенных массивов вида ARRAY(I,J,K) заменяются компилятором на ссылки вида BASE(ARRAY_OFFSET+I+COEFF_ARRAY1*J+COEFF2*K), где BASE – это база, относительно которой адресуются все распределяемые массивы; ARRAY_OFFSET – смещение начала массива относительно базы; COEFF_ARRAY1, COEFF_ARRAY2 – коэффициенты адресации распределенного массива. Такая программа хуже распознается и оптимизируется стандартными Fortran компиляторами.

DVMH-версия программы лишена данного недостатка. Для каждого параллельного цикла компилятор с языка Fortran DVMH гененерирует CUDA-обработчик для вычислений на GPU, а также хост-обработчик для выполнения на универсальных ядрах процессора. Обработчик – подпрограмма, осуществляющая выполнение части параллельного цикла на конкретном устройстве. Распределенные массивы передаются в хост-обработчики как параметры — массивы, перенимающие размер (в формальном параметре вместо верхней границы последней размерности указывается звездочка “*”). Такой подход позволяет не линеаризовать обращения к массивам внутри хост-обработчиков. В результате время работы DVMH-программы на 1 ядре практически совпадает с временем работы последовательной программы.

OpenMP-вариант для приложения Спекание2D выполняется на 1 узле кластера быстрее других вариантов. DVMH-вариант на 1 GPU выполняется быстрее, чем на 1 ядре универсального процессора (например, последовательная программа), но на 3 GPU (1 узел кластера K-100) уступает 12 ядрам универсального процессора (также 1 узел кластера K-100).

Для Спекания3D OpenMP-вариант не самый лучший для ядер универсального процессора, а DVMH-вариант обгоняет на 3-х GPU 12 ядер универсального процессора, тем самым, является самым быстрым вариантом для случая использования 1 узла кластера K-100.

При увеличении размера задачи можно ожидать большего выигрыша от использования графических ускорителей вплоть до получения выигрыша на нескольких узлах K-100.

Для всех параллельных программ (полученных автоматически через APK или через распараллеливающий компилятор от Intel) была проверена корректность работы путем сравнения результатов работы программ с последовательным вариантом. Относительная погрешность менее $1*10^{-9}$.

2. Распараллеливание программы, содержащей цикл с регулярными зависимостями по данным между витками при помощи автоматически распараллеливающего компилятора

В качестве такой программы взята программа, реализующая метод последовательной верхней релаксации (Successive over-relaxation — программа SOR).

Текст последовательной программы на языке Fortran занимает 48 строк. Без каких-либо изменений он подавался на вход APK и распараллеливающему компилятору Intel (с опцией -parallel).

При помощи APK были получены тексты программ на языках Fortran OpenMP, Fortran DVM, Fortran DVMH.

Времена выполнения программ на кластере K-100 приведены в табл. 5. В программе использовались массивы размера 800x800x800 и было выполнено 50 итераций.

Intel-компилятор не ускорил выполнение программы SOR.

OpenMP-вариант программы выполняется на 1 узле кластера K-100 быстрее других вариантов.

DVMH-вариант программы на 1 GPU выполняется почти в 2 раза быстрее, чем лучший вариант на 1 ядре универсального процессора. Лучшее время для DVMH-версии программы дает использование 3-х GPU, по сравнению с использованием 12 ядер универсального процессора.

Таблица 5

Времена выполнения программы SOR на кластере K-100, в секундах

	Последовательная	Intel -parallel	APK OpenMP	APK DVM	APK DVMH
1 ядро	88,96	89,00	88,98	90,12	89,02
6 ядер	–	88,99	15,58	32,59	30,67
12 ядер	–	88,99	7,91	22,87	22,47
1 GPU	–	–	–	–	48,34
2 GPU	–	–	–	–	26,82
3 GPU	–	–	–	–	20,33

Заключение

Использование системы автоматизации распараллеливания упрощает процесс разработки программ для современных ЭВМ, многие из которых имеют различного вида вычислительные устройства в своем составе. Особенно в случае автоматизации преобразований. В настоящий момент над последовательными программами требуется выполнить определенные преобразования, которые препятствуют их распараллеливанию системой САПФОР. Эти преобразования не содержат вставок никаких указаний о распараллеливании. В этом плане программа остается последовательной, написанной на языке Fortran.

Получены результаты распараллеливания на кластер с графическими ускорителями двух приложений из области лазерных технологий обработки материалов (Спекание2D и 3D) и программы SOR, которая содержит цикл с регулярными зависимостями по данным между витками. Использование 1 GPU на приложении Спекание2D и SOR дает ускорение на кластере K-100 относительно использования 1 ядра универсального процессора. На приложении Спекание3D использование полученной с помощью системы САПФОР программы на языке Fortran DVMH на графических ускорителях позволяет получить наилучшее время выполнения на 1 узле кластера K-100 относительно вариантов программ, полученных с помощью системы САПФОР на языках Fortran OpenMP, Fortran DVM или полученных компилятором Intel программ в модели OpenMP.

В настоящий момент производится улучшение системы САПФОР для распараллеливания реальных приложений, содержащих циклы с регулярными зависимостями по данным: тесты NAS Parallel Benchmarks LU, BT, SP.

Исследование выполнено при финансовой поддержке грантов РФФИ №13-07-00580, 14-01-00109 и Программ фундаментальных исследований президиума РАН №15, №16 и №18.

Литература

1. Бахтин, В.А. Диалог с программистом в системе автоматизации распараллеливания САПФОР. / В.А. Бахтин, И.Г. Бородич, Н.А. Катаев, М.С. Клинов, Н.В. Ковалева, В.А. Крюков, Н.В. Поддерюгина. // Вестник Нижегородского университета им. Н.И. Лобачевского. — Н. Новгород: Изд-во ННГУ, 2012. — №5 (2) . — С. 242–245.
2. Бахтин, В.А. Расширение DVM-модели параллельного программирования для кластеров с гетерогенными узлами / В.А. Бахтин, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула, Ю.Л. Сазанов // Вестник Южно-Уральского государственного университета, серия «Математическое моделирование и программирование». — Челябинск: Издательский центр ЮУрГУ, 2012. — № 18 (277), вып. 12 — С. 82–92.
3. Бахтин, В.А. Распараллеливание с помощью DVM-системы некоторых приложений гидродинамики для кластеров с графическими процессорами / В.А. Бахтин, И.Г. Бородич, Н.А. Катаев, М.С. Клинов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула, Ю.Л. Сазанов // Научный сервис в сети Интернет: поиск новых решений: Труды Международной суперкомпьютерной конференции (17-22 сентября 2012 г., г. Новороссийск). — М.: Изд-во МГУ, 2012. — С. 444–450.
4. Низьев, В.Г. Численное моделирование плавления двухкомпонентных порошков при лазерном спекании / В.Г. Низьев, А.В. Колдоба, Ф.Х. Мирзаде, В.Я. Панченко, Ю.А. Повещенко, М.В. Попов // Математическое моделирование. — 2011. — Т. 23, №4. — С. 90–102.
5. Алексахин, В.Ф. Распараллеливание на графические процессоры тестов NAS NPB 3.3.1 на языке Fortran DVMH / В.Ф. Алексахин, В.А. Бахтин, О.Ф. Жукова, А.С. Колганов, В.А. Крюков, Н.В. Поддерюгина, М.Н. Притула, О.А. Савицкая, А.В. Шуберт // Параллельные вычислительные технологии (ПаВТ'2014): труды международной научной конференции (1-3 апреля 2014 г., г. Ростов-на-Дону). — Челябинск: Издательский центр ЮУрГУ, 2014. — С. 30–41.
6. Гибридный вычислительный кластер К-100. URL: http://www.kiam.ru/MVS/resourses/k100.html (дата обращения 04.08.2014).

Бахтин Владимир Александрович, кандидат физико-математических наук, Институт прикладной математики им. М.В. Келдыша РАН (г. Москва, Российская Федерация), bakhtin@keldysh.ru.

Клинов Максим Сергеевич, кандидат физико-математических наук, Институт прикладной математики им. М.В. Келдыша РАН (г. Москва, Российская Федерация), klinov@keldysh.ru.

Колганов Александр Сергеевич, Институт прикладной математики им. М.В. Келдыша РАН (г. Москва, Российская Федерация), alex-w900i@yandex.ru.

Крюков Виктор Алексеевич, доктор физико-математических наук, профессор, Институт прикладной математики им. М.В. Келдыша РАН (г. Москва, Российская Федерация), krukov@keldysh.ru.

Поддерюгина Наталия Викторовна, кандидат физико-математических наук, Институт прикладной математики им. М.В. Келдыша РАН (г. Москва, Российская Федерация), konov@keldysh.ru.

Притула Михаил Николаевич, кандидат физико-математических наук, Институт прикладной математики им. М.В. Келдыша РАН (г. Москва, Российская Федерация), pritmick@yandex.ru.

Поступила в редакцию 4 августа 2014 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2014, vol. 3, no. 3, pp. 86–96

AUTOMATIC MAPPING OF FORTRAN PROGRAMS ONTO CLUSTERS WITH GRAPHICS PROCESSING UNITS

V.A. Bakhtin, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation),

M.S. Klinov, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation),

A.S. Kolganov, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation),

V.A. Krukov, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation),

N.V. Podderyugina, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation),

M.N. Pritula, Keldysh Institute of Applied Mathematics Russian Academy of Sciences (Moscow, Russian Federation),

The paper shows results of exploitation of automated parallelization system SAPFOR for sequential programs parallelization for cluster with GPUs, including programs with regular data dependencies. The system translates Fortran program into Fortran DVMH program. Produced program can be executed on cluster. Fortran DVMH language, its compilers and debugging tools are bundled within DVM-system. Performed transformations with original programs are considered. Programs which use a variety of parallel programming technologies are emitted by the system. Source code characteristics are considered. Performance analysis is done on CPUs and GPUs of the K-100 computational cluster.

Keywords: DVM for Heterogeneous systems, Fortran DVMH, hybrid computational systems with accelerators, GPU, Fortran, OpenMP.

References

1. Bakhtin V.A., Borodich I.G., Kataev N.A, Klinov M.S., Kovaleva N.V., Krukov V.A., Podderyugina N.V. Dialog s programmistom v sisteme avtomatizacii rasparallelivania SAPFOR [Dialog with programmer in automated parallelization system SAPFOR]. Vestnik Nizhegorodskogo universiteta im. N.I. Lobachevskogo [Vestnik of Lobachevsky State University of Nizhni Novgorod]. Nizhni Novgorod, Nizhni Novgorod University Press, 2012. No. 5 (2). P. 242–245.

2. Bakhtin V.A., Klinov M.S., Krukov V.A., Podderyugina N.V., Pritula M.N., Sazanov Y.L. Rasshirenie DVM-modeli parallel'nogo programmirovaniya dlya klasterov s geterogennymi uzlami [Extension of the DVM parallel programming model for clusters with heterogeneous nodes]. Vestnik Yuzhno-Ural'skogo gosudarstvennogo universiteta, seriya "Matematicheskoe modelirovanie i programmirovanie". Chelyabinsk, Publishing of the South Ural State University, 2012. No. 18 (277), Issue 12. P. 82–92.
3. Bakhtin V.A., Borodich I.G., Kataev N.A., Klinov M.S., Krukov V.A., Podderyugina N.V., Pritula M.N., Sazanov Y.L. Rasparallelivanie s pomosch'yu DVM-sistemy nekotoryh prilozheniy gidrodinamiki dlya klasterov s graficheskimi processorami [Parallelization of some CFD applications for clusters with graphics processing units using DVM-system]. Nauchniy servis v seti Internet: poisk novyh resheniy: Trudy Mezhdunarodnoy superkomp'yuternoy konferencii (17–22 sentyabrya 2012, Novorossiysk) [Science service on Internet: search for new decisions: Proceedings of International supercomputer conference (17–22 of September, 2012, Novorossiysk)]. Moscow, Moscow University Press, 2012. P. 444–450.
4. Niziev V.G., Koldoba A.V., Mirzade F.H., Panchenko V.Y., Poveschenko Y.A., Popov M.V. Chislennoe modelirovanie plavleniya dvuhkomponentnyh poroshkov pri lazernom spekanii [Numerical modeling of melting process of two-component powders in laser agglomeration]. Matematicheskoe modelirovanie [Mathematical modeling]. 2011. Vol. 23, No. 4. P. 90–102.
5. Aleksahin V.F., Bakhtin V.A., Zhukova O.F., Kolganov A.S., Krukov V.A., Podderyugina N.V., Pritula M.N., Savitskaya O.A., Shubert A.V. Rasparallelivanie na graficheskie processory testov NAS NPB 3.3.1 na yazyke Fortran DVMH [Parallelization of NAS NPB 3.3.1 benchmarks on graphics processing units using Fortran DVMH language]. Parallel'nye vychislitel'nye tehnologii (PaVT'2014): trudy mezhdunarodnoy nauchnoy konferencii (1–3 aprelya 2014, Rostov-na-Donu) [Parallel computing technologies (PaCT'2014): proceedings of international science conference (1–3 of April, 2014, Rostov-on-Don)]. Chelyabinsk, Publishing of the South Ural State University, 2014. P. 30–41.
6. Gibridniy vychislitel'niy klaster K-100 [Hybrid computational cluster K-100]. URL: http://www.kiam.ru/MVS/resourses/k100.html (accessed 04.08.2014).

Received 4 August 2014.

УДК 519.816

КОМПЬЮТЕРНАЯ РЕАЛИЗАЦИЯ ОПЕРАЦИЙ С НЕЧЕТКИМИ ЧИСЛАМИ

Е.Р. Галлямов, В.И. Ухоботов

В данной работе рассматривается проблема представления нормальных нечетких чисел в виде, приемлемом для компьютерной реализации. В частности, приведено исследование арифметики нечетких чисел с использованием множеств уровня нечеткого числа, а также освещены возникшие проблемы и пути их решения, связанные с необходимостью получения в качестве результата операции нечеткого числа с параметрами, удовлетворяющими заданным в работе ограничениям. В рамках данного исследования был реализован программный пакет для работы с нечеткими числами с L-R представлением, основанный на исследованных в работе операциях. Реализация пакета подразумевает возможность его дальнейшего расширения и использования в других научных исследованиях по близкой тематике.

Ключевые слова: арифметика нечетких чисел, нечеткие числа, компьютерная реализация.

Введение

Для целого класса экономических и социальных задач информация о присутствующих в них параметрах и переменных носит нечеткий характер. Поэтому для их описания используются нечеткие числа. Получаемое нечеткое решение для таких задач дает возможность изначально учитывать неполноту и неточность исходных данных.

Нужно отметить, что с момента публикации Л.А. Заде своей работы по нечетким множествам [1] вышло большое количество работ, в которых, так или иначе, рассматривались арифметические операции с нечеткими числами (см., например, [2–8]).

Целью работы является исследование и обоснование основных операций над нечеткими числами, имеющими определенное представление, а также разработка библиотеки, реализующей эти операции.

Статья организована следующим образом. В первом разделе даны результаты анализа предметной области. Во втором разделе приводятся результаты исследования арифметики нечетких чисел, множества уровня которых являются отрезками, а также рассматриваются основные проблемы, возникшие при обосновании данных операций. Третий раздел посвящен аспектам компьютерной реализации нечеткого числа и исследованных операций. В заключении суммируются основные результаты работы.

1. Анализ предметной области

Пусть заданы множество X в обычном смысле, которое принято называть универсальным, и функция μ_A: $X \to [0,1]$. Нечетким множеством A [1] универсального множества X называется совокупность пар вида $(x|\ \mu_A(x))$, где $x \in X$.

Функция μ_A: $X \to [0,1]$ называется функцией принадлежности нечеткого множества A, а значение $\mu_A(x)$ на конкретном элементе $x \in X$ называется степенью (или мерой) принадлежности этого элемента нечеткому множеству A.

Для каждого числа $\alpha \in [0,1]$ обычное множество $A(\alpha) = \{x \in X: \mu_A(x) \geq \alpha\}$ называется множеством уровня нечеткого множества A. Эти множества уровня удовлетворяют следующим свойствам:

$$A(0) = X;\ 0 \le \alpha \le \beta \le 1 \Rightarrow A(\beta) \subset A(\alpha);\ \ 0 < \alpha \le 1 \Rightarrow \bigcap_{0 \le t < \alpha} A(t) = A(\alpha). \qquad (1)$$

Пусть при каждом $0 \le \alpha \le 1$ определено множество $A(\alpha) \subset X$. Если семейство этих множеств $A(\alpha) \subset X$ удовлетворяет свойствам (1), то оно является семейством множеств уровня нечеткого множества A, функция принадлежности которого равна

$$\mu_A(x) = sup\{\alpha \in [0,1] : x \in A(\alpha)\}.$$

Поэтому нечеткое множество A универсального множества X можно задать семейством множеств $A(\alpha) \subset X$ при каждом $0 \le \alpha \le 1$, удовлетворяющим свойствам (1).

Нечеткие множества, универсальным множеством которых является множество действительных чисел R, называются нечеткими числами.

В данной работе рассматриваются нечеткие числа, множества уровня которых имеют следующий вид:

$$A(\alpha) = [a + \varepsilon f(\alpha); b - \delta\varphi(\alpha)]. \qquad (2)$$

Здесь функции $f, \varphi : [0,1] \to [0,1]$ удовлетворяют следующим свойствам:

$$\begin{gathered} f(0) = \varphi(0) = 0,\ f(1) = \varphi(1) = 1; \\ 0 \le \alpha_1 < \alpha_2 \le 1 \Rightarrow f(\alpha_1) \le f(\alpha_2), \varphi(\alpha_1) \le \varphi(\alpha_2); \\ \lim_{t \to \tau - 0} f(t) = f(\tau),\ \lim_{t \to \tau - 0} \varphi(t) = \varphi(\tau). \end{gathered} \qquad (3)$$

Графическое представление таких нечетких чисел приведено на рис. 1.

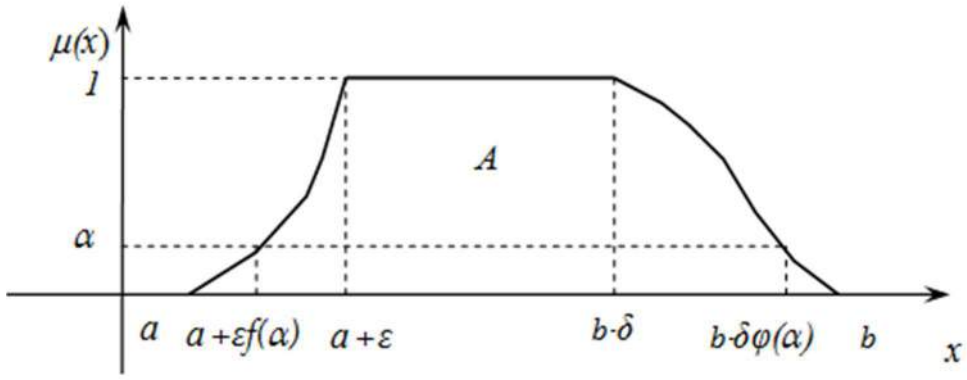


Рис. 1. Графическое представление нечеткого числа

Интервал $(a; b)$ задает множество, на котором функция принадлежности нечеткого числа принимает значения больше нуля, а на отрезке $[a + \varepsilon; b - \delta]$ значение функции принадлежности равно 1. Коэффициенты $a, b, \varepsilon, \delta$ удовлетворяют следующим условиям:

$$a + \varepsilon \le b - \delta, \delta \ge 0, \varepsilon \ge 0. \qquad (4)$$

При выполнении условий (3) и (4) множества уровня (2) удовлетворяют свойствам (1).

2. Арифметика нечетких чисел с LR-представлением

При определении арифметических операций над нечеткими числами вида (2) будем руководствоваться следующим соображением. Множества уровня, полученного в результате примененной арифметической операции, нечеткого числа должны соответствовать виду (2) с условиями (3) и (4).

2.1. Сложение двух нечетких чисел

Результатом сложения двух нечетких чисел A_i будет являться нечеткое число A с множеством уровня вида

$$A(\alpha) = A_1(\alpha) + A_2(\alpha) = [a_1 + \varepsilon_1 f_1(\alpha); b_1 - \delta_1 \varphi_1(\alpha)] + [a_2 + \varepsilon_2 f_2(\alpha); b_2 - \delta_2 \varphi_2(\alpha)].$$

Результатом сложения двух отрезков является отрезок (см. [10])

$$A(\alpha) = [a_1 + a_2 + \varepsilon_1 f_1(\alpha) + \varepsilon_2 f_2(\alpha); b_1 + b_2 - \delta_1 \varphi_1(\alpha) - \delta_2 \varphi_2(\alpha)].$$

Таким образом, результатом сложения двух нечетких чисел вида (2) является нечеткое число того же вида, параметры которого имеют вид

$$a = a_1 + a_2,\ \ \varepsilon = \varepsilon_1 + \varepsilon_2,\ b = b_1 + b_2,\ \delta = \delta_1 + \delta_2,$$
$$f(\alpha) = \frac{\varepsilon_1 f_1(\alpha) + \varepsilon_2 f_2(\alpha)}{\varepsilon_1 + \varepsilon_2}\ , \varphi(\alpha) = \frac{\delta_1 \varphi_1(\alpha) + \delta_2 \varphi_2(\alpha)}{\delta_1 + \delta_2}.$$

Нетрудно проверить, что, если параметры исходных нечетких чисел удовлетворяют условиям (3) и (4), то этим условиям удовлетворяют и параметры их суммы.

2.2. Умножение нечеткого числа на действительное число

Пусть заданы отрезок $[c, C] \subset \mathbb{R}$ и число $\lambda \in \mathbb{R}$. Результатом их умножения $\lambda \times [c, C]$ будет являться отрезок $[\lambda c, \lambda C]$ при $\lambda \geq 0$ и $\lambda \times [c, C] = [\lambda C, \lambda c]$, если $\lambda < 0$. Отсюда следует, что результатом умножения $\lambda \times A$ нечеткого числа (2) на положительное число λ является нечеткое число A_* вида (2) с параметрами

$$a_* = \lambda a, b_* = \lambda b, \varepsilon_* = \lambda \varepsilon, \delta_* = \lambda \delta, f_*(\alpha) = f(\alpha), \varphi_*(\alpha) = \varphi(\alpha).$$

Если $\lambda < 0$, то $a_* = \lambda b, b_* = \lambda a, \varepsilon_* = -\lambda \delta, \delta_* = -\lambda \varepsilon, f_*(\alpha) = \varphi(\alpha), \varphi_*(\alpha) = f(\alpha)$.

При $\lambda = 0$ результатом умножения является нечеткое число, у которого

$$a_* = 0, b_* = 0, \varepsilon_* = 0, \delta_* = 0, f_*(\alpha) = f(\alpha), \varphi_*(\alpha) = \varphi(\alpha).$$

Легко проверить, что, если параметры исходного нечеткого числа удовлетворяют условиям (3) и (4), то этим условиям удовлетворяют параметры произведения $\lambda \times A$.

2.3. Умножение нечетких чисел

Результатом умножения двух нечетких чисел A_i будет являться нечеткое число A с множеством уровня вида

$$A(\alpha) = A_1(\alpha) \times A_2(\alpha) = [a_1 + \varepsilon_1 f_1(\alpha); b_1 - \delta_1 \varphi_1(\alpha)] \times [a_2 + \varepsilon_2 f_2(\alpha); b_2 - \delta_2 \varphi_2(\alpha)]. \quad (5)$$

Дальнейший анализ формулы (5) требует разбиения множества нечетких чисел на различные классы. Это обусловлено спецификой операции умножения отрезков [10].

Определение 1. Нечеткое число (2) назовем неотрицательным, если $a \geq 0$.

Определение 2. Нечеткое число (2) назовем неположительным, если $b \leq 0$.

Определение 3. Нечеткое число (2) назовем числом с нулем первого типа, если $a \leq a + \varepsilon \leq 0 \leq b - \delta \leq b$.

Определение 4. Нечеткое число (2) назовем числом с нулем второго типа, если $a \leq 0 \leq a + \varepsilon \leq b - \delta \leq b$.

Определение 5. Нечеткое число (2) назовем числом с нулем третьего типа, если $a \leq a + \varepsilon \leq b - \delta \leq 0 \leq b$.

2.3.1. Умножение неотрицательных нечетких чисел

В рассматриваемом случае $0 \le a_1 < b_1$ и $0 \le a_2 < b_2$. Поэтому формула (5) принимает вид

$$A(\alpha) = [(a_1 + \varepsilon_1 f_1(\alpha)) \cdot (a_2 + \varepsilon_2 f_2(\alpha)); (b_1 - \delta_1 \varphi_1(\alpha)) \cdot (b_2 - \delta_2 \varphi_2(\alpha))].$$

Отсюда следует, что множество $A(\alpha)$ представимо в виде (2) при

$$a = a_1 a_2,\ \varepsilon = \varepsilon_1 a_2 + \varepsilon_2 a_1 + \varepsilon_1 \varepsilon_2, b = b_1 b_2,\ \delta = \delta_1 b_2 + \delta_2 b_1 - \delta_1 \delta_2,$$

$$f(\alpha) = \frac{\varepsilon_1 a_2 f_1(\alpha) + \varepsilon_2 a_1 f_2(\alpha) + \varepsilon_1 \varepsilon_2 f_1(\alpha) f_2(\alpha)}{\varepsilon_1 a_2 + \varepsilon_2 a_1 + \varepsilon_1 \varepsilon_2},$$

$$\varphi(\alpha) = \frac{\delta_1 b_2 \varphi_1(\alpha) + \delta_2 b_1 \varphi_2(\alpha) - \delta_1 \delta_2 \varphi_1(\alpha) \varphi_2(\alpha)}{\delta_1 b_2 + \delta_2 b_1 - \delta_1 \delta_2}.$$

Нетрудно проверить, что, если параметры исходных нечетких чисел удовлетворяют условиям (3) и (4), то этим условиям удовлетворяют и параметры их произведения.

2.3.2. Умножение нечеткого числа с нулем первого типа на неотрицательное нечеткое число

В рассматриваемом случае выполнены неравенства $a_1 < a_1 + \varepsilon_1 \le 0 \le b_1 - \delta_1 \le b_1$ и $a_2 \ge 0$. Поэтому формула (5) принимает вид

$$A(\alpha) = \left[(a_1 + \varepsilon_1 f_1(\alpha)) \cdot (b_2 - \delta_2 \varphi_2(\alpha)); (b_1 - \delta_1 \varphi_1(\alpha))(b_2 - \delta_2 \varphi_2(\alpha))\right].$$

Отсюда следует, что множество $A(\alpha)$ представимо в виде (2) при

$$a = a_1 b_2,\ \varepsilon = -\delta_2 a_1 + \varepsilon_1 b_2 - \varepsilon_1 \delta_2,\ b = b_1 b_2,\ \delta = \delta_1 b_2 + \delta_2 b_1 - \delta_1 \delta_2,$$

$$f(\alpha) = \frac{-\delta_2 a_1 \varphi_2(\alpha) + \varepsilon_1 b_2 f_1(\alpha) - \varepsilon_1 \delta_2 f_1(\alpha) \varphi_2(\alpha)}{-\delta_2 a_1 + \varepsilon_1 b_2 - \varepsilon_1 \delta_2},$$

$$\varphi(\alpha) = \frac{\delta_1 b_2 \varphi_1(\alpha) + \delta_2 b_1 \varphi_2(\alpha) - \delta_1 \delta_2 \varphi_1(\alpha) \varphi_2(\alpha)}{\delta_1 b_2 + \delta_2 b_1 - \delta_1 \delta_2}.$$

Можно показать, что, если параметры исходных нечетких чисел удовлетворяют условиям (3) и (4), то этим условиям удовлетворяют и параметры их произведения.

2.3.3. Умножение нечеткого числа с нулем второго типа на неотрицательное нечеткое число

В рассматриваемом случае выполнены неравенства $a_1 \le 0 < a_1 + \varepsilon_1 \le b_1 - \delta_1 \le b_1$ и $a_2 \ge 0$. Поэтому множество (5) равно

$$\left[(a_1 + \varepsilon_1 f_1(\alpha)) \cdot (b_2 - \delta_2 \varphi_2(\alpha)); (b_1 - \delta_1 \varphi_1(\alpha)) \cdot (b_2 - \delta_2 \varphi_2(\alpha))\right] \text{ при } a_1 + \varepsilon_1 f_1(\alpha) \le 0$$

и

$$\left[(a_1 + \varepsilon_1 f_1(\alpha)) \cdot (a_2 + \varepsilon_2 f_2(\alpha)); (b_1 - \delta_1 \varphi_1(\alpha)) \cdot (b_2 - \delta_2 \varphi_2(\alpha))\right] \text{ при } a_1 + \varepsilon_1 f_1(\alpha) > 0.$$

Отсюда следует, что множество $A(\alpha)$ представимо в виде (2) при

$$a = a_1 b_2, b = b_1 b_2, \varepsilon = (a_1 a_2 + a_1 \varepsilon_2 + a_2 \varepsilon_1 + \varepsilon_1 \varepsilon_2) - a_1 b_2, \delta = b_1 \delta_2 + b_2 \delta_1 - \delta_1 \delta_2.$$

$$\varphi(\alpha) = \frac{\delta_1 b_2 \varphi_1(\alpha) + \delta_2 b_1 \varphi_2(\alpha) - \delta_1 \delta_2 \varphi_1(\alpha) \varphi_2(\alpha)}{\delta_1 b_2 + \delta_2 b_1 - \delta_1 \delta_2},$$

$$f(\alpha) = \frac{-\delta_2 a_1 \varphi_2(\alpha) + \varepsilon_1 b_2 f_1(\alpha) - \varepsilon_1 \delta_2 f_1(\alpha) \varphi_2(\alpha)}{\varepsilon} \text{ при } a_1 + \varepsilon_1 f_1(\alpha) \le 0$$

и

$$f(\alpha) = \frac{\left(a_1a_2 + a_1\varepsilon_2 f_2(\alpha) + a_2\varepsilon_1 f_1(\alpha) + \varepsilon_1\varepsilon_2 f_1(\alpha)f_2(\alpha)\right) - a_1b_2}{\varepsilon} \text{ при } a_1 + \varepsilon_1 f_1(\alpha) > 0.$$

Можно показать, что, если параметры исходных нечетких чисел удовлетворяют условиям (3) и (4), то этим условиям удовлетворяют и параметры их произведения.
Аналогичным образом рассматривается операция умножения двух нечетких чисел и в оставшихся случаях. Нетрудно подсчитать, что в случае пяти выделенных классов нечетких чисел для обоснования операции умножения необходимо рассмотреть пятнадцать различных случаев. Однако данное число можно сократить до шести в силу сводимости ряда случаев к уже рассмотренным. К примеру, случай умножения двух неположительных нечетких чисел сводится к случаю умножения неотрицательных нечетких чисел с помощью умножения обоих нечетких чисел на -1.

2.4. Деление нечетких чисел

Операция деления $A_1 : A_2$ двух нечетких чисел определена для случая строгой положительности или строгой отрицательности числа A_2. Это значит, что $a_2 > 0$ или $b_2 < 0$.

В этом случае нечеткое число $\tilde{A} = \frac{1}{A_2}$ представимо в виде (2). Рассмотрим случай строго положительного нечеткого числа A_2. Тогда

$$\tilde{A}(\alpha) = \left[\frac{1}{b_2 - \delta_2\varphi_2(\alpha)}; \frac{1}{a_2 + \varepsilon_2 f_2(\alpha)}\right].$$

Для определения параметров в формуле (2) имеем соотношения

$$\tilde{a} + \tilde{\varepsilon}\tilde{f}(\alpha) = \frac{1}{b_2 - \delta_2\varphi_2(\alpha)}, \quad \tilde{b} - \tilde{\delta}\tilde{\varphi}(\alpha) = \frac{1}{a_2 + \varepsilon_2 f_2(\alpha)}. \tag{6}$$

Положим в формулах (6) $\alpha = 0$. Получим $\tilde{a} = \frac{1}{b_2}$, $\tilde{b} = \frac{1}{a_2}$. Полагая в (6) $\alpha = 1$, будем иметь

$$\tilde{\varepsilon} = \frac{1}{b_2 - \delta_2} - \frac{1}{b_2} > 0, \tilde{\delta} = \frac{1}{a_2} - \frac{1}{a_2 + \varepsilon_2} > 0.$$

Далее, из неравенств $a_2 + \varepsilon_2 > 0, b_2 - \delta_2 > 0, a_2 + \varepsilon_2 < b_2 - \delta_2$ следует, что $\tilde{a} + \tilde{\varepsilon} \leq \tilde{b} - \tilde{\delta}$. Подставим значения параметров $\tilde{a}, \tilde{\varepsilon}, \tilde{b}, \tilde{\delta}$ в формулы (6). Получим, что

$$\tilde{f}(\alpha) = \frac{\frac{1}{b_2 - \delta_2\varphi_2(\alpha)} - \tilde{a}}{\tilde{\varepsilon}} = \frac{(b_2 - \delta_2)\varphi_2(\alpha)}{b_2 - \delta_2\varphi_2(\alpha)},$$

$$\tilde{\varphi}(\alpha) = \frac{\tilde{b} - \frac{1}{a_2 + \varepsilon_2 f_2(\alpha)}}{\tilde{\delta}} = \frac{(a_2 + \varepsilon_2)f_2(\alpha)}{a_2 + \varepsilon_2 f_2(\alpha)}.$$

Нетрудно проверить, что $\tilde{\varphi}(0) = 0, \tilde{\varphi}(1) = 1$, $\tilde{f}(0) = 0, \tilde{f}(1) = 1$. Можно показать, что функции $\tilde{f}(\alpha)$ и $\tilde{\varphi}(\alpha)$ монотонно возрастают.

Случай со строго отрицательным нечетким числом A_2 сводится к рассмотренному путем умножения A_2 на -1.

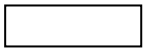
3. Программная реализация нечеткого числа и арифметических операций

Реализуемый программный пакет ориентирован на построение приложений, работающих в среде CLR платформы .NET. Также возможно использование данного пакета для исследований и проведения вычислений с нечеткими числами в различных пакетах математического моделирования с поддержкой работы с .NET сборками. В качестве главного инструментария была выбрана интегрированная среда разработки Microsoft Visual Studio 2012. Язык реализации – C#.

Классы, реализующие логику работы с нечеткими числами, являются частью ядра реализуемого пакета и располагаются в отдельной сборке *FuzzyCore.dll*.

Основываясь на описании объектов предметной области и принципах объектно-ориентированного проектирования, можно построить частичную диаграмму классов, реализующих логику работы с нечеткими числами (рис. 2).

При построении диаграммы использовались следующие обозначения в соответствии со стандартами UML:

 -класс;  - отношение использования;  - наследование.

Класс *LeftRightFuzzyNumber* (рис. 2) представляет собой нечеткое число с LR-представлением. Свойства (иначе property) *Left, Right, LeftFuzzynessCoefficient, RightFuzzynessCoefficient* реализуют соответственно параметры $a, b, \varepsilon, \delta$ нечеткого числа. Для создания экземпляра класса *LeftRightFuzzyNumber* необходимо воспользоваться одним из двух публичных конструкторов (public constructor). В данном случае, два конструктора предусмотрены для сохранения гибкости инстанцирования нечеткого числа. В первом случае для создания экземпляра класса необходимы параметры $a, b, \varepsilon, \delta$, а также функция принадлежности нечеткого числа, удовлетворяющая виду (2), переданная в виде делегата типа *Func<double,double>*. Во втором случае, кроме параметров $a, b, \varepsilon, \delta$, необходимо передать функцию принадлежности, заданную в виде массива точек.

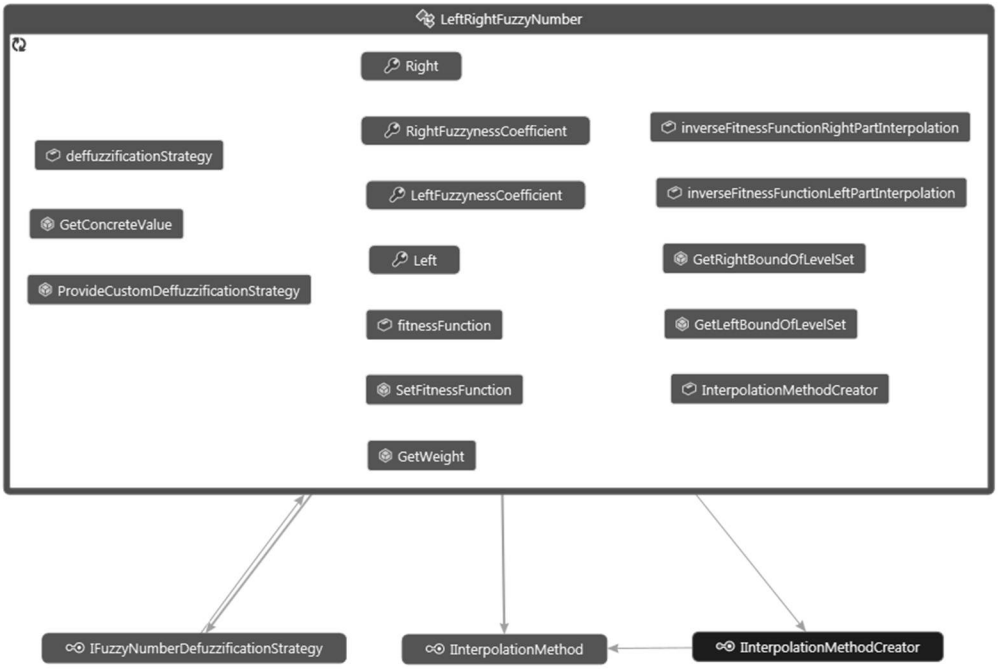


Рис. 2. Частичная диаграмма классов для работы с нечеткими числами

Теперь подробнее разберем методы, реализованные в рамках класса *LeftRightFuzzyNumber*.

Метод *GetWeight* необходим для получения значения функции принадлежности числа для некоторого переданного аргумента. В случае инстанцирования нечеткого числа функцией принадлежности, заданной в виде конечного набора точек, возникла проблема обеспечения возможности получения «веса» точки не только для ограниченного набора заданных аргументов, но для всей числовой оси. Решением данной проблемы стало интерполирование функции принадлежности. Как видно из частичной диаграммы классов, класс *LeftRightFuzzyNumber* агрегирует реализацию интерфейса *IInterpolationMethodCreator,* который, являясь реализацией паттерна «фабричный метод», при вызове метода *Create* возвращает экземпляр класса, наследующего интерфейс *IInterpolationMethod.* Назначение любого класса, реализующего контракт *IInterpolationMethod*, заключается в построении интерполяции функции по набору эталонных аргументов и значений функции в них. Так как задача интерполяции не относится непосредственно к теме нечетких чисел и множеств, интерфейсы *IInterpolationMethodCreator, IInterpolationMethod* и их классы-реализации помещены в сборку *MathExtensions.dll* в рамках пакета FuzzySolutions. В рамках пакета предоставляется два метода интерполяции (рис. 3). По умолчанию класс нечеткого числа использует фабричный метод, возвращающий экземпляр класса *LinearSplineInterpolationMethod* с функционалом интерполяции линейными сплайнами.

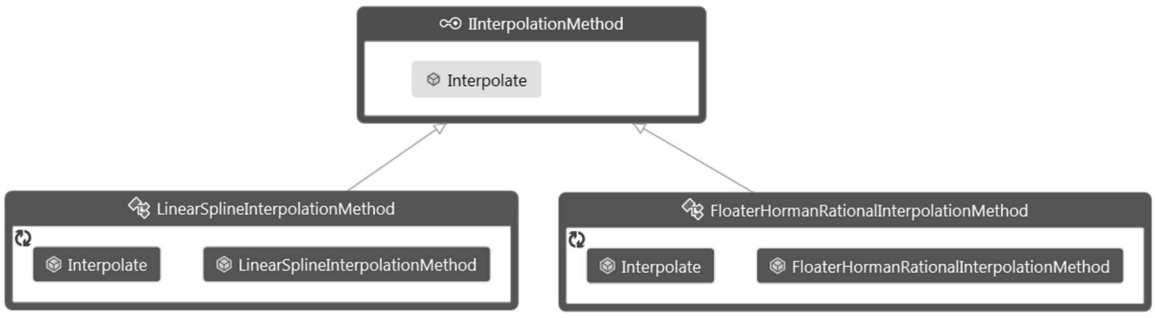


Рис. 3. Диаграмма классов для методов интерполяции, реализованных в рамках пакета

Для реализации арифметических операций над нечеткими числами в терминах множеств уровня было необходимо построить функции $f(\alpha)$ и $\varphi(\alpha)$. Так как данные функции не задаются пользователем напрямую, они интерполируются исходя из значений функции принадлежности и вида множества уровня (2). Поля класса, отвечающие за интерполяцию функций $f(\alpha)$ и $\varphi(\alpha)$, являются закрытыми (private) и необходимы также для функций *GetLeftBoundOfLevelSet* и *GetRightBoundOfLevelSet*, возвращающих абсциссы левого и правого краев множества уровня нечеткого числа для переданного значения α −уровня.

После выполнения арифметической операции дополнительно проводится процедура интерполирования функции принадлежности результирующего нечеткого числа исходя из значений функций $f(\alpha)$ и $\varphi(\alpha)$ для набора α −уровней.

Не будем подробно останавливаться на реализации арифметических операций. Скажем только, что все они реализованы в точности с полученными в ходе исследования формулами.

Стоит отметить, что вследствие расположения классов, реализующих алгоритмы интерполяции, в отдельной сборке *MathExtensions.dll*, для работы с нечеткими числами необходима одновременная поставка следующих сборок: *MathExtensions.dll*, *FuzzyCore.dll*, *MathNet.Numercs.dll*.

Одним из способов использования функционала нечетких чисел, реализованного в пакете FuzzySolutions, является создание нечеткого числа и вызов методов данного класса из системы математического моделирования с поддержкой работы с .NET сборками. Таковой системой, к примеру, является Matlab.

Для комфортной работы пользователя возможно «оборачивание» вызовов методов сборки *FuzzyCore.dll* средствами интерпретируемого языка Matlab. Для работы с базовым функционалом нечетких чисел необходимо подключить следующие сборки: *FuzzyCore.dll*, *MathExtensions.dll*, *MathNet.Numercs.dll*.

Приведем примеры некоторых функций, написанных на языке Matlab, для работы с нечеткими числами:

- инстанцирование нечеткого числа (рис. 4): результатом данной операции является экземпляр класса *LeftRightFuzzyNumber*;

```
function fuzzyNumber = createFuzzyNumber(left,leftFuzzynessCoefficient,
                                         right,
                                         rightFuzzynessCoefficient,
                                         fitnessFuncLeftPartValues,
                                         fitnessFuncRightPartValues)
fitnessFuncLeftPartValuesMapLength=length(fitnessFuncLeftPartValues);
fitnessFuncRightPartValuesMapLength=length(fitnessFuncRightPartValues);

fitnessFuncLeftPartValuesDictonary =
             NET.createGeneric('System.Collections.Generic.Dictionary',
                               {'System.Double','System.Double'},1);
fitnessFuncRightPartValuesDictionary =
              NET.createGeneric('System.Collections.Generic.Dictionary',
                               {'System.Double','System.Double'},1);

leftPartKeySet = keys(fitnessFuncLeftPartValues);
leftPartValueSet = values(fitnessFuncLeftPartValues);
for i=1:fitnessFuncLeftPartValuesMapLength
   key = leftPartKeySet{i};
   value = leftPartValueSet{i};
   fitnessFuncLeftPartValuesDictonary.Add(key,value);
end

rightPartKeySet = keys(fitnessFuncRightPartValues);
rightPartValueSet = values(fitnessFuncRightPartValues);
for j=1:fitnessFuncRightPartValuesMapLength
   key = rightPartKeySet{j};
   value = rightPartValueSet{j};
   fitnessFuncRightPartValuesDictionary.Add(key,value);
end

fuzzyNumber = IGS.Fuzzy.Core.LeftRightFuzzyNumber(left,
                                  leftFuzzynessCoefficient,right,
                                  rightFuzzynessCoefficient,
                                  fitnessFuncLeftPartValuesDictonary,
                                  fitnessFuncRightPartValuesDictionary);
```

Рис. 4. Инстанцирование нечеткого числа

- получение значений функции принадлежности с заданным шагом (рис. 5): результатом данной операции являются два массива, первый из которых содержит набор точек, взятых с заданным шагом, а второй значения функции принадлежности числа в данных точках.

Далее приведем пример выполнения операции сложения двух нечетких чисел и построения графика функции принадлежности результирующего нечеткого числа с помощью пакета FuzzySolutions и средств системы Matlab (рис. 6).

```
function [points, values] =
                 getFuzzyNumberFitnessFunctionValues(fuzzyNumber, step)

if step <= 0
   points = [fuzzyNumber.Left,
             fuzzyNumber.Left + fuzzyNumber.LeftFuzzynessCoefficient,
             fuzzyNumber.Right - fuzzyNumber.RightFuzzynessCoefficient,
             fuzzyNumber.Right];
   values = [0.0, 1.0, 1.0, 0.0];
else
   iterationNumber =floor((fuzzyNumber.Right-fuzzyNumber.Left)/step)+1;
   points = zeros(1,iterationNumber);
   values = zeros(1,iterationNumber);
   argument = fuzzyNumber.Left;
   for i=1:iterationNumber
       points(i)=argument;
       values(i)=fuzzyNumber.GetWeight(argument);
       argument = argument + step;
   end
end
```

Рис. 5. Получение значений функции принадлежности с заданным шагом

```
>>asm = NET.addAssembly('C:\MATLAB\MathNet.Numerics.dll');
>>asm1 = NET.addAssembly('C:\MATLAB\MathExtensions.dll');
>>asm2 = NET.addAssembly('C:\MATLAB\FuzzyCore.dll');
>>firstNumber =
     createFuzzyNumber(2.0,3.0,8.0,1.0,
          containers.Map([2.1,2.15,2.2,2.45,2.5,2.65,2.75,3.0,4.2],
                         [0.08,0.12,0.18,0.21,0.43,0.6,0.75,0.8,0.93]),
          containers.Map([7.1,7.5],[0.7,0.15]));
>>secondNumber = createFuzzyNumber(-2.0,1.0,4.5,0.5,
   containers.Map([-1.85,-1.75,-1.7,-1.45,-1.35,-1.2,-1.15,-1.1,-1.05],
                  [0.12,0.15,0.25,0.3,0.43,0.6,0.67,0.78,0.93]),
   containers.Map([4.25,4.1],[0.8,0.15]));
>>result = firstNumber + secondNumber;
>>[points, values]=getFuzzyNumberFitnessFunctionValues(result, 0.01);
>>plot(points, values,'LineWidth',1.5);
  grid on;
```

Рис. 6. Демонстрация выполнения операции сложения двух нечетких чисел

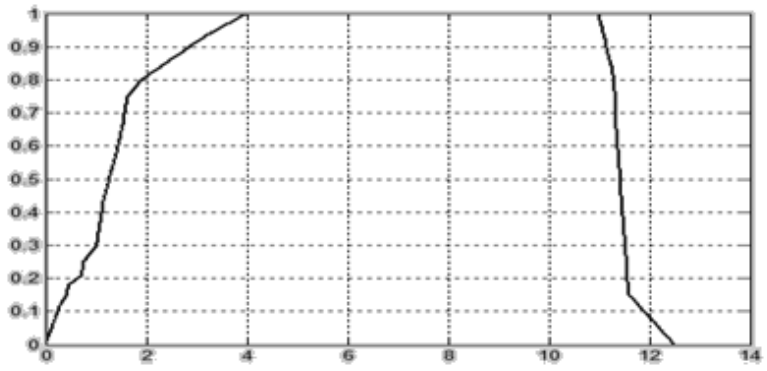


Рис. 7. Функция принадлежности результирующего нечеткого числа

Результатом сложения двух нечетких чисел, первое из которых имело параметры $a_1 = 2, b_1 = 8, \varepsilon_1 = 3, \delta_1 = 1$, а второе – параметры $a_1 = -2, b_1 = 4.5, \varepsilon_1 = 1, \delta_1 = 0.5$, стало нечеткое число, график функции принадлежности которого изображен на рис. 7.

Заключение

В работе предложено специальное представление множеств уровня нечетких чисел и исследованы базовые арифметические операции для нечетких чисел. Было показано, что результат данных операций также является нечетким числом того же самого класса.

Для таких нечетких чисел была разработана программная реализация этих операций в рамках программного пакета FuzzySolutions. Показана возможность интеграции разработанной библиотеки с системой математического моделирования Matlab, что позволяет исследователю напрямую эффективно взаимодействовать с программным кодом сборок из пакета FuzzySolutions.

Исследованное представление нечеткого числа уменьшает трудоемкость выполнения арифметических операций и сложность программной реализации.

Литература

1. Zadeh, L.A. Fuzzy Sets / L.A. Zadeh // Information and Control. — 1965. — Vol. 8. — P. 338–353.
2. Dutta, P. Fuzzy Arithmetic with and without using α-cut method: A Comparative Study / P. Dutta, H. Boruah, T. Ali // International Journal of Latest Trends in Computing — March 2011. — Vol. 2 — P. 99–107.
3. Bansal, A. Trapezoidal Fuzzy Numbers (a,b,c,d): Arithmetic Behavior / A. Bansal // International Journal of Physical and Mathematical Sciences — 2011 — Vol. 2, No. 1 — P. 39–44.
4. Dubois, D. Fuzzy Sets and Systems: Theory and Applications. / D. Dubois — Academic Press, 1980. — 393 p.
5. Леоненков, А.В. Нечеткое моделирование в среде MATLAB и fyzzyTECH. / А.В. Леоненков — СПб.: БХВ-Петербург, 2005. — 731 с.
6. Чернов, В.Г. Основы теории нечетких множеств. Решение задач многокритериального выбора альтернатив. / В.Г. Чернов — Учебное пособие. Владимир, 2005.
7. Ухоботов, В.И. Введение в теорию нечетких множеств и ее приложения. / В.И. Ухоботов — Учебное пособие. Челябинск, 2005. — 133 с.

8. Stefanini, L. Fuzzy Arithmetic with Parametric LR Fuzzy Numbers. / L. Stefanini, L. Sorini // European Society of Fuzzy Logic and Technology Conference, Lisbon, Portugal — 2009. — P. 600–605.
9. Stefanini, L. Representing fuzzy numbers for fuzzy calculus. / L. Stefanini, L. Sorini — Analysis and Design of Intelligent Systems using Soft Computing Techniques — 2007. — Vol. 41 — P. 485–494.
10. Жолен Л., Кифер М., Дидро О., Вольтер Э. / Жолен Л. /Прикладной интервальный анализ. — М.–Ижевск: Институт компьютерных исследований, 2005. — 468 с.

Галлямов Евгений Расимович, магистр прикладной математики и информатики, кафедра теории управления и оптимизации, Челябинский государственный университет (Челябинск, Российская Федерация), gallyamov.evgeniy@gmail.com.

Ухоботов Виктор Иванович, доктор физико-математических наук, профессор, заведующий кафедрой теории управления и оптимизации, Челябинский государственный университет (Челябинск, Российская Федерация), ukh@csu.ru.

Поступила в редакцию 7 июля 2014 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2014, vol. 3, no. 3, pp. 97–108

COMPUTER IMPLEMENTATION OF OPERATIONS WITH FUZZY NUMBERS

E.R. Gallyamov, Chelyabinsk State University (Chelyabinsk, Russian Federation),
V.I. Ukhobotov, Chelyabinsk State University (Chelyabinsk, Russian Federation)

In this paper the problem of representing normal fuzzy numbers in a form suitable for computer implementation is considered. In particular, the results of the investigation of fuzzy arithmetic using level sets of fuzzy numbers are given. Also in the paper the problems associated with necessity of receiving a resulting fuzzy number with the parameters satisfying the given constraints are analyzed. Within this research the software package for work with fuzzy numbers with L-R representation was realized. Implementation of the package gives an opportunity of increasing the capabilities and using in other scientific researches of similar subject.

Keywords: arithmetic of fuzzy numbers, fuzzy numbers, computer implementation.

References

1. Zadeh L.A. Fuzzy Sets // Information and Control. 1965. Vol. 8. P. 338–353.
2. Dutta P., Boruah H., Ali T. Fuzzy Arithmetic with and without using α-cut method: A Comparative Study // International Journal of Latest Trends in Computing. March 2011. Vol. 2. P. 99–107.
3. Bansal A. Trapezoidal Fuzzy Numbers (a,b,c,d): Arithmetic Behavior // International Journal of Physical and Mathematical Sciences. 2011. Vol. 2, No. 1. P. 39–44.
4. Dubois D. Fuzzy Sets and Systems: Theory and Applications. Academic Press, 1980. 393 P.

5. Leonenkov A.V. Nechetkoe modelirovanie v srede MATLAB i fuzzyTECH [Fuzzy Modeling in MATLAB Environment and fuzzyTECH]. SPb.: BHV-Peterburg, 2005. 731 P.
6. Chernov V.G. Osnovy' teorii nechetkix mnozhestv. Reshenie zadach mnogokriterial'nogo vy'bora al'ternativ [Foundations of the Theory of Fuzzy Sets. Solving of Problems of a Multicriteria Choice of Alternatives]. Vladimir, 2005.
7. Ukhobotov V.I. Vvedenie v teoriyu nechetkix mnozhestv i ee prilozheniya [Introduction to the Theory of Fuzzy Sets and Its Applications]. Chelyabinsk, Publishing of Chelyabinsk State University, 2005. 133 P.
8. Stefanini L., Sorini L. Fuzzy Arithmetic with Parametric LR Fuzzy Numbers. European Society of Fuzzy Logic and Technology Conference (Lisbon, Portugal, 2009). P. 600–605.
9. Stefanini L., Sorini L. Representing fuzzy numbers for fuzzy calculus. // Analysis and Design of Intelligent Systems using Soft Computing Techniques. 2007. Vol. 41. P. 485–494.
10. Jaulin L., Kieffer M., Didrit O., Walter E. Applied Interval Analysis. Springer, 2001. 384 P.

Received 7 July 2014.

УДК 004.45

МЕТОДЫ МОДЕЛИРОВАНИЯ И ОЦЕНКИ ПРОИЗВОДИТЕЛЬНОСТИ ОБЛАЧНЫХ СИСТЕМ

П.А. Михайлов, Г.И. Радченко

В процессе использования промышленных грид и облачных систем, возникают вопросы, связанные с внесением изменений в структуру и алгоритмы работы распределенных вычислительных сетей (РВС) и тем как эти изменения отразятся на работоспособности системы. В статье раскрываются основные подходы к методологиям экспериментального исследования РВС, и облачных систем в частности. Оцениваются достоинства и недостатки подходов натурного моделирования, эмуляции, эталонного тестирования и симуляции РВС. Дается краткий обзор систем симуляции РВС. Результатом проведенного анализа является архитектура и реализация прототипа собственной системы моделирования частных облачных PaaS-систем. Раскрываются особенности реализации системы, а также результаты испытания разработанных моделей на облачной платформе Mjolnirr.

Ключевые слова: распределенные вычислительные системы, моделирование, симуляция, облачные вычисления, облака, Mjolnirr.

Введение

В процессе разработки и эксплуатации распределенных вычислительных систем, в особенности облачных вычислительных систем, возникают вопросы, связанные изменением структуры данных систем, алгоритмов их работы и архитектуры. Особый интерес представляет исследование влияния этих изменений на ключевые параметры распределенной вычислительной системы: производительность, безопасность, отказоустойчивость, масштабируемость и др. В настоящее время проводится множество исследований, посвященных изучению поведения больших распределенных систем, а также создается программное обеспечения для осуществления этих исследований. Нами предлагается собственное решение для симуляции алгоритмов планирования в частных облачных платформах, на основе системы CloudSim.

Дальнейшая структура статьи выглядит следующим образом. В разделе 1 представлены моделирования распределенных вычислительных, включая натурное моделирования, эмуляцию, эталонное тестирование и симуляцию таких систем. Раздел 2 посвящен обзору систем симуляции распределенных вычислительных систем, включая такие системы как CloudSim, CDOSim, TeachCloud, SPECI, DCSim. В разделе 3 дается обзор наиболее распространенных подходов к планированию облачных ресурсов: централизованного, иерархического и распределенного планирования. В разделе 4 представлена архитектура и особенности реализации разработанной нами системы моделирования частных облачных PaaS-платформ. В последнем разделе кратко изложены результаты испытания системы моделирования частных облачных PaaS-платформ на примере платформы Mjolnirr. В заключении представлены результаты работы и направления дальнейших исследований.

1. Технологии моделирования распределенных вычислительных систем

В статье [1] предлагается следующая классификация методологий экспериментального исследования крупномасштабных и распределенных вычислительных сетей (рис. 1).

Натурное моделирование — это метод, при котором реальная задача (некоторое приложение, сервис и т.д.) выполняется в реальной вычислительной среде (операционной системе, сервере и др.). Процесс выполнения задачи при натурном моделировании отличается от штатного тем, что в процессе выполнения собирается статистическая информация о работе приложения и состоянии системы. Натурное моделирование применяется в тех случаях, когда сложное поведение и взаимодействие РВС не может быть адекватно проанализировано и смоделировано.

		Вычислительная среда	
		Реальная	*Модельная*
Задача	*Реальная*	Натурное моделирование	Эмуляция
	Модельная	Эталонное тестирование	Симуляция

Рис. 1. Классификация методологий экспериментального исследования РВС

Данная ситуация может возникать в силу различных программных (сложные алгоритмы планирования, резервирования ресурсов, различные стеки базового ПО) или аппаратных особенностей РВС (hyper-threading, управление кэшем, паразитный траффик внутри сети и др.). С другой стороны, для проведения достоверного эксперимента требуется получить полный контроль над РВС, что бывает крайне сложно в реальных условиях. Для решения данной проблемы разрабатываются широкомасштабные вычислительные среды, предназначенные исключительно для проведения экспериментов. Примерами таких систем могут служить Grid'5000 [2] или Planet-Lab [3] и др.

Эмуляция — это подход, при котором реальная задача выполняется в рамках модели вычислительной среды. Основой данного подхода является создание синтетических экспериментальных условий для исполнения реального приложения. Выделяют два типа эмуляторов. Первый тип эмуляторов основывается на технологии виртуальных машин, обеспечивая прослойку между реальным аппаратным обеспечением и исполняемой задачей (например, система Microgrid [4]). Второй тип эмуляторов исполняет задачи непосредственно на базе предоставляемого аппаратного обеспечения. В этом случае, контроль над ним обеспечивается посредством ограничения производительности (сетевых соединений или процессоров).

Эталонное тестирование состоит в том, что пакет моделирования автоматизирует процесс физического моделирования РВС путем формирования синтетических потоков заданий с целью оценки эффективности ее функционирования [5]. Яркими примерами таких методик являются системы HPL Linpack [6] и пакет NAS [6]. Модельное приложение должно охватывать и измерять такие характеристики тестируемой вычислительной среды, как процессорные мощности, пропускная способность сети, скорость ввода/вывода данных и т.д. Главным отличием модельного приложения от реального явля-

ется то, что при выполнении модельного приложения собираются данные о функционировании системы в процессе вычисления, а результаты выполнения самого приложения не имеют значения.

Симуляция — это подход, при котором модельное приложение выполняется в рамках модели вычислительной среды. Симуляция позволяет сфокусировать процесс на моделировании определенной части среды, абстрагируясь от остальной части системы. Такой подход позволяет достичь высокого уровня повторяемых результатов в рамках большого набора различных платформ и экспериментальных условий, позволяя оценить поведение РВС при изменяющихся условиях и, на основе этого, оптимизировать стратегии управления потоками задач. Главным достоинством данного подхода является гибкость системы, так как и приложение и вычислительная среда являются моделями, есть возможность легко изменять условия эксперимента. Минусом же является крайне высокая сложность разработки моделей приложения и вычислительной среды. Примерами средств симуляции могут служить такие проекты, как GridSim [8], SimGrid [9], CloudSim [10] и другие.

2. Обзор систем симуляции распределенных вычислительных систем

В настоящее время проводится множество исследований посвященных изучению поведения больших распределенных систем, а также создается программное обеспечения для осуществления этих исследований. Примерами подобного рода ПО могут выступать GridSim, SimGrid и CloudSim. В то время как первые два решения нацелены на моделирование грид-систем, CloudSim является одной из немногих платформ, ориентированных на моделирование облачных вычислительных систем.

Стоит отметить, что средства моделирования грид-систем являются подходящим решением для создания моделей очень больших облачных вычислительных систем.

Однако виртуализация позволяет разворачивать частные облачные системы на небольших испытательных стенда, где важную роль играет детальное моделирование политик резервирования облачных ресурсов, сервисов, загруженности приложений и др. Поэтому создания детальных моделей облачных вычислительных систем необходимо использовать ПО, разработанное непосредственно для моделирования облачных вычислительных систем.

2.1. Платформа CloudSim

Платформа CloudSim — это обобщенное и масштабируемое средство симуляции, которое позволяет осуществлять полноценное моделирование и симуляцию облачных вычислительных систем и инфраструктур [10, 11]. Она является расширением базовой функциональности платформы GridSim, обеспечивая возможность моделирования хранилища данных, веб-сервисы, распределение ресурсов между виртуальными машинами и др.

На рис. 2 представлена многоуровневая архитектура платформы CloudSim.

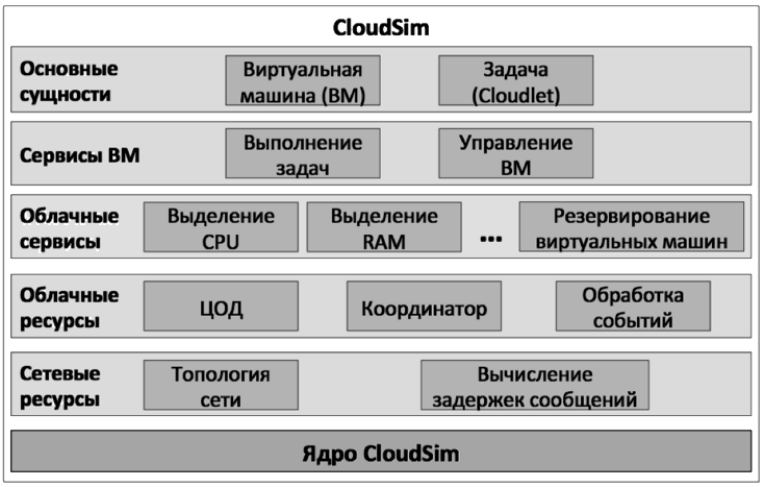


Рис. 2. Архитектура платформы CloudSim

Ядро CloudSim основывается на движке SimJava, который поддерживает такие базовые функции, как использование очередей при обработке событий, создание облачных сущностей (сервисов, узлов, центров обработки данных, брокеров ресурсов и виртуальных машин), взаимодействие между элементами системы и управление протеканием симуляции. При разработке модели облачной среды, пользователю необходимо произвести доработку ключевых для своей модели компонентов для достижения результатов, максимально приближенных к реальности.

Основными сущностями платформы при моделирования являются Виртуальная машина и Задача. Данные компоненты являются специфичными для различных групп облачных систем. Так, например, для облачных систем типа PaaS будет характерно развертывание нескольких приложений на одной виртуальной машине.

Наиболее важными компонентами при моделировании являются компоненты, отвечающие за политики управления ресурсами. К задачам, которые решают эти компоненты относятся следующие:

– выделение процессорных мощностей, оперативной памяти и других ресурсов для различных сущностей моделируемой системы;
– развертывание виртуальных машин на узлах моделируемой системы;
– распределение задач между виртуальными машинами моделируемой системы.

Принцип работы модели, основанной на платформе CloudSim, подразумевает доработку необходимых базовых компонентов платформу и предварительное описание моделируемой системы и сценария проведения симуляции в виде исходного кода. После запуска симуляции все данные о моделируемой системе передаются в ядро CloudSim, где производится симуляция.

Стоит отметить, что платформа CloudSim не подразумевает изменений в моделируемой системе или сценарии проведения симуляции непосредственно во время функционирования модели, что накладывает некоторые ограничения на возможности платформы.

2.2. CDOSim

Платформа CDOSim [12] (Cloud Deployment Options Simulator) — это симулятор, основной задачей которого является оценка производительности облачной вычислительной системы или инфраструктуры. Платформа CDOSim предназначена для сравнения

различных вариантов развертывания облачных систем и инфраструктур, что позволяет оптимизировать использование доступных ресурсов и повысить производительность облачной вычислительной системы. Платформа CDOSim позволяет гибко изменять основные параметры развертывания облачных систем и инфраструктур:

– стратегии резервирования ресурсов системы;
– конфигурации экземпляров виртуальных машин;
– аппаратные и программные средства, используемые для разработки вычислительной системы;
– параметры сети.

2.3. TeachCloud

Платформа TeachCloud [13] — это симулятор облачных вычислительных систем, который специально разрабатывался для обучения. Платформа TeachCloud предоставляет простой графический интерфейс, используя который ученики могут изменять конфигурацию и настройки облачной вычислительной системы, проводить эксперименты с возможными конфигурациями такой системы.

2.4. iCanCloud

Платформа iCanCloud — это программный симулятор больших сетевых хранилищ данных [14]. Платформа iCanCloud позволяет максимально оптимизировать использование ресурсов конкретным приложением в конкретной вычислительной среде. Платформа имеет полноценный графический интерфейс, который позволяет полностью спроектировать и выполнить симуляцию облачного хранилища данных. Кроме того, платформа iCanCloud позволяет выполнить распараллеливание симуляции облачной вычислительной системы.

2.5. SPECI

Программа SPECI [15] (Simulation Program for Elastic Cloud Infrastructures) — это симулятор который позволяет воссоздавать и исследовать поведение и масштабирование крупных центров обработки данных. Программа SPECI направлена на работу с центрами обработки данных, которые уже спроектированы, но еще не построены. Использование программы SPECI в подобных случая позволяет выявить уязвимости и «узкие» места системы.

2.6. DCSim

Платформа DCSim (Data Center Simulator) — это симулятор, который в первую очередь предназначен для работы с облачными вычислительными IaaS-системами [16], где основной проблемой является выбор подходящей в каждом конкретном случае политики выделения и резервирования ресурсов. Платформа DCSim позволяет значительно ускорить процесс разработки и запуска IaaS-системы.

3. Подходы к планированию облачных ресурсов

Планирование ресурсов является критически важным аспектом управления ресурсами облачных вычислительных систем. Соответственно, возможность моделирования различных подходов и алгоритмов планирования ресурсов является необходимой возможностью любой системы моделирования распределенных вычислительных систем.

Планирование подразумевает совместное использование ресурсов на нескольких уровнях [17]:

– *аппаратный уровень*, т.е. совместное использование сервера между несколькими виртуальными машинами;
– *уровень виртуальных машин*, т.е. каждая виртуальная машина может использоваться одновременно несколькими приложениями;
– *прикладной уровень*, т.е. каждое приложение может выполнять несколько задач параллельно.

Основной задачей планировщика облачной вычислительной системы является увеличение пропускной способности системы, то есть увеличение количества задач, завершенных за единицу времени, и сокращение времени выполнения каждой задачи.

Существуют различные подходы к планированию ресурсов в облачных вычислительных системах. Наиболее распространенными являются:

– централизованное планирование;
– иерархическое планирование;
– распределенное планирование.

3.1. Централизованное планирование ресурсов

При централизованном подходе, планирование ресурсов осуществляется одним главным узлом, который обладает актуальной информацией о состоянии всех компонентов системы [18].

При использовании этого подхода все задачи отправляются на главный узел, где они попадают в очередь и ожидают подбора ресурсов для выполнения. Схема работы системы, основанной на данном подходе, представлена на рис. 3.

Основной проблемой этого подхода является малая масштабируемость при росте количества компонентов системы, так как главный узел является «узким местом» системы. Основным достоинством этого подходя является высокая эффективность планирования, так как главный узел обладает всей необходимой для планирования информацией. Кроме того, этот подход является относительно простым в реализации.

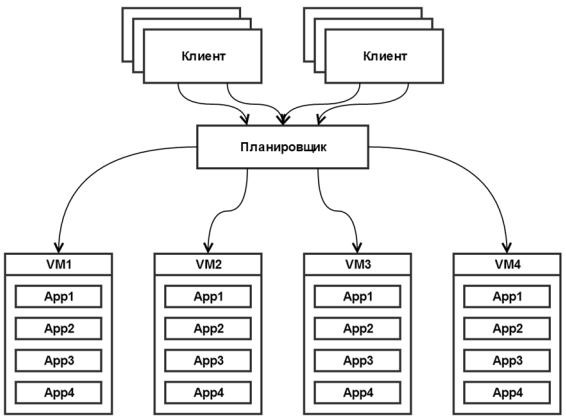


Рис. 3. Схема работы централизованного планировщика ресурсов облачной вычислительной системы

3.2. Иерархическое планирование ресурсов

Данный подход подразумевает, что кроме главного планировщика, который получает все задачи у каждого узла или группы узлов есть свой собственный планировщик (рис. 4), на который происходит перенаправление заданий с главного узла [19].

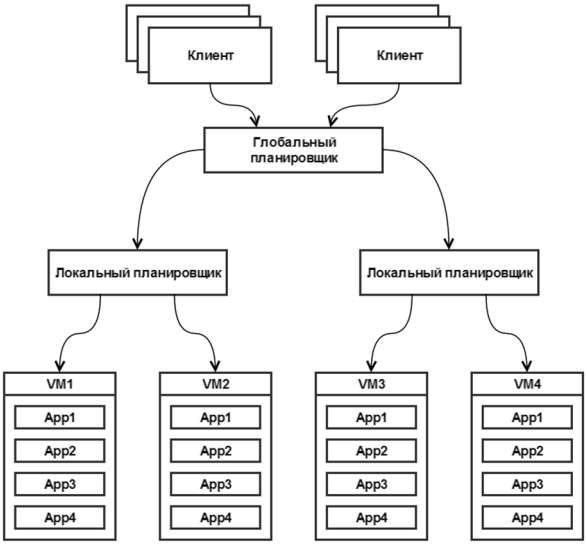


Рис. 4. Схема работы иерархического планировщика ресурсов облачной вычислительной системы

Таким образом, главный узел отвечает за глобальное планирование, а вспомогательные планировщики за локальное. Основным достоинством данного подхода является то, что он позволяет использовать разные алгоритмы для глобального и локального планирования выполнения задач.

3.3. Распределенное планирование ресурсов

При использовании данного подхода система разделяется на группы узлов, у каждой из которых есть свой собственный планировщик задач, способный принимать новые задачи из вне.

Существует два возможных способа реализации данного подхода.

Прямое взаимодействие планировщиков (рис. 5). У каждого планировщика есть список других доступных планировщиков, что позволяет каждому из них получать полную информацию о состоянии системы. Процесс планирования ресурсов для выполнения задачи строится следующим образом: если задача не может быть выполнена на доступных ресурсах, то производится подбор планировщика, которому доступны необходимые ресурсы, и задача передается ему.

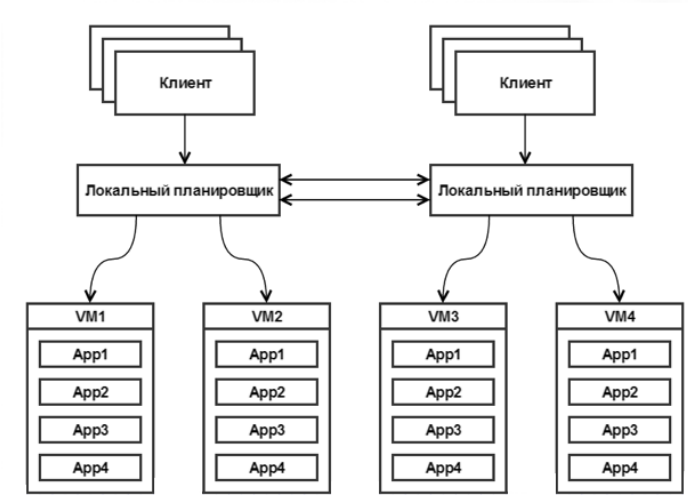


Рис. 5. Схема работы децентрализованного планировщика ресурсов облачной вычислительной системы с прямым взаимодействием

Взаимодействие планировщиков через общий пул задач (рис. 6). При использовании данного подхода планировщики не знают о состоянии друг друга. Взаимодействие происходит через общий пул задач. Процесс планирования ресурсов для выполнения задачи строится следующим образом: если задача не может быть выполнена, то она помещается в общий пул задач, откуда ее забирает для выполнения планировщик, который может ее выполнить. Данный подход может быть модифицирован так, что все задачи изначально будут поступать в общий пул задач, что в некоторых случаях может быть более выгодно.

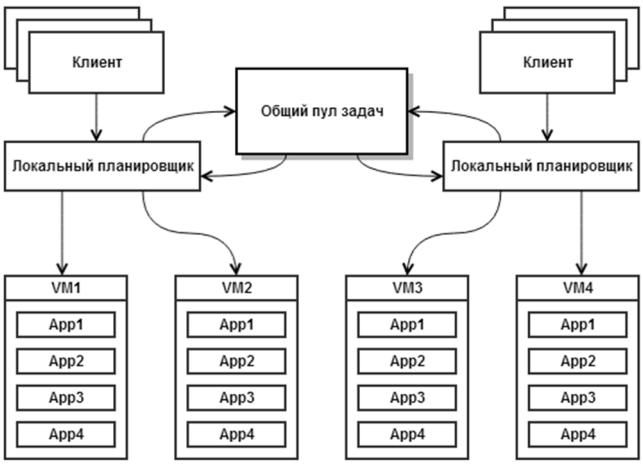


Рис. 6. Схема работы децентрализованного планировщика ресурсов облачной вычислительной системы с взаимодействием через общий пул

4. Система моделирования частных облачных PaaS-платформ

Для моделирования эффективности различных алгоритмов планирования ресурсов, нами была разработана система для моделирования и оценки производительности алгоритмов планирования частных облачных PaaS-систем. В качестве основы для создания собственной системы моделирования, нами была использована платформа

CloudSim — масштабируемая платформа симуляции, позволяющая осуществлять полноценное моделирование и симуляцию облачных вычислительных систем и инфраструктур.

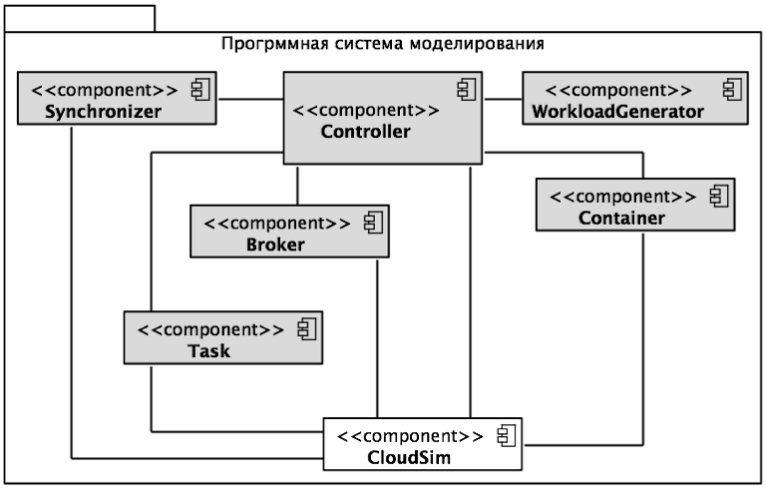


Рис. 7. Компоненты программной системы моделирования частных облачных PaaS-систем

Система моделирования частных облачных PaaS-систем состоит из шести компонентов (рис. 7):

1. *Task* — компонент, описывающий структуру и особенности выполнения задач, выполнение которых симулирует система. Данный компонент обеспечивает возможность симуляции выполнения вложенных подзадач, а также возможность указания приложения, которым должно быть выполнено данное задание.
2. *Container* — компонент, обеспечивающий функционал виртуальной машины в рамках симуляции. Обеспечивает возможность моделирования развертывания приложений, а также управления их жизненным циклом.
3. *WorkloadGenerator* — компонент, который осуществляет во время симуляции генерацию нагрузки в режиме реального времени. Данный компонент обеспечивает периодическое добавление задач в подсистему планирования моделируемой системы, что позволяет симулировать предсказуемую динамическую нагрузку на систему.
4. *Synchronizer* — компонент, обеспечивающий корректное внесение изменений в моделируемую систему во время выполнения симуляции. В ходе симуляции необходимость синхронизации возникает при симуляции спонтанной нагрузки, а также при использовании децентрализованного подхода к планированию, при котором за планирование ресурсов отвечают несколько планировщиков ресурсов
5. *Broker* — компонент, реализующий управление ресурсами моделируемой системы во время симуляции. В задачи данного компонента входит распределение виртуальных машин по имеющимся в моделируемой системе узлам, распределение приложений по виртуальным машинам, распределение задач по приложениям.
6. *Controller* — компонент, обеспечивающий подготовку сущности моделируемой системы на основе содержимого конфигурационного файла, подготовленного пользователем.

Основной для моделирования облачной системы является конфигурационный файл модели — JSON документ, содержащий подробное описание аппаратной инфраструкту-

ры моделируемой системы, а также сценарий проведения симуляции. Конфигурационный файл состоит из двух блоков:

– *datacenter* — содержит описание инфраструктуры моделируемой системы;
– *workload* — описание сценария проведения симуляции.

В блоке *datacenter* описываются физические узлы *(*блок *hosts),* виртуальные машины *(*блок *vms*), алгоритмы планирования задач и распределения приложений *(*блок *brokers).* Каждый описываемый физический узел в блоке *hosts* обладает рядом атрибутов:

– *quantity* — количество узлов с описываемыми характеристиками;
– *ram* — оперативная память узла;
– *storage* — доступная дисковая память;
– *bandwidth* — пропускная способность сети;
– *CPUs* — список имеющихся процессоров с вычислительной мощности, указанной в миллионах процессорных инструкций в секунду (mips).

Описание виртуальных машин в блоке *vms* имеет структуру, схожую с описанием физических узлов:

– *quantity* — количество виртуальных машин с описываемыми характеристиками;
– *cpu* — количество требуемых вычислительных ядер;
– *size* — размер образа;
– *ram* — требуемая оперативная память;
– *bandwidth* — пропускная способность сети;
– *mips* — требуемая вычислительная мощность в миллионах процессорных инструкций в секунду.

В блоке *brokers* указываются пути к файлам, содержащим алгоритмы планирования задач и распределения приложений в моделируемой облачной среде. Разработанная система моделирования частных облачных PaaS-систем обеспечивает возможность внедрения собственных алгоритмов управления выполнения задач и распределения приложений. Для их описания необходимо реализовать программные интерфейсы *com.model.wrapper.broker.TasksScheduler* и *com.model.wrapper.broker.AppsScheduler* (рис. 8).

```
package com.model.wrapper.broker;

import java.util.List;
import com.model.wrapper.cloudlet.Task;
import com.model.wrapper.vm.Container;
import com.model.wrapper.App;

public interface TasksScheduler {
        public void scheduleTasks(List<Task> tasks,
                        List<Container> containers);
}

public interface AppsScheduler {
        public void distributeApps(List<Task> tasks,
                List<App> apps, List<Container> containers);
        public void redistributeAppsFor(Task task,
                List<Container> containers);
}
```

Рис. 8. Интерфейсы TaskScheduler и AppsScheduler

Описание сценария проведения симуляции в блоке *workload* содержит список задач для моделируемой системы, количество повторного добавления задач в очередь и временную задержку перед повторным добавлением задач. Описание задачи имеет следующую структуру:

– *quantity* — количество задач с аналогичным описанием;
– *length* — количество инструкций процессора, необходимых для выполнения задачи;
– *fileSize* — размер файла задания в мегабайта;
– *outputSize* — размер результата выполнения задачи в мегабайтах;
– *app* — приложение, которое должно произвести выполнение задачи;
– *subtasks* — необязательный блок, описывающий список задач, которые будут добавлены в очередь после завершения симуляции выполнения задачи-владельца.

5. Испытание системы моделирования частных облачных PaaS-платформ

Для испытания системы моделирования частных облачных PaaS-платформ, нами была разработана модель частной облачной платформы Mjolnirr [20].

Тестирование производилось на основе задачи параллельной обработки текстового файла большой размера. Платформа Mjolnirr была развернута и протестирована в виртуальной распределенной вычислительной среде, состоящей из 11 узлов, один из которых был оборудован 4-ядерным процессором и 2 гигабайтами оперативной памяти и играл роль сервера, а остальные имели по одному одноядерному процессору и по 512 мегабайт оперативной памяти и играли роль пользовательских машин. Виртуальные машины были развернуты на вычислительном сервере со следующими характеристиками: 2 процессора Intel Xeon X5680 (6 ядер, 3.33 GHz) 12 GB DDR3 RAM.

Файл размером в 1 гигабайт был разбит на 100 фрагментов по 10 мегабайт. Каждый фрагмент был отправлен в очередь для обработки. Запущенные экземпляры обработчика брали отправленные фрагменты и подсчитывали количество вхождений для каждого слова в тексте. Результаты эксперимента, а также сравнение экспериментальных результатов и результатов симуляции представлены на рис. 9.

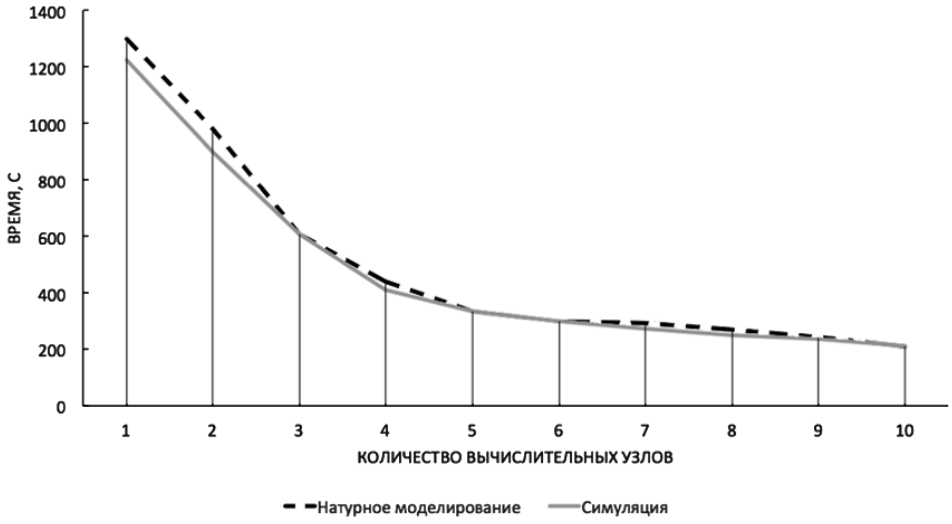


Рис. 9. Сравнение результатов симуляции и натурного моделирования

Сравнение результатов работы модели и данных вычислительного эксперимента показало, что относительная погрешность оценки времени выполнения задач платформы Mjolnirr составляет 5.2 %, что показывает, что модель корректна и пригодна для разработки алгоритмов планирования созданной платформы.

Заключение

В данной работе, нами был выполнен обзор технологий моделирования распределенных вычислительных систем. Были проанализированы возможности наиболее распространенных на сегодняшний день платформ симуляции облачных систем. Также, были рассмотрены основные подходы к планированию облачных ресурсов. На основе обзора, нами разработана архитектура и проведены испытания системы моделирования частных облачных PaaS-платформ. Сравнение результатов работы модели и данных вычислительного эксперимента показало, что модель корректна и пригодна для разработки алгоритмов планирования созданной платформы.

В качестве развития разработанной системы, планируется создание веб-приложения, обеспечивающего удобный пользовательский интерфейс для решения задач симуляции частных облачных PaaS-систем.

Работа выполнена при поддержке Российского Фонда Фундаментальных исследований (грант № 14-07-00420).

Литература

1. Gustedt, J. Experimental methodologies for large-scale systems: a survey / J. Gustedt, E. Jeannot, M. Quinson // Parallel Process. Lett. — World Scientific, 2009. — Vol. 19. — P. 399–418.
2. Bolze, R. Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed / R. Bolze, F. Cappello, E. Caron, M. Dayde, F. Desprez et al. // Int. J. High Perform. Comput. Appl. — USA: Sage Publications, 2006. — Vol. 20. — P. 481–494.
3. Chun, B. Planetlab: an overlay testbed for broad-coverage services / B. Chun, D. Culler, T. Roscoe // ACM SIGCOMM. — USA: ACM, 2003. — Vol. 33. — P. 3–12.
4. Song, H.J. The MicroGrid: a Scientific Tool for Modeling Computational Grids / H.J. Song // Proc. IEEE Supercomput. — USA: IEEE, 2000. — P. 4–10.
5. Корсуков, А.С. Инструментальные средства полунатурного моделирования распределенных вычислительных систем /А.С. Корсуков // Современные технологии. Системный анализ. Моделирование. — Россия: Иркутский государственный университет путей сообщения, 2011. — Т. 3. — С. 105–110.
6. Endo, T. Linpack evaluation on a supercomputer with heterogeneous accelerators / T. Endo // Parallel & Distrib. Process. (IPDPS), 2010 IEEE Int. Symp. — USA: IEEE, 2010. — P. 1–8.
7. Bailey, D.H. NAS parallel benchmark results / D.H. Bailey // Proc. Supercomput. '92. — USA: IEEE, 1992. — P. 1–13.
8. Buyya, R. GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing / R. Buyya, M. Murshed // Concurr. Comput. Pract. Exp. — USA: Wiley, 2002. — Vol. 14. — № 13-15. — P. 1175–1220.

9. Quinson, M. SimGrid: a generic framework for large-scale distributed experiments / M. Quinson // 2009 IEEE Ninth Int. Conf. Peer-to-Peer Comput. — USA: IEEE, 2009. — P. 126–131.
10. Calheiros, R.N. CloudSim : A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services./ R.N. Calheiros. Eprint: Australia, 2009. — 9 p.
11. Buyya, R. Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities / R. Buyya, R. Ranjan, R.N. Calheiros // 2009 Int. Conf. High Perform. Comput. Simul. — USA: IEEE, 2009. — P. 1–11.
12. Fittkau, F. CDOSim: Simulating cloud deployment options for software migration support / F. Fittkau, S. Frey, W. Hasselbring // 2012 IEEE 6th Int. Work. Maint. Evol. Serv. Cloud-Based Syst. — USA: IEEE, 2012. — P. 37–46.
13. Jararweh, Y. TeachCloud: a cloud computing educational toolkit / Y. Jararweh et al. // Int. J. Cloud Comput. 2012. — InderScience Publ., 2012. — Vol. 2. — P. 237–257.
14. Núñez, A. iCanCloud: A Flexible and Scalable Cloud Infrastructure Simulator / A. Núñez // J. Grid Comput. 2012. — Germany: Springer, 2012. — Vol. 10. — P. 185–209.
15. Sriram, I. SPECI, a Simulation Tool Exploring Cloud-Scale Data Centres / I. Sriram // Lect. Notes Comput. Sci. 2009. — Germany: Springer, 2009.– Vol. 5931. — P. 381–392.
16. Keller, G. DCSim: A data centre simulation tool / G. Keller // Integr. Netw. Manag. 2012. — USA: IEEE, 2012. — P. 1090–1091.
17. Li, W. Modeling for Dynamic Cloud Scheduling Via Migration of Virtual Machines / W. Li, J. Tordsson, E. Elmroth // 2011 IEEE Third Int. Conf. Cloud Comput. Technol. Sci. 2011. — USA: IEEE, 2011. — P. 163–171.
18. Maguluri, S.T. Stochastic models of load balancing and scheduling in cloud computing clusters / S.T. Maguluri, R. Srikant, L. Ying // INFOCOM, 2012 Proc. IEEE. — USA: IEEE, 2012. — P. 702–710.
19. Nidhi, K. Cloud Load Balancing Techniques : A Step Towards Green Computing / K. Nidhi, I. Chana // IJCSI Int. J. Comput. Sci. Issues. 2012. — USA: Eprint, 2012. — P. 238–246.
20. Savchenko, D. Mjolnirr: private PaaS as distributed computing evolution / D. Savchenko, G. Radchenko // MIPRO 2014. Proceedings of the 37th International Convention, 2014. — USA: IEEE. — P. 386–391.

Михайлов Прохор Андреевич, магистр кафедры системного программирования, Южно-Уральский государственный университет (Челябинск, Российская Федерация), prohormihailov@gmail.com

Радченко Глеб Игоревич, к.ф.-м.н., доцент кафедры системного программирования, Южно-Уральский государственный университет (Челябинск, Российская Федерация), gleb.radchenko@susu.ru.

Поступила в редакцию 4 августа 2014 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2014, vol. 3, no. 3, pp. 109–123

MODELING AND PERFORMANCE EVALUATION OF CLOUD SYSTEMS

P.A. Mihailov, South Ural State University (Chelyabinsk, Russian Federation),
G.I. Radchenko, South Ural State University (Chelyabinsk, Russian Federation)

During usage of industrial grid and cloud systems, there are issues related to the changes in the structure and algorithms of distributed computing systems and how these changes will affect the system performance. The article describes the main approaches to the experimental study of methodologies for cloud systems. The strengths and weaknesses of approaches of natural modeling, simulation, benchmarking and simulation of cloud systems are evaluated. A brief review of systems simulation is provided. As the result of the analysis we present the design and implementation of a prototype of own system for simulation of private cloud PaaS-systems. We describe the implementation of the system, as well as the test results of the developed models on the example of Mjolnirr cloud platform.

Ключевые слова: distributed computing systems, modeling, simulation, cloud computing, cloud, Mjolnirr.

References

1. Gustedt J., Jeannot E., Quinson M. Experimental methodologies for large-scale systems: a survey // Parallel Process. Lett. World Scientific. 2009. Vol. 19. P. 399–418.
2. Bolze R., Cappello F., Caron E., Dayde M., Desprez, F. et al. Grid'5000: A Large Scale And Highly Reconfigurable Experimental Grid Testbed / R. Bolze, F. Cappello, E. Caron, M. Dayde, F. Desprez et al. // Int. J. High Perform. Comput. Appl. USA: Sage Publications. 2006. Vol. 20. P. 481–494.
3. Chun B., Culler D., Roscoe T. Planetlab: an overlay testbed for broad-coverage services // ACM SIGCOMM. USA: ACM. 2003. Vol. 33. P. 3–12.
4. Song H.J. The MicroGrid: a Scientific Tool for Modeling Computational Grids Proc. IEEE Supercomput. USA: IEEE. 2000. P. 4–10.
5. Korsukov A.S. Instrumentalnyye sredstva polunaturnogo modelirovaniya raspredelennyh vychislitelnyh system [Tools for seminatural simulation of distributed computing systems] // Sovremennyye tekhnologii. Sistemnyy analiz. Modelirovaniye. [Modern technology. System analysis. Modeling]. Russia: Irkutsk State University of Railway Transport. 2011. Vol. 3. P. 105–110.
6. Endo T. Linpack evaluation on a supercomputer with heterogeneous accelerators // Parallel & Distrib. Process. (IPDPS), 2010 IEEE Int. Symp. USA: IEEE, 2010. P. 1–8.
7. Bailey D.H. NAS parallel benchmark results // Proc. Supercomput. '92. USA: IEEE, 1992. P. 1–13.
8. Buyya R., Murshed M. GridSim: a toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing // Concurr. Comput. Pract. Exp. — USA: Wiley. 2002. Vol. 14. № 13-15. P. 1175–1220.

9. Quinson M. SimGrid: a generic framework for large-scale distributed experiments // 2009 IEEE Ninth Int. Conf. Peer-to-Peer Comput. USA: IEEE. 2009. P. 126 — 131.
10. Calheiros R.N. CloudSim : A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services. Eprint: Australia, 2009. 9 p.
11. Buyya R., Ranjan R., Calheiros R.N. Modeling and simulation of scalable Cloud computing environments and the CloudSim toolkit: Challenges and opportunities // 2009 Int. Conf. High Perform. Comput. Simul. USA: IEEE, 2009. P. 1–11.
12. Fittkau F., Frey S., Hasselbring W. CDOSim: Simulating cloud deployment options for software migration support // 2012 IEEE 6th Int. Work. Maint. Evol. Serv. Cloud-Based Syst. USA: IEEE. 2012. P. 37–46.
13. Jararweh Y. et al. TeachCloud: a cloud computing educational toolkit // Int. J. Cloud Comput. 2012. InderScience Publ. 2012. Vol. 2. P. 237–257.
14. Núñez A. iCanCloud: A Flexible and Scalable Cloud Infrastructure Simulator // J. Grid Comput. 2012. Germany: Springer. 2012. Vol. 10. P. 185–209.
15. Sriram I. SPECI, a Simulation Tool Exploring Cloud-Scale Data Centres // Lect. Notes Comput. Sci. 2009. Germany: Springer. 2009. Vol. 5931. P. 381–392.
16. Keller G. DCSim: A data centre simulation tool // Integr. Netw. Manag. 2012. USA: IEEE. 2012. P. 1090–1091.
17. Li W., Tordsson J., Elmroth E. Modeling for Dynamic Cloud Scheduling Via Migration of Virtual Machines // 2011 IEEE Third Int. Conf. Cloud Comput. Technol. Sci. 2011. USA: IEEE, 2011. P. 163–171.
18. Maguluri S.T., Srikant R., Ying L. Stochastic models of load balancing and scheduling in cloud computing clusters // INFOCOM, 2012 Proc. USA: IEEE. 2012. P. 702–710.
19. Nidhi K. Chana I. Cloud Load Balancing Techniques : A Step Towards Green Computing // IJCSI Int. J. Comput. Sci. Issues. 2012. USA: Eprint. 2012. P. 238–246.
20. Savchenko D., Radchenko G. Mjolnirr: private PaaS as distributed computing evolution // MIPRO 2014. Proceedings of the 37th International Convention, 2014. USA: IEEE. P. 386–391.

Received 4 August 2014.

Краткие сообщения

УДК 004.89

МОДЕЛИРОВАНИЕ КАРЬЕРОВ РУДНЫХ МЕСТОРОЖДЕНИЙ НА ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ГИБРИДНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ[1]

Д.В. Петров, В.М. Михелев

В статье описаны принципы моделирования предельных границ рудных месторождений на высокопроизводительных вычислительных системах с гибридной архитектурой с применением параллельного генетического алгоритма.

Ключевые слова: суперкомпьютерное моделирование, предельные границы рудных месторождений, грид-системы, параллельное программирование.

Введение

Одной из важнейших задач при проектировании открытой разработки недр является определение конечных контуров карьеров. При нахождении границ карьера необходимо учитывать пространственное распределение компонентов полезных ископаемых и принятых устойчивых или технологически допустимых углов откосов бортов карьера [6]. С вычислительной точки зрения данная задача является крайне сложной, т.к. для моделирования месторождений даже среднего размера приходится обрабатывать большие массивы данных, поэтому для сокращения времени расчетов и увеличения точности получаемого решения в данной области целесообразно применение суперкомпьютерных технологий.

Цель данной статьи — продемонстрировать основные принципы моделирования предельных границ рудных месторождений на высокопроизводительных вычислительных системах с гибридной архитектурой с применением параллельного генетического алгоритма.

1. Параллельный генетический алгоритм поиска предельных границ

Для моделирования месторождения предлагается использовать двухуровневый параллельный генетический алгоритм (см. рис. 1), который хорошо накладывается на архитектуру больших гетерогенных распределенных вычислительных систем и позволяет равномерно разнести нагрузку по вычислительной системе, максимально эффективно используя многоядерные и гибридные вычислительные узлы [2].

Первый уровень параллелизма организуется за счет применения островной модели многопопуляционного параллельного генетического алгоритма [3]. Здесь ускорение достигается за счет выделения нескольких начальных популяций, развивающихся независимо, и периодически обменивающихся наиболее хорошим генетическим материалом. Данный обмен осуществляется посредством механизма миграции особей между популяциями.

[1]Статья рекомендована к публикации программным комитетом Международной научной конференции «Параллельные вычислительные технологии — 2014».

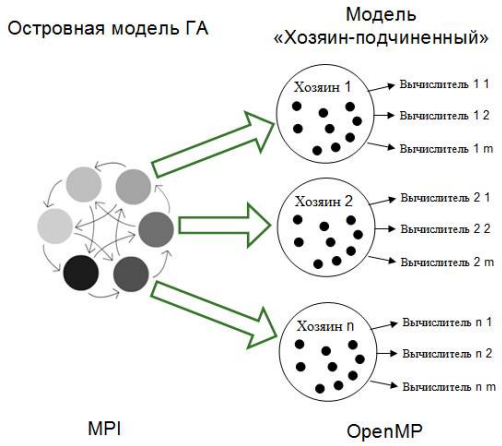


Рис. 1. Схема двухуровневого параллельного генетического алгоритма

Такой подход обеспечивает снижение вероятности преждевременного вырождения популяций, увеличению их разнообразия и ускорению схождения алгоритма поиска.

Второй уровень иерархии организуется за счет применении для каждой подпопуляции однопопуляционной модели параллельного генетического алгоритма типа «Хозяин-подчиненный». Она заключается в том, что в рамках одной популяции функция приспособленности каждого индивидуума вычисляется в отельном потоке, что в итоге приводит к ускорению работы алгоритма. При этом один поток является главным, «хранителем» популяции и отвечает за работу генетических операторов, а ряд потоков-подчиненных только вычисляют функцию приспособленности.

2. Архитектура вычислительного комплекса

В качестве технической платформы для проведения вычислительных экспериментов использовался суперкомпьютер «Нежеголь» Белгородского государственного национального исследовательского университета. Структурную схему взаимодействия основных компонентов системы можно увидеть на рис. 2.

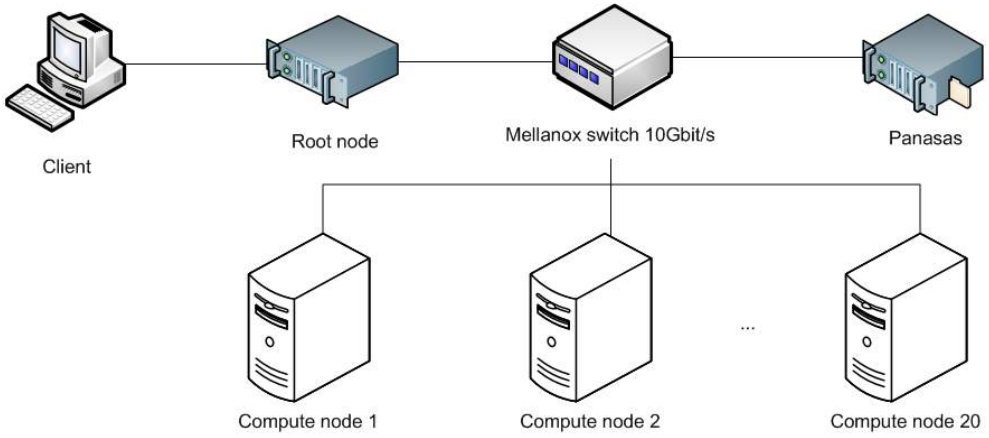


Рис. 2. Схема оборудования

Суммарные технические характеристики суперкомпьютера приведены в таблице.

Таблица

Суммарные технические характеристики кластерной системы

Характеристика	**Значение**
Семейство процессора	Intel Xeon
Частота процессора	2,4 ГГц
Количество процессоров	40
Количество ядер	320
Объем ОЗУ	1280 Гбайт
Объем HDD	8 Тбайт
Сеть	10 Гбит/с

3. Вычислительный эксперимент

Вследствие отсутствия доступа к геологическим моделям реальных месторождений полезных ископаемых, для проверки разработанного алгоритма исходные данные генерировались квазислучайным методом. Алгоритма тестировался на нескольких моделях пространственного распределения полезных компонентов в земной поверхности: наклонное послойное залегание, вертикальное залегание, равномерное случайное распределение. На рис. 3 приведен пример визуального представления граничной формы карьера размером 100 на 100 на 100 метров с разрешением 1 метр, рассчитанного генетическим алгоритмом.

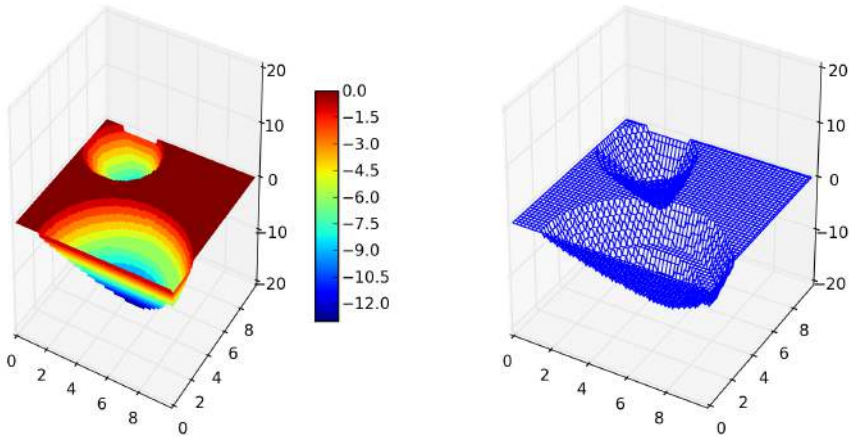


Рис. 3. Трехмерное изображение формы карьера, масштаб 1:10000

В рамках эксперимента проводилась проверка работы алгоритма на нескольких вычислительных узлах с общим количеством видеокарт равным 8. Целью данного эксперимента было выяснить, как меняется время выполнения программы в зависимости от количества используемых графических ускорителей и сделать вывод, целесообразно ли применение второго уровня параллелизма в алгоритме.

В качестве тестовых данных использовалась модель карьера со случайным пространственным распределением полезных компонентов размеров 1000 на 1000 на 100 блоков. Проведя 8 запусков программы, с постоянно увеличивающимся количеством вычислительных потоков, был получен результат, приведенный на рис. 4.

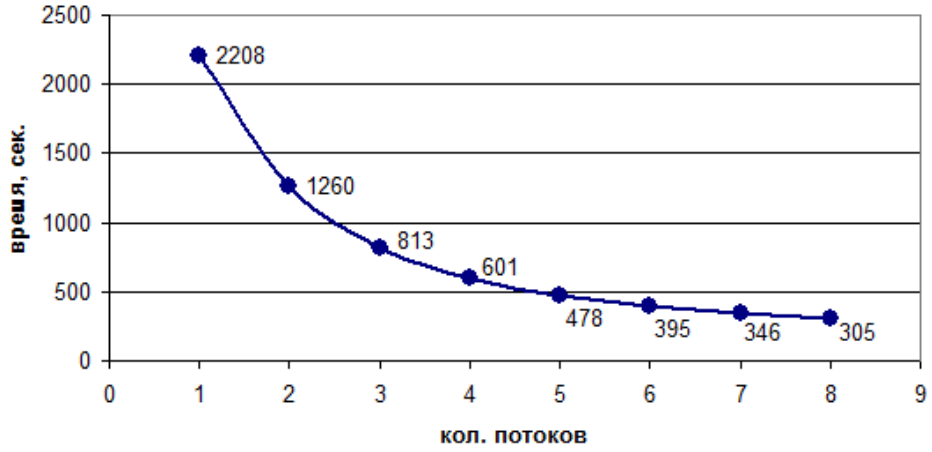


Рис. 4. Зависимость времени расчета от количества использованных видеокарт

Для оценки качества масштабируемости построенного алгоритма необходимо рассчитать ускорение в зависимости от количества вычислительных потоков по формуле

$$S = \frac{T_1}{T_n},$$

где T_1 — время выполнения алгоритма одним потоком, T_n — время выполнения на n потоках.

В результате был получен график, приведенный на рис. 5. Он показывает зависимость ускорения от количества вычислительных потоков.

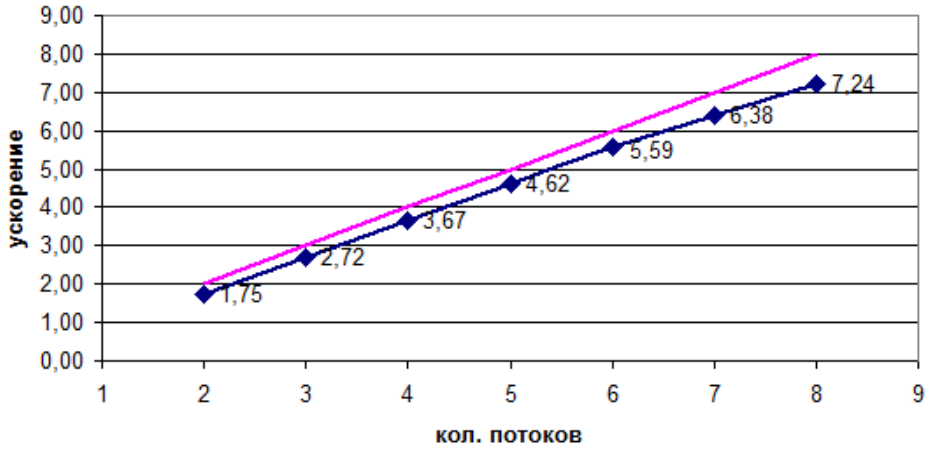


Рис. 5. Зависимость ускорения от количества вычислительных потоков

По итогам проведения эксперимента можно сделать вывод, что второй уровень параллелизма хорошо масштабируется в рамках вычислительного узла с несколькими вычислительными ядрами и его применение дает существенное преимущество по сравнению с обычным генетическим алгоритмом.

Выводы

Результаты вычислительных экспериментов показали перспективность предложенного метода для выполнения расчетов на регулярных блочных моделях месторождений твердых полезных ископаемых, разрабатываемых открытым способом. Основные преимуще-

ства предложенного метода заключаются в предоставлении нового принципа решения задачи оптимизации карьеров, позволяющего работать напрямую с трехмерной моделью месторождения, что значительно повышает адекватность получаемой модели. Кроме того, возможности гибкого масштабирования вычислительного процесса позволяют сокращать время обсчета модели почти линейно с увеличением количества вычислительных узлов.

Литература

1. Denby B., Schofield D. The Use of Genetic Algorithms in Underground Mine Scheduling., 1995
2. Васильев П.В. Ускорение моделирования и оптимизации извлечения запасов рудных месторождений на основе параллельных вычислений // Горный информационно-аналитический бюллетень. — М.: МГГУ , 2012. — №3. — С. 205-211.
3. Ramazan S., Dagdelen K., Johnson T.B. Fundamental tree algorithm in optimizing production scheduling for open pit mine design. Trans IMM (Section A: Mining Industry) vol. 114, 2005
4. Гергель В.П. Высокопроизводительные вычисления для многопроцессорных многоядерных систем : учебник для студентов вузов, обучающихся по направлениям ВПО: 010400 "Прикладная математика и информатика"и 010300 "Фундаментальная информатика и информационные технологии"/ Гергель В.П.; Б-ка Нижегородского гос. ун-та им. Н.И. Лобачевского; УМО по классическому университетскому образованию . - М.: Московский университет, 2010. - 544 с.
5. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы, Горячая Линия Телеком, 2007
6. Шпаковский Г.И. Реализация параллельных вычислений: MPI, OpenMP, кластеры, грид, многоядерные процессоры, графические процессоры, квантовые компьютеры. Минск. БГУ, 2011

Петров Денис Васильевич, Белгородский государственный университет (Белгород, Российская Федерация), petrov@bsu.edu.ru.

Михелев Владимир Михайлович, Белгородский государственный университет (Белгород, Российская Федерация)

Поступила в редакцию 6 марта 2014 г.

Bulletin of the South Ural State University
Series "Computational Mathematics and Software Engineering"
2014, vol. 3, no. 3, pp. 124–129

MODELING THE OPEN PIT LIMITS ON HIGH PERFORMANCE HYBRID COMPUTING SYSTEMS

D.V. Petrov, Belgorod National Research University (Belgorod, Russian Federation)**,**
V.M. Mikhelev, Belgorod National Research University (Belgorod, Russian Federation)

This paper describes the principles of supercomputer simulations of the searching open pit limits on high performance hybrid computing systems using parallel genetic algorithm.

Keywords: supercomputing simulation, the open pit limits, grid systems, parallel programming.

References

1. Denby B., Schofield D. The Use of Genetic Algorithms in Underground Mine Scheduling., 1995
2. Vasil'ev P.V. Uskorenie modelirovanija i optimizacii izvlechenija zapasov rudnyh mestorozhdenij na osnove parallel'nyh vychislenij [Accelerate modeling and optimization of extraction of reserves of ore deposits based on parallel computing] // Gornyj informacionno-analiticheskij bjulleten' [Mining information-analytical bulletin] — M.: MGGU , 2012. No. 3. P. 205–211.
3. Ramazan S., Dagdelen K., Johnson T.B. Fundamental tree algorithm in optimizing production scheduling for open pit mine design. Trans IMM (Section A: Mining Industry). Vol. 114. 2005
4. Gergel' V.P. Vysokoproizvoditel'nye vychislenija dlja mnogoprocessornyh mnogojadernyh sistem : uchebnik dlja studentov vuzov, obuchajushhihsja po napravlenijam VPO: 010400 "Prikladnaja matematika i informatika"i 010300 "Fundamental'naja informatika i informacionnye tehnologii"[High-performance computing for multi-core systems: a textbook for students who study HPE: 010400 "Applied Mathematics and Informatics"and 010300 "Fundamental science and information technologies"Mountain information-analytical bulletin] M.: Moskovskij universitet [Moscow University], 2010. 544 p.
5. Rutkovskaja D., Pilin'skij M., Rutkovskij L. Nejronnye seti, geneticheskie algoritmy i nechetkie sistemy [Neural networks, genetic algorithms and fuzzy systems], Gorjachaja Linija Telekom [Hotline Telecom], 2007.
6. Shpakovskij G.I. Realizacija parallel'nyh vychislenij: MPI, OpenMP, klastery, grid, mnogojadernye processory, graficheskie processory, kvantovye komp'jutery [Implementation of parallel computing: MPI, OpenMP, clusters, grid, multicore processors, graphics processors, quantum computers]. Minsk. BGU [Minsk. BSU], 2011.

Received 6 March 2014.

СВЕДЕНИЯ ОБ ИЗДАНИИ

Серия основана в 2012 году.

Свидетельство о регистрации ПИ ФС77-57377 выдано 24 марта 2014 г. Федеральной службой по надзору в сфере связи, информационных технологий и массовых коммуникаций.

Журнал включен в Реферативный журнал и Базы данных ВИНИТИ. Сведения о журнале ежегодно публикуются в международной справочной системе по периодическим и продолжающимся изданиям «Ulrich's Periodicals Directory».

ПРАВИЛА ДЛЯ АВТОРОВ

1. Правила подготовки рукописей и пример оформления статей можно загрузить с сайта серии `http://vestnikvmi.susu.ru`. **Статьи, оформленные без соблюдения правил, к рассмотрению не принимаются и назад авторам не высылаются.**
2. Адрес редакции научного журнала «Вестник ЮУрГУ», серия «Вычислительная математика и информатика»:

   Россия 454080, г. Челябинск, пр. им. В.И. Ленина, 76, Южно-Уральский государственный университет, факультет Вычислительной математики и информатики, кафедра СП, ответственному секретарю, доценту Цымблеру Михаилу Леонидовичу.
3. Адрес электронной почты редакции: `vestnikvmi@gmail.com`
4. **Плата с авторов за публикацию рукописей не взимается, и гонорары авторам не выплачиваются.**
5. Подписной индекс научного журнала «Вестник ЮУрГУ», серия «Вычислительная математика и информатика»: 10244, каталог «Пресса России». Периодичность выхода — 4 выпуска в год (февраль, май, август и ноябрь).