УДК 004.45

МОДЕЛИ И МЕТОДЫ ПРОФИЛИРОВАНИЯ И ОЦЕНКИ ВРЕМЕНИ ВЫПОЛНЕНИЯ ПОТОКОВ РАБОТ В СУПЕРКОМПЬЮТЕРНЫХ СИСТЕМАХ

Г. И. Радченко[1], **Л. Б. Соколинский**[1], **А. В. Шамакина**[1]

Решение сложных инженерно-научных задач на распределенных и суперкомпьютерных вычислительных системах часто может быть эффективно реализовано посредством организации потоков работ, объединяющих отдельные программные компоненты (генераторы сеток, решатели, визуализаторы, системы многокритериальной оптимизации и др.) для решения конечной задачи. Для оптимизации загрузки таких платформ применяются методы распознавания структуры приложений на основе модели потоков работ, а также методы формирования их профилей и оценки времени их исполнения на выделенных вычислительных ресурсах. Приводится обзор существующих методов генерации интеллектуального профилирования и оценки времени выполнения потоков работ. Предлагается новая математическая модель представления задания в виде размеченного взвешенного ориентированного ациклического графа. Рассматривается проблема кластеризации графа задания и его отображения на вычислительную среду.
Работа выполнена при финансовой поддержке Минобрнауки РФ (государственный контракт № 14.514.11.4106) в рамках ФЦП “Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007–2013 годы”.

Ключевые слова: профилирование, оценка времени выполнения, граф задания, поток работ.

1. Введение. Решение сложных инженерно-научных задач на суперкомпьютере часто может быть реализовано только посредством организации потоков работ, объединяющих отдельные программные компоненты (генераторы сеток, решатели, визуализаторы, системы многокритериальной оптимизации и др.) для решения конечной задачи.

Для оптимизации загрузки современных суперкомпьютерных систем необходимо решить задачу распознавания структуры таких приложений на основе модели потоков работ, формирования их профилей и оценки времени их исполнения на выделенных вычислительных ресурсах. Эта проблема особенно актуальна по отношению к системам, оснащенным многоядерными ускорителями, которые могут значительно увеличить скорость решения определенных типов задач.

На сегодняшний день предложено большое количество алгоритмов планирования, ориентированных на их использование в суперкомпьютерных и распределенных вычислительных средах. Некоторые из алгоритмов осуществляют планирование с учетом потоков работ в сложных приложениях. Однако разработанные алгоритмы часто не учитывают специфику области задачи и не используют дополнительные знания о ней. Кроме того, алгоритмы планирования рассматривают информацию о прикладных сервисах, но не используют знания о производственном процессе, описываемом потоком работ. В связи с этим, актуальной является задача разработки методов управления ресурсами в проблемно-ориентированных вычислительных средах, которые позволят создать эффективную и действенную систему планирования потоковых приложений.

Для достижения этой цели необходимо построить формальную модель проблемно-ориентированной среды, которая представляет поток работ в виде ориентированного ациклического графа, предусматривает использование кластеризации вершин графа и позволяет оценить время выполнения потока работ.

В настоящей статье приводится обзор существующих методов генерации интеллектуального профилирования и оценки времени выполнения потоков работ. На основе проведенного обзора предлагается новая математическая модель представления задания в виде размеченного взвешенного ориентированного ациклического графа. Рассматривается проблема кластеризации графа задания и его отображения на вычислительную среду.

[1] Южно-Уральский государственный университет, факультет вычислительной математики и информатики, просп. им. В.И. Ленина, 76, к. 477/3а, 454080, г. Челябинск; Г. И. Радченко, доцент, e-mail: gleb.radchenko@gmail.com; Л. Б. Соколинский, профессор, e-mail: sokolinsky@acm.org; А. В. Шамакина, преподаватель, e-mail: avshamakina@gmail.com

© Научно-исследовательский вычислительный центр МГУ им. М. В. Ломоносова

Статья организована следующим образом. В разделе 2 рассматриваются основные методы интеллектуального планирования потоковых приложений в распределенных и суперкомпьютерных средах. В разделе 3 приводится обзор методов оценки производительности и времени выполнения задач. В разделе 4 представлена модель вычислительного задания в виде ориентированного ациклического графа. Раздел 5 содержит заключение и возможные направления дальнейших исследований.

2. Методы интеллектуального планирования потоковых приложений. Интеллектуальное планирование и диспетчеризация (AI planning and scheduling) — это область задач искусственного интеллекта, касающаяся выполнения стратегии или последовательности действий для поиска решений поставленных задач в рамках ограниченного времени и ресурсов [15]. Результатом планирования является план (последовательность действий) по достижению поставленных целей с заданными начальными условиями и удовлетворяющих ограничениям предметной области согласно установленным правилам. Диспетчеризация — это оптимизационная задача, в которой ограниченный объем ресурсов распределяется среди частично упорядоченного множества действий таким образом, чтобы обеспечить оптимальное значение некоторой целевой функции [16]. Примером плана с учетом интеллектуального планирования является экземпляр бизнес-процесса или конкретный поток работ. Однако если при формировании бизнес-процесса обычно в план добавляется вся информация о выделенных ресурсах и временны́х ограничениях, то в области интеллектуального планирования эту задачу берет на себя диспетчер, хотя на сегодняшний день эти задачи все чаще объединяются в единый процесс [15].

В последнее время ряд работ был направлен на автоматизацию процесса интеллектуального планирования выполнения потоков работ в вычислительных системах. Так, в работе [7] предлагается методика формирования отдельных проблемно-ориентированных планировщиков для решения задач для каждой конкретной предметной области. В работе [2] представлена модель операторов, формирующих потоки работ в системе Pegasus. Оператор представляет собой запуск программного компонента на конкретном вычислительном устройстве (для генерации файла или передачи файла по сети). В операторе определяются также предварительные условия запуска компонента с учетом зависимости по данным, необходимым для его исполнения, а также требований к вычислительной системе, на которой может выполняться компонент. Представленный язык описания операторов во многом похож на язык описания виртуальных данных “Химера” (Chimera’s Virtual Data Language) [5]. Однако отличительной особенностью этого языка является возможность описания дополнительной информации о предварительных условиях, необходимых для использования компонента, и описания того, каким образом исполнение компонента оказывает влияние на состояние системы (например, объем потребляемых компонентом ресурсов).

В статье [15] представлена реализация автоматической генерации потока работ для решения задач в рамках сформированной базы знаний о предметной области и существующих ресурсах. В системе реализована основанная на действиях модель онтологии проблемной области. В рамках данной модели действия используют существующие ресурсы для удовлетворения технологических требований, определенных пользователем. Каждое действие имеет набор предварительных условий, которые должны быть истинными для выполнения действия или процесса. Правила могут быть использованы для определения этих условий, проверки того, может ли действие быть выполнено в настоящий момент, и проверки успешности выполнения действия (цели процесса).

Для построения потоков работ используется система IPSS (Integrated Planning and Scheduling System), обеспечивающая решение задач интеллектуального планирования и диспетчеризации с учетом временны́х ограничений и ограничений доступных ресурсов [14, 16].

Система IPSS представляет собой интегрированную систему планирования и диспетчеризации. Система состоит из двух решателей, каждый из которых пользуется собственным представлением знаний предметной области. Так, планировщик использует информацию о целях и доступных действиях, которые обеспечивают достижение этих целей, и устанавливает причинно-следственные связи между ними. Диспетчер же отвечает за распределение времени решения задачи и поиск необходимых ресурсов. Таким образом, суть подхода IPSS к решению задачи реализуется посредством двухшаговой интегрированной архитектуры, объединяющей компоненты планирования и диспетчеризации, которые постепенно строят решение задачи.

3. Методы оценки производительности выполнения задач. Службы контроля производительности (performance engineering) включают в себя службы оценки, мониторинга и измерения времени исполнения отдельных задач и всего потока работ и являются неотъемлемыми компонентами любой проблемно-ориентированной среды, так как надежные модели прогнозирования производительности необходимы для любого сколь-нибудь претендующего на реалистичность пакета планирования и учета ресурсов вычислительных систем [7].

Одной из существенных проблем при применении методов прогнозирования времени исполнения является значительная сложность выявления корреляции между значениями входных параметров задачи и временем исполнения приложения. В зависимости от алгоритмов, применяемых для реализации вычислений, время выполнения алгоритма может быть как детерминированной (с малым значением возможных отклонений), так и практически стохастической величиной. В качестве примера в работе [9] сравниваются два алгоритма: алгоритм разложения Холецкого для симметричных матриц и алгоритм перебора делителей для поиска простых чисел (рис. 1). В то время как скорость выполнения алгоритма Холецкого строго зависит от порядка входной матрицы (рис. 1а), время выполнения алгоритма перебора делителей слабо коррелирует со значением исходного числа, с которого начинает работу этот алгоритм (рис. 1б).

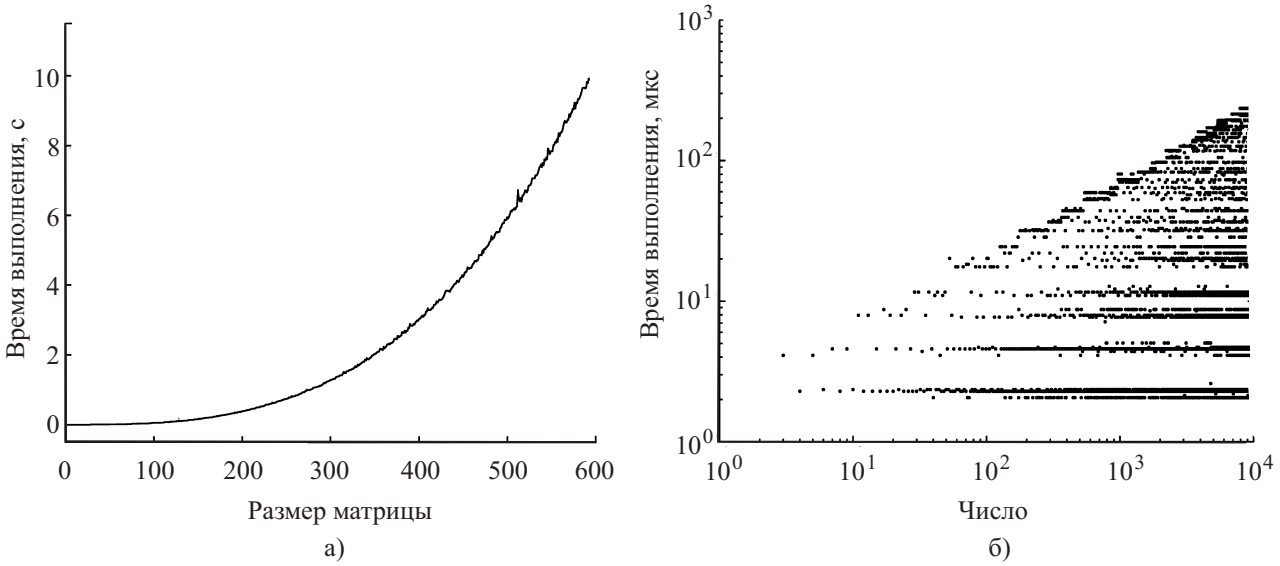

Рис. 1. Зависимость времени выполнения алгоритма от входных параметров:
а) разложение Холецкого; б) перебор делителей [9]

Выделяют три подхода к оценке времени исполнения задачи на вычислительном ресурсе [9]:
— анализ исходного кода;
— профилирование кода;
— статистические методы прогнозирования.

3.1. Методы анализа исходного кода. Пример использования анализа исходного кода представлен, например, в работе [10]. Авторами предложен подход по оценке времени работы приложения на основе аналитической модели внутренних алгоритмов работы системы и ключевых характеристик входных данных. Разработанные модели производительности параметризованы по вычислительной и коммуникационной производительности на индивидуальной системе и могут быть использованы для анализа достижимой производительности приложения на конкретной системе. В качестве примера реализации этого подхода рассматриваются два приложения: адаптивная система сгущения сеток и код Sn-транспорта на неструктурированных сетках.

К сожалению, такой подход значительно ограничен специфическими классами возможных решаемых задач и базовой архитектурой. Таким образом, данный подход не очень хорошо подходит для применения в системах с гетерогенной вычислительной архитектурой, в которых могут применяться проприетарные приложения, не обеспечивающие аналитический разбор исходных кодов.

3.2. Методы профилирования кода. Метод профилирования более применим для оценки времени исполнения приложения в гетерогенной вычислительной среде. В основе подхода лежит определение набора примитивов исходного кода. На каждой доступной машине запускаются тестовые наборы данных исходных кодов и определяется производительность системы на исследуемом типе кода. Далее, код задачи определяется как композиция блоков исходного кода определенного вида. Аналитическая модель кода и база профилирования в дальнейшем объединяются для оценки времени исполнения конкретного кода на конкретной вычислительной системе.

Профилирование исходного кода используется для прогнозирования времени работы приложения в системе GrADS (Grid Analysis and Display System). Для оценки производительности приложения на отдельном вычислительном узле разработчики собирают информацию о количестве операций с плавающей

точкой и паттерны обращений приложения к памяти. При этом основной задачей прогнозирования является не определение точного времени исполнения приложения, а предоставление планировщику информации об оценке времени исполнения для определения наиболее эффективной аппаратной конфигурации для указанного приложения [12].

В работе [17] представлена реализация концепции контрактов производительности (performance contracts). Контракт производительности выполняет оценку того, что при заданном наборе ресурсов (например, процессоров или сетей), с определенными возможностями (например, количество операций с плавающей точкой в секунду или пропускная способность) и для определенных параметров начальных условий (например, размер матрицы или разрешение изображения) приложение будет выполняться с определенной производительностью (например, обеспечит визуализацию R кадров в секунду или завершит исполнение итерации i за t секунд). Для формирования контракта производительности используется модель производительности приложения, основанная на внутренних метриках, формирующих отпечаток исполнения (execution signature) программы. Для мониторинга исполнения контрактов используется система оценки исполнения программы, основанная на механизме нечеткой логики.

В работе [18] представлена методика оценки времени исполнения приложения в грид-среде. Для оценки времени исполнения используются нормированные характеристики базовой вычислительной среды (например, так называемая "эффективная вычислительная мощность узла", определяющая количество циклов процессора, которые можно непосредственно использовать для решения поставленной задачи). Для оценки вычислительной сложности приложения используется программа TPROF, вычисляющая количество циклов процессора, использованных конкретным модулем на конкретных данных, после чего используется полиномиальная функция регрессии для нахождения промежуточных значений. Тестирование этого метода для оценки времени исполнения задач на вычислительных узлах показало в среднем 20-процентное ускорение исполнения потока задач по сравнению с методом планирования [19], который учитывал исключительно аппаратные характеристики вычислительных систем.

Однако рассматриваемый подход обладает как минимум двумя недостатками. Во-первых, отсутствует механизм, обеспечивающий достоверные результаты оценки времени работы алгоритма на основе профилирования и тестирования кода на большом количестве алгоритмов и архитектур. Во-вторых, этот подход очень тяжело связать с вариативностью входных данных, хотя он и является хорошим вариантом оценки производительности работы определенных алгоритмов на базе определенных вычислительных систем.

3.3. Статистические методы прогнозирования времени выполнения. Статистические методы прогнозирования используют данные предыдущих запусков задач. Набор информации о прошедших запусках, который хранится для каждой вычислительной системы, используется для прогнозирования времени работы новых задач. Алгоритм планирования использует данные прогнозирования (и иную информацию) для выбора машины, на которую следует передать исполнение той или иной задачи. В то время как задача исполняется на выбранной системе, производятся замеры времени исполнения, и эти замеры последовательно добавляются к набору предыдущих запусков. Естественно, чем больше хранится замеров предыдущих запусков задачи, тем более точные результаты оценки может дать статистический алгоритм. Статистические методы прогнозирования представлены, например, в работах [4, 8].

Преимуществами использования статистических методов является возможность принимать во внимание входные данные задачи, а также отсутствие необходимости непосредственного знания о внутренней реализации анализируемой программной системы. В работе [9] представлен вывод, анализ и тестирование статистического алгоритма оценки времени выполнения приложения на узлах распределенной вычислительной системы, а также предложен метод, обеспечивающий нормирование результатов замеров производительности решения задачи. Это позволяет расширить базу для статистического анализа времени исполнения задачи на различных вычислительных системах.

В работе [3] предложен метод, обеспечивающий оценку времени решения задачи непосредственно в процессе исполнения на основе комбинации информации о предыдущих запусках аналогичных задач и промежуточных данных, поступающих в процессе работы.

В работе [11] представлен подход применения нейронных сетей для анализа ключевых параметров файлов постановки задачи и прогнозирования параметров нагрузки, требуемой для решения задачи, с учетом таких параметров, как необходимое количество операций в секунду, количество операций ввода-вывода, объем необходимой памяти и дискового пространства, количество отправленных и полученных байт. Для оценки времени исполнения задачи рассчитывается предположительный объем затрат времени на коммуникацию (communication time), время на ожидание в очереди (queue time interval) и время на исполнение задачи (execution time). При оценке этих параметров учитываются как прогнозирование нагрузки, так и ключевые характеристики целевого вычислительного узла.

В отличие от методов, рассмотренных выше, которые ориентированы на анализ и прогнозирование времени исполнения отдельной задачи, в работе [13] представлена попытка прогнозирования выполнения потока работ в целом. Для этого применяется метод, основанный на поиске общих шаблонов в исполняемых потоках работ. Под шаблоном понимается набор ключевых атрибутов потока работ. Авторы выделяют статические и динамические характеристики потока работ. Статические характеристики включают в себя такие параметры, как зависимости действий по данным, размеры начальных данных, исполняемые файлы, версии и др. Динамические параметры отражают процесс исполнения работы и параметры вычислительной среды, такие как текущее количество задач в очереди вычислительного ресурса, объем доступной памяти на узлах, используемые планировщики и др. Авторами предложен алгоритм, обеспечивающий построение шаблона потока работ на основе указанных характеристик. На основе полученных шаблонов потоки работ разбиваются на классы, к которым применяются методы прогнозирования времени исполнения на основе исторических данных. Примененный подход показал хорошие результаты на трех примерах приложений. Нормализованная средняя абсолютная ошибка прогнозирования времени исполнения потока работ в зависимости от приложения, количества узлов и начальных данных варьировалась от 10% до 63% в лучшем и худшем случаях соответственно.

Приведенный выше анализ методов интеллектуального планирования потоковых приложений и методов оценки производительности выполнения показал, что на сегодняшний день предложено большое количество алгоритмов планирования, ориентированных на их использование в гетерогенных распределенных вычислительных средах. Некоторые из алгоритмов выполняют планирование с учетом потоков работ в сложных приложениях. Однако разработанные алгоритмы часто не учитывают специфику предметной области задачи и не используют дополнительные знания о ней. Кроме того, алгоритмы планирования рассматривают информацию о прикладных сервисах, но не используют знания о производственном процессе, описываемом потоком работ. В связи с этим актуальной является задача разработки методов управления ресурсами в проблемно-ориентированных вычислительных средах, которые позволят создать эффективную и действенную систему планирования потоковых приложений.

Для достижения этой цели нами построена модель проблемно-ориентированной среды, которая представляет поток работ в виде ориентированного ациклического графа, предусматривает использование кластеризации вершин графа и позволяет оценить время выполнения потока работ.

4. Модель вычислительного задания в виде ориентированного ациклического графа.

4.1. Основные определения. *Ориентированным графом* называется четверка $G = \langle V, E, \mathrm{init}, \mathrm{fin}\rangle$, где V — множество вершин; E — множество дуг; $\mathrm{init} : E \to V$ — функция, определяющая *начальную вершину* дуги; $\mathrm{fin} : E \to V$ — функция, определяющая *конечную вершину* дуги.

Вершины ν, $\nu' \in V$ являются *смежными*, если

$$\exists\, e \in E\Big(\big(\nu = \mathrm{init}(e)\,\&\,\nu' = \mathrm{fin}(e)\big)\bigvee\big(\nu = \mathrm{fin}(e)\,\&\,\nu' = \mathrm{init}(e)\big)\Big);$$

другими словами, если существует дуга e, соединяющая эти вершины. Если $\nu = \mathrm{init}(e)\,\&\,\nu' = \mathrm{fin}(e)$, то будем использовать обозначение $(\nu, \nu') = e \in E$ и говорить, что вершины ν и ν' *инцидентны* дуге e.

Пусть ν, $nu' \in V$ и $n \geqslant 1$. Упорядоченная последовательность дуг $(e_1, e_2, \ldots, e_n) \in E^n$ называется *путем длины* n от вершины ν к вершине ν', если $\nu = \mathrm{init}(e_1)$, $\nu' = \mathrm{fin}(e_n)$ и $\mathrm{fin}\,(e_i) = \mathrm{init}(e_{i+1})$ для всех $i \in \{1, \ldots, n-1\}$.

Если $(e_1, e_2, \ldots, e_n) \in E^n$ — путь от вершины ν к вершине ν', то *обратным путем* от вершины ν' к вершине ν называется упорядоченная последовательность дуг $(e_n, e_{n-1}, \ldots, e_1) \in E^n$.

Путь $(e_1, e_2, \ldots, e_n)$ называется *простым*, если $\mathrm{init}(e_1), \mathrm{init}(e_2), \ldots, \mathrm{init}(e_n)$ все различны между собой и если $\mathrm{fin}(e_1), \mathrm{fin}(e_2), \ldots, \mathrm{fin}(e_n)$ тоже все различны.

Циклом называется простой путь от некоторой вершины к ней самой. Ориентированный граф называется *ациклическим*, если он не содержит циклов [1].

Вершины ν, $\nu' \in V$ называются *независимыми*, если не существует прямого или обратного пути от вершины ν к вершине ν'. В противном случае вершины ν и ν' *зависимы*.

Пусть задан ориентированный граф $G = \langle V, E, \mathrm{init}, \mathrm{fin}\rangle$. *Взвешиванием графа* G будем называть функцию $\delta : E \to \mathbb{Z}_{\geqslant 0}$. *Разметкой графа* G будем называть функцию $\gamma : V \to \mathbb{N}^2$.

4.2. Модель вычислительной среды. *Графом задания* называется размеченный взвешенный ориентированный ациклический граф $G = \langle V, E, \mathrm{init}, \mathrm{fin}, \delta, \gamma\rangle$, где V — множество вершин, соответствующих задачам, и E — множество дуг, соответствующих потокам данных.

Вес $\delta\,(e)$ определяет объем данных, который необходимо передать по дуге e от задачи, ассоциированной с вершиной $\mathrm{init}(e)$, к задаче, ассоциированной с вершиной $\mathrm{fin}(e)$.

Метка $\gamma\,(\nu) = (m_\nu, t_\nu)$ определяет количество процессорных ядер m_ν, на которых задача ν имеет

линейную масштабируемость, и время t_ν выполнения задачи ν на одном ядре.

Линейная масштабируемость предполагает, что *вычислительная стоимость* $\chi(\nu, j)$ задачи ν на j процессорных ядрах определяется следующей формулой:

$$\chi(\nu, j) = \begin{cases} t_\nu / j, & \text{если } 1 \leqslant j \leqslant m_\nu; \\ t_\nu / m_\nu, & \text{если } m_\nu < j. \end{cases} \tag{1}$$

Вычислительной системой называется множество вычислительных узлов P. Функция $\eta : P \to \mathbb{Z}_{>0}$ вычисляет *количество процессорных ядер* для каждого узла $p \in P$. В реальности вычислительная система может являться распределенной вычислительной средой, объединяющей несколько вычислительных кластеров, каждый из которых является отдельным узлом этой системы.

Кластеризацией называется однозначное отображение $\omega : V \to P$ множества вершин V графа задания G на множество вычислительных узлов P.

Пусть задана вычислительная система $P = \{p_0, \ldots, p_{k-1}\}$, состоящая из k узлов. *Кластер* W_i — это подмножество всех вершин, отображаемых на вычислительный узел $p_i \in P$: $W_i = \{\nu \in V \,|\, \omega(\nu) = p_i \in P\}$.

Имеем $W_i \bigcap W_j = \emptyset$ для $i \neq j$; $V = \bigcup\limits_{i=0}^{k-1} W_i$.

Кластеризация называется *нелинейной*, если существуют две независимые вершины, которые отображаются на один вычислительный узел. В противном случае — кластеризация называется *линейной* [6].

На рис. 2 приведены примеры: а) граф задания с указанием коммуникационных стоимостей дуг и вычислительных стоимостей вершин; б) линейная кластеризация с тремя кластерами $\{v_1, v_2, v_7\}$, $\{v_3, v_4, v_6\}$, $\{v_5\}$; в) нелинейная кластеризация с кластерами $\{v_1, v_2\}$, $\{v_3, v_4, v_5, v_6\}$ и $\{v_7\}$.

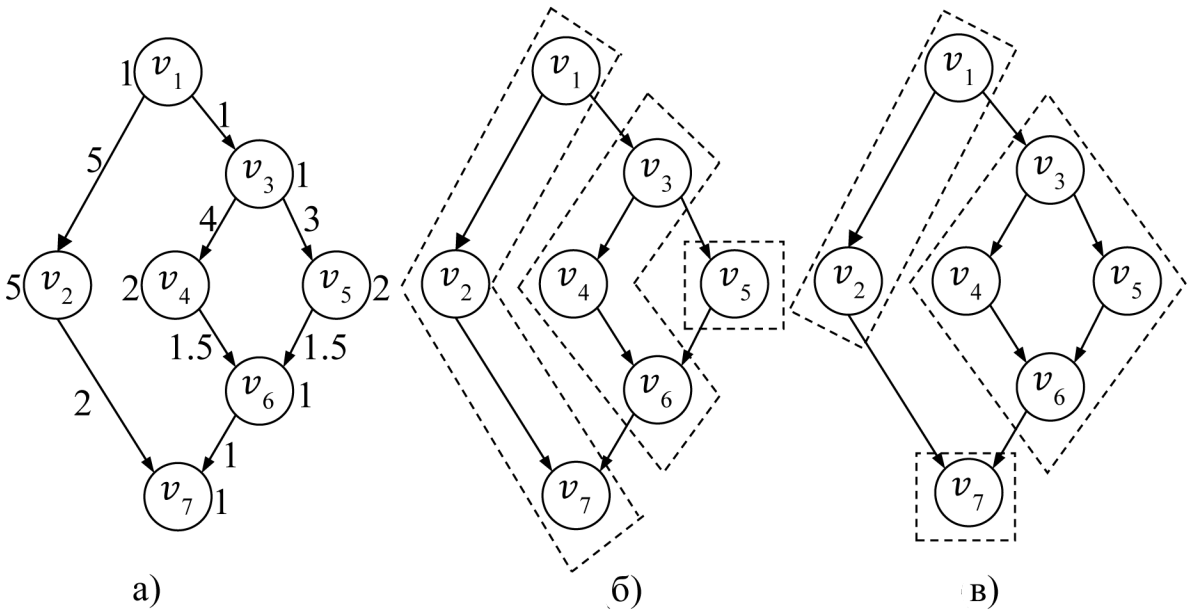

Рис. 2. Граф задания (а), линейная кластеризация (б), нелинейная кластеризация (в)

Пусть задан граф задания $G = \langle V, E, \text{init}, \text{fin}, \delta, \gamma \rangle$, для которого определена функция кластеризации ω. Будем называть такой граф *кластеризованным* и обозначать через $G = \langle V, E, \text{init}, \text{fin}, \delta, \gamma, \omega \rangle$.

Мы предполагаем, что время передачи любого объема данных между узлами, принадлежащими одному кластеру, равно нулю, а время передачи данных между узлами, принадлежащими разным кластерам, пропорционально объему передаваемых данных с коэффициентом 1. В соответствии с этим мы можем определить функцию $\sigma : E \to \mathbb{Z}_{\geqslant 0}$, вычисляющую *коммуникационную стоимость* (время) передачи данных по дуге $e \in E$, следующим образом:

$$\sigma(e) = \begin{cases} 0, & \text{если } \omega\big(\text{init}(e)\big) = \omega\big(\text{fin}(e)\big); \\ \delta(e), & \text{если } \omega\big(\text{init}(e)\big) \neq \omega\big(\text{fin}(e)\big). \end{cases}$$

Пусть задан кластеризованный граф $G = \langle V, E, \text{init}, \text{fin}, \delta, \gamma, \omega \rangle$. *Расписанием* G называется отображение $\xi : V \to \mathbb{Z}_{\geqslant 0} \times \mathbb{N}$, которое произвольной вершине $v \in V$ сопоставляет двойку чисел $\xi(v) = (\tau_v, j_v)$, где

τ_v определяет время запуска задачи и j_v — количество процессорных ядер, выделяемых задаче, ассоциированной с этой вершиной. Обозначим через s_v время останова задачи v. Имеем $s_v = \tau_v + \chi(v, j_v)$, где χ — функция временно́й сложности, определенная в (1). Расписание называется *корректным*, если оно удовлетворяет следующим условиям:

$$\forall\, e \in E\Big(\tau_{\mathrm{fin}(e)} \geqslant \tau_{\mathrm{init}(e)} + \chi\big(\mathrm{init}\,(e)\,, j_{\mathrm{init}(e)}\big) + \sigma\,(e)\Big); \tag{2}$$

$$\forall\, t \in \mathbb{N}\left(\forall\, i \in [0, \ldots, k-1]\left(\sum_{v \in W_i \;\&\; \tau_v \leqslant t \leqslant s_v} j_v \leqslant \eta\,(p_i)\right)\right). \tag{3}$$

Условие (2) означает, что для любых двух смежных вершин $\nu = \mathrm{init}\,(e)$ и $\nu' = \mathrm{fin}\,(e)$ время запуска ν' не может быть меньше суммы следующих величин: времени запуска ν, времени выполнения ν и коммуникационной стоимости дуги e.

Условие (3) означает, что в любой момент времени t количество процессорных ядер, выделяемых задачам на узле с номером i, не может превосходить общего количества ядер на этом узле. В дальнейшем мы будем считать любое расписание корректным, если явно не оговорено противное.

Кластеризованный граф с заданным расписанием будем называть *распланированным* и обозначать через $G = \langle V, E, \mathrm{init}, \mathrm{fin}, \delta, \gamma, \omega, \xi\rangle$.

Разработанная формальная модель проблемно-ориентированной среды является универсальной и позволяет описывать поток работ в виде ориентированного ациклического графа, предусматривает использование кластеризации вершин графа и позволяет оценивать время выполнения потока работ. Предложенная модель позволит обеспечить разработку эффективного алгоритма планирования ресурсов, удовлетворяющего следующим основным требованиям:

1) учет общего представления о потоке работ и его предметной области;
2) возможность предварительного резервирования вычислительных ресурсов;
3) возможность предварительной оценки времени выполнения задач;
4) возможность учета программных, аппаратных и лицензионных требований задач.

5. Заключение. В настоящей статье представлен обзор современных моделей и методов профилирования и оценки времени выполнения потоков работ в суперкомпьютерных системах. Рассмотрены подходы, основанные на методах анализа исходного кода и методах профилирования кода, а также статистические методы прогнозирования времени выполнения. Кроме того, рассмотрены методы интеллектуальной генерации и управления потоками работ в суперкомпьютерных системах и распределенных вычислительных системах. На основе проведенного анализа предложена новая математическая модель представления задания в виде размеченного взвешенного ориентированного ациклического графа. Рассмотрена проблема кластеризации графа задания и его отображения на вычислительную среду.

Дальнейшие исследования будут направлены на разработку эффективных алгоритмов планирования ресурсов на базе разработанной модели проблемно-ориентированной среды и на реализацию этих алгоритмов в виде программной системы, обеспечивающей анализ и планирование потоковых задач в суперкомпьютерных системах.

СПИСОК ЛИТЕРАТУРЫ

1. *Кнут Д.Э.* Искусство программирования. Т. 1. Основные алгоритмы. М.: Издательский дом “Вильямс”, 2000.
2. *Blythe J. et al.* The Role of Planning in Grid Computing // Proc. 13th International Conference on Automated Planning & Scheduling. Menlo Park: AAAI Press, 2003. 154–163.
3. *Chtepen M. et al.* Online execution time prediction for computationally intensive applications with periodic progress updates // J. of Supercomputing. 2012. **62**, N 2. 768–786.
4. *Devarakonda M.V., Iyer R.K.* Predictability of process resource usage: a measurement-based study of UNIX // IEEE Trans. on Software Engineering. 1989. **15**. 1579–1586.
5. *Foster I. et al.* Chimera: a virtual data system for representing, querying, and automating data derivation // Proc. 14th International Conference on Scientific and Statistical Database Management. New York: IEEE Press, 2002. 37–46.
6. *Gerasoulis A., Yang T.* A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors // J. of Parallel and Distributed Computing. 1992. **16**, N 4. 276–291.
7. *Hey T., Papay J.* Performance engineering, PSEs and the GRID // Scientific Programming. 2002. **10**, N 1. 3–17.
8. *Ilghami O., Nau D.S.* A general approach to synthesize problem-specific planners. Technical Report CS-TR-4597. College Park: Univ. of Maryland, 2004.

9. *Iverson M.A., Ozguner F., Potter L.C.* Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment // Proc. Eighth Heterogeneous Computing Workshop (HCW'99). New York: IEEE Press, 1999. 99–111.
10. *Kerbyson D.J., Wasserman H.J., Hoisie A.* Exploring advanced architectures using performance prediction // International Workshop on Innovative Architecture for Future Generation High Performance Processors and Systems. New York: IEEE Press, 2002. 27–37.
11. *Litke A. et al.* Computational workload prediction for grid oriented industrial applications: the case of 3D-image rendering // CCGrid 2005. IEEE International Symposium on Cluster Computing and the Grid. New York: IEEE Press, 2005. 962–969.
12. *Mandal A. et al.* Scheduling strategies for mapping application workflows onto the grid // HPDC-14. Proc. 14th IEEE International Symposium on High Performance Distributed Computing. New York: IEEE Press, 2005. 125–134.
13. *Nadeem F., Fahringer T.* Using templates to predict execution time of scientific workflow applications in the grid // Proc 9th IEEE/ACM International Symposium on Cluster Computing and the Grid. New York: IEEE Press, 2009. 316–323.
14. *Rodriguez-Moreno M.D. et al.* Integrating planning and scheduling in workflow domains // Expert Systems with Applications. 2007. **33**, N 2. 389–406.
15. *Rodriguez-Moreno M.D. et al.* IPSS: a hybrid approach to planning and scheduling integration // IEEE Trans. on Knowledge and Data Engineering. 2006. **18**, N 12. 1681–1695.
16. *Topcuoglu H., Hariri S., Society I.C.* Performance-effective and low-complexity // IEEE Trans. on Parallel and Distributed Systems. 2002. **13**, N 3. 260–274.
17. *Vraalsen F. et al.* Performance contracts: predicting and monitoring grid application behavior // Lecture Notes in Computer Science. Vol. 2242. Berlin: Springer, 2001. 154–165.
18. *Wu Q., Datla V.V.* On performance modeling and prediction in support of scientific workflow optimization // Proc. IEEE World Congress on Services. New York: IEEE Press, 2011. 161–168.
19. *Zhang J. et al.* Task mapper and application-aware virtual machine scheduler oriented for parallel computing // J. of Zhejiang University SCIENCE C. 2012. **13**, N 3. 155–177.

Поступила в редакцию
17.10.2013