ПРОГРАММИРОВАНИЕ, 2016, № 1, с. 26–38

ОПЕРАЦИОННЫЕ СИСТЕМЫ

УДК 004.75

МЕТОДЫ УПРАВЛЕНИЯ РЕСУРСАМИ В ПРОБЛЕМНО-ОРИЕНТИРОВАННЫХ ВЫЧИСЛИТЕЛЬНЫХ СРЕДАХ *

© 2016 г. Л.Б. Соколинский, А.В. Шамакина
Южно-Уральский государственный университет
454080 Челябинск, просп. им. В.И. Ленина, 76
E-mail: Leonid.Sokolinsky@susu.ru, shamakinaav@susu.ru
Поступила в редакцию 22.04.2015

Одним из важных классов ресурсоемких приложений в распределенных вычислительных средах являются комплексные задания с потоковой структурой, возникающие в определенной проблемной области. Подобное проблемно-ориентированное задание может быть представлено в виде ориентированного графа, узлами которого являются вычислительные задачи, составляющие задание, а дуги соответствуют потокам данных, передаваемых от одной задачи к другой. Проблемная ориентированность задания выражается в том, что имеется возможность априорного получения оценок времени выполнения задач и объемов передаваемых данных. Распределенная вычислительная среда, ориентированная на выполнение подобных заданий в определенной предметной области, называется проблемно-ориентированной. Для эффективного использования ресурсов распределенной вычислительной среды применяются специальные алгоритмы планирования. В настоящее время известно большое число таких алгоритмов. Некоторые из них (например, алгоритм DSC) учитывают специфику проблемно-ориентированных приложений с потоковой структурой. Другие (например, алгоритм Min-min) учитывают многоядерную структуру узла вычислительной системы. Однако, ни один из известных алгоритмов не учитывает оба фактора. В данной работе строится математическая модель проблемно-ориентированной вычислительной среды и предлагается новый проблемно-ориентированный алгоритм планирования ресурсов POS (Problem-Oriented Scheduling), учитывающий как проблемно-ориентированную специфику задания, так и многоядерную структуру узлов вычислительной системы. Приводятся результаты вычислительных экспериментов, сравнивающие алгоритм POS с другими известными алгоритмами планирования.

1. 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]–[4], и затем – к новой концепции облачных вычислений (cloud computing) [5]–[6]. Для раскрытия всех потенциальных возможностей использования распределенных вычислительных ресурсов принципиально важно наличие результативных и эффективных алгоритмов планирования, используемых менеджерами ресурсов.

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

*Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта 15-29-07959.

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 [7]–[10] 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 [11]. 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. Вычислительные задания в подобных предметных областях во многих случаях имеют потоковую структуру и могут быть описаны с помощью модели потока работ (workflow) [12], 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. Известны два основных класса алгоритмов, ориентированных на планирование приложений с потоковой структурой [13]: алгоритмы кластеризации и списочные алгоритмы. К алгоритмам кластеризации, например, относятся алгоритм Кима-Брауна [14] и алгоритм DSC [15]. Алгоритмы данного класса используют знания о проблемно-ориентированной специфике задач, составляющих вычислительное задание, однако они допускают выполнение задачи только на одном процессорном ядре многопроцессорной системы. Одним из наиболее популярных списочных алгоритмов является алгоритм Min-min [16]. Главным недостатком списочных алгоритмов является то, что они не анализируют граф задания в целом. В соответствие с этим актуальной является задача разработки методов и алгоритмов управления ресурсами в проблемно-ориентированных распределенных вычислительных средах, в полной мере учитывающих специфику предметной области, масштабируемость отдельных задач в задании и использующих возможность выполнения одной задачи на нескольких процессорных ядрах.

Статья организована следующим образом. В разделе 2 дается обзор нескольких известных алгоритмов планирования ресурсов в проблемно-ориентированных вычислительных средах. В разделе 3 строится математическая модель проблемно-ориентированной распределенной вычислительной среды. В разделе 4 описывается новый алгоритм планирования ресурсов POS. В разделе 5 приводятся результаты вычислительных экспериментов по исследованию алгоритма POS и его сравнению с известными алгоритмами.

2. ПЛАНИРОВАНИЕ В ПРОБЛЕМНО-ОРИЕНТИРОВАННЫХ СРЕДАХ

В данном разделе приводится обзор нескольких известных алгоритмов планирования выполнения заданий в проблемно-ориентированных вычислительных средах. В таких средах вы-

числительное задание представляется в виде ориентированного ациклического графа, пример которого изображен на рис. 1. Узлами графа являются взаимосвязанные вычислительные задачи, а дуги соответствуют потокам данных, передаваемых между отдельными задачами. При этом множество классов задач, из которых строятся задания, является конечным и предопределенным. Для каждой задачи указывается время выполнения задачи на одном процессорном ядре. Вес дуги определяет объем передаваемых данных.

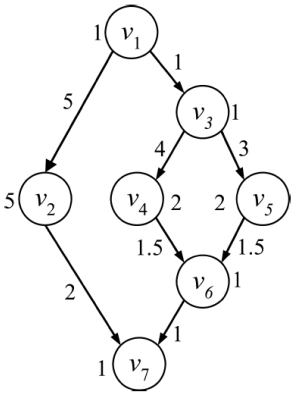

Рис. 1.
Ориентированный ациклический граф.

Одним из наиболее известных алгоритмов планирования для проблемно-ориентированных вычислительных сред является алгоритм кластеризации доминирующей последовательности DSC (Dominant Sequence Clustering) [17]. Основная идея алгоритма DSC состоит в разбиении множества вершин графа на непересекающиеся подмножества-кластеры. На рис. 2а приведен пример разбиения графа задания на три кластера, обозначенных пунктирными линиями. Задачи, попадающие в один кластер последовательно выполняются на одном и том же процессорном ядре в определенном порядке, определяемым алгоритмом. Для кластеризованного графа строится план выполнения задания, в котором для каждой задачи определяется номер процессорного ядра, на котором она будет выполняться, и время ее старта. Пример такого плана изображен на рис. 2б в виде диаграммы Ганта. Здесь задачи v_1 и v_2 выполняются на процессорном ядре p_0, v_7 – на p_1, а v_3-v_6 – на p_2.

В кластеризованном графе коммуникационные стоимости дуг, соединяющих узлы одного кластера, обнуляются. В результате этого получается распланированный граф. Пример такого графа приведен на рис. 2в. Алгоритм DSC учитывает порядок выполнения задач в кластерах, создавая в распланированном графе дополнительные псевдодуги. На рис. 2в такой дополнительной псевдодугой является дуга $e = (v_4, v_5)$. Для распланированного графа алгоритм DSC строит доминирующую последовательность, которая, по существу, является критическим путем с учетом псевдодуг. На рис. 2в доминирующая последовательность обозначена жирными стрелками.

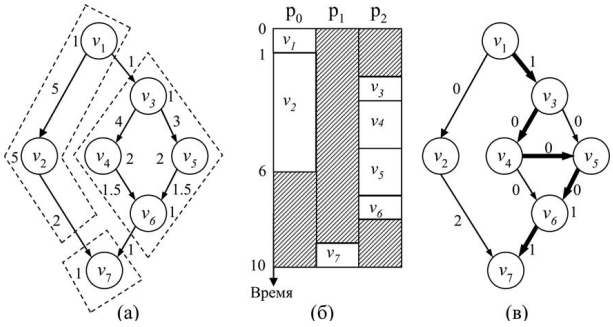

Рис. 2.
(а) Кластеризованный граф и его критический путь; (б) диаграмма Ганта; (в) распланированный граф и его доминирующая последовательность.

В начальный момент времени работы алгоритма DSC каждая вершина помещается в отдельный кластер, и 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 SN, не распланированные вершины составляют множество USN. 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. 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. Главным недостатком алгоритма DSC является ограниченность его применения для вычислительных систем с многоядерными процессорами, так как DSC планирует выполнение каждой задачи только на одном процессорном ядре.

Другим известным алгоритмом планирования для проблемно-ориентированных сред является алгоритм линейной кластеризации Кима и Брауна, также называемый алгоритмом KB/L [14]. Алгоритм KB/L предназначен для кластеризации графа задания и предполагает, что количество вычислительных узлов неограниченно. Первоначально все дуги графа задания маркируются как "нерассмотренные". 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 [14] использует стоимостную функцию

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

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. Целевой функцией алгоритма KB/L является минимизация параллельного времени графа. Алгоритм KB/L также не учитывает многоядерность вычислительных узлов.

Еще одним известным алгоритмом кластеризации является алгоритм Саркара [18]. 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. Алгоритм Саркара минимизирует параллельное время выполнения графа хуже, чем алгоритм DSC. При этом он также не учитывает многоядерность вычислительных узлов.

Примером популярного списочного алгоритма является Min-Min [16], который итеративно выполняет следующие операции. На каждом шаге для каждой задачи рассчитываются самое раннее время выполнения (ECT – Early Completion Time) для всех доступных ресурсов и минимальное оценочное время выполнения (MCT – Minimum Estimated Completion Time). Задача, имеющая наименьшую метрику MCT, первой получает все необходимые ей ресурсы. Задаче выделяются те ресурсы, на которых она завершит свое выполнение раньше. Процесс планирования завершается, когда все задачи распланированы. Данный алгоритм обычно используется для планирования заданий, которые состоят из множества независимых задач с большими модулями и интенсивными вычислениями. Основным недостатком алгоритма Min-Min является то, что он не анализирует граф задания в целом.

3. МОДЕЛЬ ВЫЧИСЛИТЕЛЬНОЙ СРЕДЫ

В данном разделе строится математическая модель проблемно-ориентированной распределенной вычислительной среды.

Графом задания будем называть размеченный взвешенный ориентированный ациклический граф $G = \langle V, E, init, fin, \delta, \gamma \rangle$, где V – множество вершин, соответствующих задачам; E – множество дуг, соответствующих потокам данных; $init : E \to V$ – функция, определяющая *начальную вершину* дуги; $fin : E \to V$ – функция, определяющая *конечную вершину* дуги. Вес $\delta(e)$ дуги e определяет объем данных,

который необходимо передать по дуге e от задачи, ассоциированной с вершиной $init(e)$ к задаче, ассоциированной с вершиной $fin(e)$. Метка

$$\gamma(\nu) = (m_\nu, t_\nu) \tag{2}$$

определяет максимальное количество процессорных ядер m_ν, на которых задача ν имеет ускорение, близкое к *линейному*, и время t_ν выполнения задачи ν на одном ядре. Данная модель предполагает, что *вычислительная стоимость* $\chi(\nu, j_\nu)$ задачи ν на j_ν процессорных ядрах определяется следующей формулой:

$$\chi(\nu, j_\nu) = \begin{cases} t_\nu / j_\nu, & \text{если } 1 \le j_\nu \le m_\nu; \\ t_\nu / m_\nu, & \text{если } m_\nu < j_\nu; \end{cases} \tag{3}$$

Другими словами, при наращивании количества процессорных ядер в диапазоне от 1 до m_ν, мы будем получать прямо пропорциональное уменьшение времени счета; при увеличении количества ядер в интервале от m_ν до $+\infty$ ускорение будет отсутствовать.

На рис. 3 приведен пример графа задания, содержащего 8 вершин. Для каждой из вершин указана метка в виде пары (m_ν, t_ν). Каждой дуге графа сопоставляется вес - объем данных $\delta(e)$, передаваемых по данной дуге.

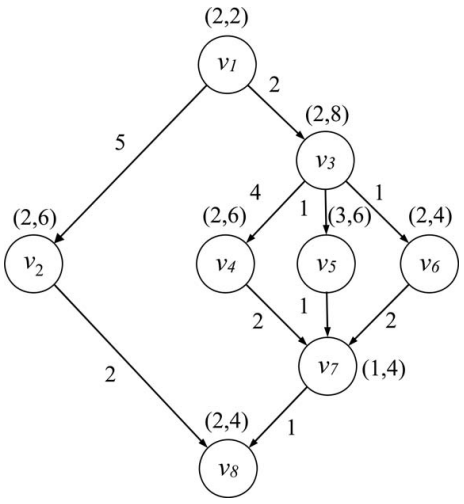

Рис. 3.
Граф задания

Вычислительным узлом P называется упорядоченное множество процессорных ядер $\{c_0, \ldots, c_{d-1}\}$.

Вычислительной системой называется упорядоченное множество вычислительных узлов $\mathfrak{P} = \{P_0, \ldots, P_{k-1}\}$. В реальности вычислительная система может являться распределенной вычислительной системой, объединяющей несколько вычислительных кластеров, каждый из которых является отдельным узлом этой системы.

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

Пусть задана вычислительная система $\mathfrak{P} = \{P_0, \ldots, P_{k-1}\}$, состоящая из k узлов. *Кластер* W_i – это подмножество всех вершин, отображаемых на вычислительный узел $P_i \in \mathfrak{P}$:

$$W_i = \{\nu \in V | \omega(\nu) = P_i \in \mathfrak{P}\}. \tag{4}$$

Имеем

$$W_i \bigcap W_j = \emptyset \text{ для } i \ne j; \tag{5}$$

$$V = \bigcup_{i=0}^{k-1} W_i. \tag{6}$$

Пусть задан граф задания $G = \langle V, E, init, fin, \delta, \gamma,$ для которого определена функция кластеризации $\omega(\nu)$. Будем называть такой граф *кластеризованным* и обозначать как $G = \langle V, E, init, fin, \delta, \gamma, \omega \rangle$.

В рамках модели 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 эти мы можем определить функцию $\sigma : E \to \mathbb{Z}_{\ge 0}$, вычисляющую *коммуникационную стоимость* (время) передачи данных по дуге $e \in E$, следующим образом:

$$\sigma(e) = \begin{cases} 0, & \text{при } \omega(init(e)) = \omega(fin(e)); \\ \delta(e), & \text{в противном случае.} \end{cases} \tag{7}$$

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

$$\xi(v) = (\tau_v, j_v), \tag{8}$$

где τ_v определяет время запуска задачи ν, j_v — количество процессорных ядер, выделяемых задаче ν, ассоциированной с этой вершиной. Обозначим через s_v — время останова задачи ν. Имеем

$$s_v = \tau_v + \chi(v, j_v), \tag{9}$$

где χ — функция временной сложности, определенная в (3). Расписание называется *корректным*, если оно удовлетворяет следующим условиям:

$$\forall\, e \in E(\tau_{fin(e)} \geq \tau_{init(e)} + \\ + \chi(init\,(e)\,, j_{init(e)}) + \sigma\,(e)); \tag{10}$$

$$\forall\, \nu \in V\,(j_\nu \leq m_\nu)\,; \tag{11}$$

$$\forall\, t \in \mathbb{N}(\forall\, i \in [0, \ldots, k-1] \\ (\sum_{\nu \in W \& \tau_\nu < t \leq s_\nu} j_\nu \leq |P_i|)). \tag{12}$$

Условие (10) означает, что для любых двух смежных вершин $\nu_1 = init\,(e)$ и $\nu_2 = fin\,(e)$ время запуска ν_2 не может быть меньше суммы следующих величин: время запуска ν_1, время выполнения ν_1, коммуникационная стоимость дуги e. Условие (11) означает, что количество ядер, выделяемое задаче ν_1, не превышает границы линейной масштабируемости, заданной разметкой γ в контексте формулы (2). Условие (12) 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 ξ будем называть *распланированным* и обозначать как $G = \langle V, E, init, fin, \delta, \gamma, \omega, \xi\rangle$.

Ярусно-параллельной формой (ЯПФ) [19] называется разбиение множества вершин V ориентированного ациклического графа $G = \langle V, E, init, fin\rangle$ на перенумерованные подмножества (ярусы) $L_i (i = 1, \ldots, r)$, удовлетворяющие следующим свойствам:

$$\left.\begin{array}{l} V = \bigcup\limits_{i=1}^{r} L_i; \\ \forall i \neq j \in 1, \ldots, r (L_i \bigcap L_j = \emptyset); \\ \forall (v_1, v_2) \in E(\forall i \neq j \in \{1, \ldots, r\} \\ (v_1 \in L_i \& v_2 \in L_j \Rightarrow i < j)). \end{array}\right\} \tag{13}$$

Последнее условие означает, что если из вершины v_1 идет дуга в вершину v_2, то вершина v_2 должна располагаться на ярусе с большим номером по отношению к ярусу, на котором располагается вершина v_1. Количество вершин в ярусе L_i называется его *шириной*. Количество ярусов в ЯПФ называется ее *высотой*, а максимальная ширина ее ярусов — *шириной ЯПФ*. ЯПФ называется *канонической* [19], если все *входные* (не имеющие входных дуг) вершины принадлежат ярусу с номером 1, и максимальная длина пути, оканчивающегося в вершине, принадлежащей ярусу k, равна $k - 1$.

Пусть в распланированном графе $G = \langle V, E, init, fin, \delta, \gamma, \omega, \xi\rangle$ задан простой путь $y = (e_1, e_2, \ldots, e_n)$. *Стоимостью пути* y называется величина

$$u(y) = \chi\left(fin(e_n), j_{fin(e_n)}\right) + \\ + \sum_{i=1}^{n} (\chi(init(e_i), j_{init(e_i)}) + \\ + \max(\sigma\,(e_i)\,, \tau_{fin(e_i)} - s_{init(e_i)})), \tag{14}$$

где χ — функция, определяемая формулой (3), значением которой является вычислительная стоимость вершины; σ — функция, определяемая формулой (5), значением которой является коммуникационная стоимость дуги; j_v и τ_v определяются по формуле (6); s_v определяется по формуле (7).

Пусть Y — множество всех простых путей в распланированном графе $G = \langle V, E, init, fin, \delta, \gamma, \omega, \xi\rangle$. Простой путь $\bar{y} \in \mathrm{Y}$ называется *критическим*, если

$$u(\bar{y}) = \max_{y \in Y} u(y)\ , \tag{15}$$

то есть, критический путь обладает максимальной стоимостью.

Утверждение. Любой критический путь в распланированном графе $G = \langle V, E, init, fin, \delta, \gamma, \omega, \xi$ начинается с *входной* вершины и заканчивается в выходной (не имеющей выходных дуг) вершине.

Доказательство. Доказательство проведем методом от противного. Предположим, что существует критический путь $\bar{y} = (e_1, e_2, \ldots, e_n) \in Y$, начинающийся не с входной вершины. Тогда существует дуга e_h такая,

что $fin(e_h) = init(e_h)$. Так как G ациклический, то отсюда следует, что $\tilde{y} = (e_1, e_2, \ldots, e_n) \in Y$. В силу (2), (3) и (14) получаем $u(\bar{y}) < u(\tilde{y})$. Это противоречит (15). Аналогичным образом приходим к противоречию при предположении, что существует критический путь, заканчивающийся не в выходной вершине. Утверждение доказано.

4. АЛГОРИТМ POS

В данном разделе описывается разработанный авторами алгоритм POS (Problem-Oriented Scheduling) планирования ресурсов в распределенных проблемно-ориентированных вычислительных средах. Отличительной особенностью алгоритма POS является то, что при планировании ресурсов он учитывает знания о специфике предметной области. В рамках модели, описанной в разделе 3, эти знания выражаются в метках задач-вершин, задающих время выполнения задачи на одном ядре и ее масштабируемость, и в весах дуг, задающих объемы передаваемых данных. Алгоритм POS ориентирован на применение в распределенных вычислительных системах с многоядерными процессорами.

Для упрощения описания и понимания алгоритма POS мы будем использовать трехуровневую структуру процедур, представляющих алгоритм. Процедура первого уровня является головной. Некоторый шаг процедуры первого уровня может быть описан как процедура второго уровня. Такой шаг выделяется полужирным шрифтом. Аналогичный подход может быть применен в описании процедуры второго уровня.

4.1. *Головная процедура*

Пусть задана вычислительная система в виде упорядоченного множества вычислительных узлов $\mathfrak{P} = \{P_0, \ldots, P_{k-1}\}$. Пусть имеется граф задания $G = \langle V, E, init, fin, \delta, \gamma\rangle$. Предположим, что выполняются следующие условия:

$$|V| \le |\mathfrak{P}|, \tag{16}$$

$$\forall v \in V(\forall P \in \mathfrak{P}(m_v \le |P|)), \tag{17}$$

где m_v порог линейной масштабируемости, задаваемый функцией разметки γ. Зададим для графа G разбиение в каноническую ЯПФ с ярусами $L_i (i = 1, \ldots, r)$. Пронумеруем вершины $V = (v_1, \ldots, v_q)$ графа G таким образом, чтобы выполнялось следующее свойство:

$$\begin{aligned}&\forall i, j \in \{1, \ldots, q\}\\ &((v_i \in L_a \& v_j \in L_b \& a < b) \Rightarrow i < j),\end{aligned} \tag{18}$$

то есть на нижних ярусах располагаются вершины с большими номерами.

В самом общем виде головная процедура выглядит следующим образом.

Шаг 1. **Построить начальную конфигурацию** G_0;
Шаг 2. $i := 0$;
Шаг 3. **Построить конфигурацию** G_i;
Шаг 4. Если остались нерассмотренные дуги, $i := i + 1$ и перейти на шаг 3;
Шаг 5. **Уплотнить конфигурацию** G_{i+1};
Шаг 6. Стоп.

Таким образом, работа процедуры заключается в построении последовательности конфигураций. При этом, при переходе к очередной конфигурации, как минимум одна дуга графа помечается как просмотренная. Поскольку количество дуг конечно, процедура завершится на некоторой итерации. Последняя построенная конфигурация G_{i+1} выбирается как результирующая.

4.2. *Процедура построения начальной конфигурации*

Шаг 1.1. Зададим функцию начальной кластеризации ω_0 следующим образом:
$\forall i \in \{1, \ldots, q\}(\omega_0(v_i) = P_{i+1})$.
то есть каждая вершина отображается на отдельный вычислительный узел, и, соответственно, каждый кластер включает в себя только одну вершину.

Шаг 1.2. Зададим начальное расписание $\xi_0(v) = (\tau_v, j_v)$ следующим образом.
Определим время запуска τ_v итерационно по уровням ЯПФ:

$$\left.\begin{aligned}&\forall v \in L_1(\tau_v := 0);\\ &\forall v \in L_{i>1}(\tau_v := \max_{v'' \in L_i; v' \in L_{j<i}} (\lambda(v', v''))).\end{aligned}\right\}$$

Здесь

$$\lambda(v',v'') = \begin{cases} s_{v'}, if\ (v',v'') \notin E; \\ s_{v'} + \sigma((v',v'')), if\ (v',v'') \in E, \end{cases}$$

где $s_{v'}$ вычисляется по формуле (9).
Определим количество ядер j_v, выделяемых вершине v, следующим образом: $\forall v \in V(j_v = m_v)$.
Шаг 1.3. $G_0 = \langle V, E, init, fin, \delta, \gamma, \omega_0, \xi_0 \rangle$.
Шаг 1.4. Конец процедуры.

4.3. *Процедура построения конфигурации G_{i+1}*

Определим *субкритический путь*, как путь, имеющий максимальную стоимость среди всех путей, содержащих хотя бы одну нерассмотренную дугу. Процедура построения конфигурации G_{i+1} выглядит следующим образом.
Шаг 3.1. Найти в $\tilde{y}_i = (e_1, \ldots, e_n)$ субкритический путь;
Шаг 3.2. Найти первую от начала пути нерассмотренную дугу e_j $(1 \le j \le n)$ в $\tilde{y}_i$ и пометить ее как рассмотренную;
Шаг 3.3. Если $i = 0$, то пометить вершину $init(e_j)$ как зафиксированную;
Шаг 3.4. Если вершины $init(e_j)$ и $fin(e_j)$ зафиксированы, то перейти на шаг 3.14;
Шаг 3.5. Если вершина $fin(e_j)$ не зафиксирована, то $v'' := fin(e_j), v' := init(e_j)$;
Шаг 3.6. Если вершина $init(e_j)$ не зафиксирована, то $v'' := init(e_j), v' := fin(e_j)$;
Шаг 3.7. Построить функцию кластеризации ω_{i+1}, отличающуюся от функции ω_i только в одном значении: $\omega_{i+1}(v'') := \omega_i(v')$;
Шаг 3.8. **Построить расписание** ξ_{i+1};
Шаг 3.9. $G_{i+1} = \langle V, E, init, fin, \delta, \gamma, \omega_{i+1}, \xi_{i+1} \rangle$;
Шаг 3.10. Найти критический путь $\bar{y}_i$ в G_i;
Шаг 3.11. Найти критический путь $\overline{y_{i+1}}$ в G_{i+1};
Шаг 3.12. Если $u(\overline{y_{i+1}}) \le u(\bar{y}_i)$, то перейти на шаг 3.16;
Шаг 3.13. $G_{i+1} := G_i$;
Шаг 3.14. Если в $\tilde{y}_i$ остались нерассмотренные дуги, перейти на шаг 3.2;
Шаг 3.15. Если в G_i, остались нерассмотренные дуги, перейти на шаг 3.1;
Шаг 3.16. Конец процедуры.

4.4. *Процедура построения расписания ξ_{i+1}*

Введем следующие обозначения: $T(x)$ – номер яруса, которому принадлежит вершина x; $W_{\omega_i(x)} = \{v | v \in V, \omega_i(v) = \omega_i(x)\}$ – кластер, которому принадлежит вершина x. Процедура построения расписания ξ_{i+1} включает в себя следующие шаги.
Шаг 3.8.1. $R := W_{\omega_i(v')} \bigcap L_{T(v'')}$;
Шаг 3.8.2. Если $R = \emptyset$ или $\sum\limits_{v \in R} j_v \le |P_{\omega_i(v')}|$, то перейти на шаг 3.8.7;
Шаг 3.8.3. Для $h = q, \ldots, T(fin(e_j) + 1)$ выполнить $L_{h+1} := L_h$;
Шаг 3.8.4. $L_{T(v'')+1} := \{v''\}$; $L_{T(v'')} := L_{T(v'')} \setminus \{v''\}$;
Шаг 3.8.5. $q := q + 1$;
Шаг 3.8.6. Построить новое расписание ξ_{i+1} путем вычисления времени запуска τ_v всех вершин $v \in V$;
Шаг 3.8.7. Пометить вершину v'' как зафиксированную;
Шаг 3.8.8. Конец процедуры.

4.5. *Процедура уплотнения конфигурации G_{i+1}*

Целью процедуры уплотнения является минимизация количества задействованных вычислительных узлов. Данная процедура применяется только к кластерам, содержащим одну вершину. Указанное ограничение мотивировано тем, что, если в кластере имеются две смежные вершины, то перемещение одной из них в другой кластер может увеличить общее время выполнения задачи. Процедура уплотнения конфигурации G_{i+1} имеет следующий вид.
Шаг 5.1. $\mathfrak{M} := \emptyset$;
Шаг 5.2. Для всех $v' \in V$ выполнить цикл
Шаг 5.2.1. $W = \{v | v \in V, \omega_{i+1}(v) = \omega_{i+1}(v')\}$;
Шаг 5.2.2. Если $W \in \mathfrak{M}$, перейти к следующей итерации цикла;
Шаг 5.2.3. $\mathfrak{M} := \mathfrak{M} \bigcup \{W\}$;
Шаг 5.3. Конец цикла;
Шаг 5.4. $\mathfrak{E} := \{W \in \mathfrak{M} || W| = 1\}$; $\mathfrak{V} := \{W \in \mathfrak{M} || W| > 1\}$;
Шаг 5.5. Для всех $W' \in \mathfrak{E}$ выполнить цикл

Шаг 5.5.1. Для $l = 1, \ldots, r$ выполнить цикл
Шаг 5.5.1.1. Если $W' \bigcap L_l = \emptyset$, перейти к следующей итерации цикла;
Шаг 5.5.1.2. Для всех $W'' \in \mathfrak{V}$ выполнить цикл
Шаг 5.5.1.2.1. Если $W'' \bigcap L_l = \emptyset$, перейти к следующей итерации цикла;
Шаг 5.5.1.2.2. Взять $v' \in W'$;
Шаг 5.5.1.2.3. Взять $v'' \in W''$;
Шаг 5.5.1.2.4. Если $j_{v'} + \sum\limits_{v \in W'' \bigcap L_l} j_v > \omega_{i+1}(v'')$, перейти к следующей итерации цикла;
Шаг 5.5.1.2.5. $i := i + 1$;
Шаг 5.5.1.2.6. Построить функцию кластеризации ω_{i+1}, отличающуюся от функции ω_i только в одном значении: $\omega_{i+1}(v') := \omega_i(v'')$;
Шаг 5.5.1.2.7. $W'' := W'' \bigcup W'$;
Шаг 5.5.1.2.8. Перейти на шаг 5.5.3;
Шаг 5.5.1.3. Конец цикла;
Шаг 5.5.2. Конец цикла;
Шаг 5.5.3. Перейти к следующей итерации цикла;
Шаг 5.6. Конец цикла;
Шаг 5.7. Конец процедуры.

Дадим пояснения к процедуре уплотнения. Шаг 5.2 организует цикл, строящий $\mathfrak{M}$ – множество всех кластеров, на которые разбито задание. Шаг 5.4 вычисляет $\mathfrak{E}$ – множество *единичных кластеров* (кластеров, содержащих только одну вершину) и $\mathfrak{V}$ – множество *мультикластеров* (кластеров, содержащих две или более вершин). Шаг 5.5 организует цикл по всем единичным кластерам. Для каждого единичного кластера находим ярус параллельной формы, к которому принадлежит данный единичный кластер. В рамках этого яруса пытаемся объединить единичный кластер с некоторым мультикластером. Это возможно, если мультикластер задействует не все процессорные ядра узла, и количество свободных ядер достаточно для выполнения присоединяемого единичного кластера.

5. ИССЛЕДОВАНИЕ АЛГОРИТМА POS

Исследование алгоритма планирования ресурсов POS проводилось на следующих двух классах заданий:

1. многокритериальная оптимизация (МКО);
2. случайные задания (СЗ).

Рассмотрим оба этих класса.

5.1. *Класс МКО*

Класс МКО представляет вычислительные задания по многокритериальной оптимизации, составляющие большой процент загрузки современных суперкомпьютерных и распределенных вычислительных систем. Графы заданий класса МКО имеют следующую структуру: ярусно-параллельная форма состоит из трех ярусов. Первый и третий ярусы параллельной формы содержат по одной вершине. Второй ярус содержит w вершин. Число w указывается при генерации графа. Вершина первого яруса соединена дугами со всеми вершинами второго яруса. Вершина третьего яруса также соединена дугами со всеми вершинами второго яруса. Каждой задаче-вершине сопоставляется пара чисел: максимальная масштабируемость задачи m_ν и время выполнения задачи на одном процессорном ядре t_ν. Числа m_ν и t_ν являются константами и имеют для всех задач одинаковые значения. Аналогично, каждой дуге сопоставлялась константа δ – объем передаваемых данных, – одинаковая для всех дуг. На рис. 4 приведен пример графа задания класса МКО для ширины $w = 100$.

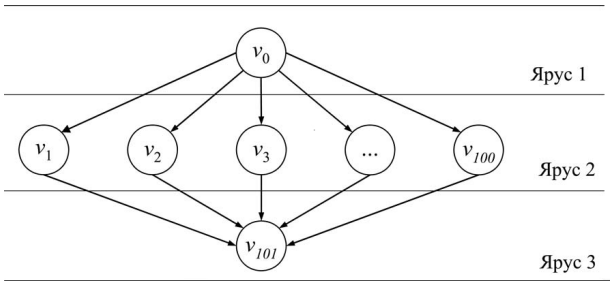

Рис. 4.
Пример графа задания класса МКО.

5.2. *Класс СЗ*

Класс СЗ представляет случайные задания с различным числом вершин и дуг. Качественной характеристикой графа задания в классе СЗ является отношение T/Δ, где T – среднее время выполнения задачи на одном ядре, Δ – средний

вес дуги. По этому параметру в классе СЗ могут быть выделены следующие три важные группы [17].

1. $M1$ – *сбалансированные* графы заданий, у которых $T/\Delta \in (0.8, 1.2)$. В таких заданиях время на передачу данных сопоставимо с временем вычислений.

2. $M2$ – *крупнозернистые* графы заданий, у которых $T/\Delta \in (3, 10)$. В таких заданиях большая часть времени тратится на вычисления.

3. $M3$ – *мелкозернистые* графы заданий, у которых $T/\Delta \in (0.1, 0.3)$. В таких заданиях значительное время затрачивается на передачу данных, а вычисления требуют незначительного времени.

5.3. *Результаты экспериментов*

В данном разделе приводятся результаты вычислительных экспериментов по исследованию алгоритма POS и его сравнению с известными алгоритмами.

В первой серии экспериментов исследовалась плотность расписания, генерируемого алгоритмом POS. Под плотностью здесь понимается величина, обратная количеству вычислительных узлов, задействованных для выполнения задания. Сначала плотность расписания была исследована для задач класса МКО. Эксперименты проводились для трех значений параметра w, определяющего ширину графа задания: 100, 200, 300. Для всех вершин и дуг графа использовались одинаковые значения весов и маркировки, приведенные в табл. 1.

Таблица 1. Параметры для построения графов заданий класса МКО

Параметр	Семантика	Значение
m_ν	Масштабируемость задачи	10
t_ν	Время выполнения задачи на 1 ядре	100
δ	Объем данных, передаваемых по дуге	50

Результаты экспериментов приведены на рис. 5а в виде зависимости количества задействованных вычислительных узлов от количества процессорных ядер в одном вычислительном узле. Графики показывают, что при увеличении количества ядер в вычислительном узле плотность расписания для заданий МКО возрастает и стремится к максимальному значению 1 во всех рассмотренных случаях. При этом плотность расписания существенно возрастает при увеличении ширины графа задания.

Затем плотность расписания была исследована для задач класса СЗ. Эксперименты проводились для трех значений параметра w, определяющего ширину графа задания: 30, 50, 70. Для всех вершин и дуг графа использовались одинаковые значения весов и маркировки, приведенные в табл. 2.

Таблица 2. Параметры для построения графов заданий класса МКО

Параметр	Семантика	Значение
l	Высота графа задания	10
m_ν	Масштабируемость задачи	10
t_ν	Время выполнения задачи на 1 ядре	100
δ	Объем данных, передаваемых по дуге	50

Результаты экспериментов приведены на рис. 5б. Графики показывают, что при увеличении количества ядер в вычислительном узле плотность расписания для заданий СЗ также возрастает и стремится к максимальному значению 1 во всех рассмотренных случаях. При этом плотность расписания существенно возрастает при увеличении ширины графа задания.

5.4. *Сравнение POS с другими алгоритмами*

Во второй серии экспериментов была исследована эффективность алгоритма POS в сравнении с двумя другими алгоритмами планирования: DSC [17] и Min-Min [16]. Для этого были подготовлены три группы графов заданий $M1$, $M2$ и $M3$ с параметрами, приведенными в табл. 3.

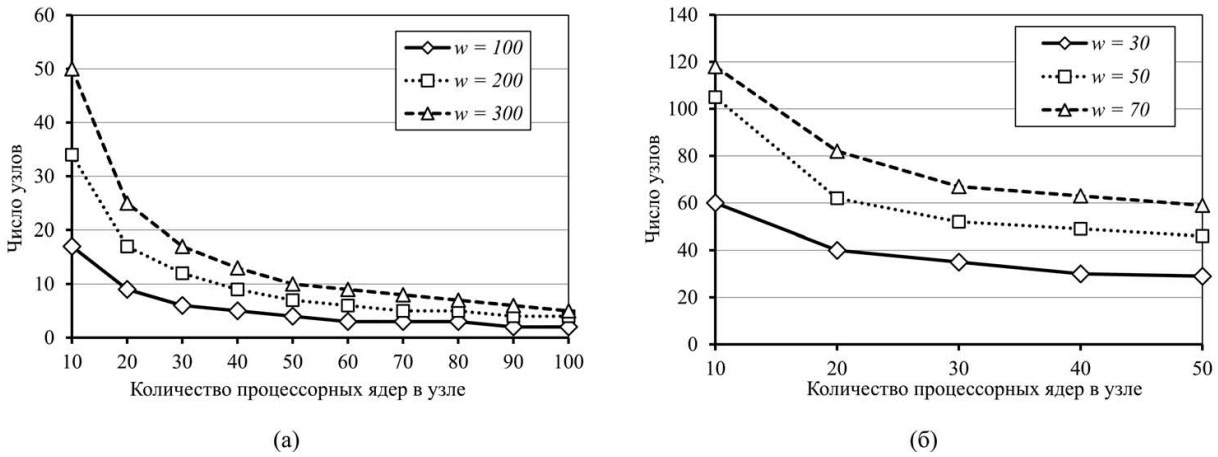

Рис. 5.
Зависимость количества задействованных вычислительных узлов от количества процессорных ядер в вычислительном кластере (а) для задания МКО (б) для класса СЗ.

Таблица 3. Параметры построения графов заданий

Параметр	Семантика	Значение
m_ν	Масштабируемость задачи	20
t_ν	Время выполнения задачи на 1 ядре	40
δ_{M1}	Средний вес дуги для группы M1	40
δ_{M2}	Средний вес дуги для группы M2	10
δ_{M3}	Средний вес дуги для группы M3	400
d	Число ядер на вычислительном узле	100

В каждой группе варьировались высота l и ширина w графа. Результаты экспериментов представлены в табл. 4, табл. 5 и табл. 6. Для сравнения алгоритма POS с алгоритмами DSC и Min-Min вычислялись количество задействованных вычислительных узлов и отношение времени выполнения задания алгоритмами DSC и Min-Min ко времени выполнения задания алгоритмом POS.

Проведенные вычислительные эксперименты показывают, что для различных заданий классов МКО и СЗ алгоритм POS генерирует расписания, которые качественно превосходят по эффективности расписания, генерируемые известными алгоритмами планирования DSC и Min-Min. Это достигается за счет того, что алгоритм POS анализирует все зависимости задач по данным, как и алгоритм DSC, и обеспечивает возможность распределения задач по процессорным ядрам, подобно алгоритму Min-Min.

6. ЗАКЛЮЧЕНИЕ

В статье рассмотрены вопросы, связанные с управлением ресурсами в проблемно-ориентированных распределенных вычислительных средах. Предложена новая модель вычислительной среды и алгоритм планирования ресурсов *POS (Problem-Oriented Scheduling)* для заданий с потоковой структурой. Данный алгоритм позволяет планировать запуск одной задачи на нескольких процессорных ядрах с учетом ограничений по масштабируемости данной задачи. Приведены результаты вычислительных экспериментов по исследованию адекватности и эффективности разработанного алгоритма для проблемно-ориентированных распределенных вычислительных сред.

СПИСОК ЛИТЕРАТУРЫ

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. V. 14. I. 13–15. P. 1507–1542.

Таблица 4. Сравнение алгоритмов POS, DSC и Min-Min для группы M1

2*№ п/п	2*l	2*w	2*$\|V\|$	2*$\|E\|$	Количество задействованных вычислительных узлов			Отношение времени выполнения задания	
					POS	DSC	Min-Min	DSC/POS	Min-Min/POS
1	5	10-20	51	126	7	21	4	3.78	8.86
2	10	10-20	117	296	4	36	4	3.67	9.53
3	10	20-30	211	505	16	68	6	3.77	8.36
4	20	5-10	137	252	2	46	2	10.19	17.74
Среднее значение								5.35	11.13

Таблица 5. Сравнение алгоритмов POS, DSC и Min-Min для группы M2

2*№ п/п	2*l	2*w	2*$\|V\|$	2*$\|E\|$	Количество задействованных вычислительных узлов			Отношение времени выполнения задания	
					POS	DSC	Min-Min	DSC/POS	Min-Min/POS
1	5	10-20	49	113	3	25	4	6.67	5.67
2	10	10-20	130	390	12	42	4	4.79	2.51
3	10	20-30	206	464	17	73	6	4.05	3.47
4	20	5-10	141	290	13	41	2	4.79	1.47
Среднее значение								5.08	3.28

Таблица 6. Сравнение алгоритмов POS, DSC и Min-Min для группы M3

2*№ п/п	2*l	2*w	2*$\|V\|$	2*$\|E\|$	Количество задействованных вычислительных узлов			Отношение времени выполнения задания	
					POS	DSC	Min-Min	DSC/POS	Min-Min/POS
1	5	10-20	59	190	4	21	4	4.50	13.15
2	10	10-20	123	378	3	34	4	4.89	42.46
3	10	20-30	201	453	9	61	6	2.06	10.78
4	20	5-10	133	287	2	29	2	3.13	7.88
Среднее значение								3.64	18.57

2. *Foster I., Kesselman C.* The Grid. Blueprint for a new computing infrastructure. San Francisco: Morgan Kaufman, 1999. 677 p.

3. *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 2000. P. 181–188.

4. *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. V. 15. № 3. P. 200–222.

5. *Jennings R.* Cloud Computing with the Windows Azure Platform, 2009. 360 p.

6. *Marshall, P., Keahey K., Freeman, T.* Improving Utilization of Infrastructure Clouds // Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2011), Newport Beach, CA. May 2011. P. 205–214.

7. *Berman F., Wolski R. et al.* Application-Level Scheduling on Distributed Heterogeneous Networks // Proceedings of the ACM/IEEE conference on Supercomputing, Pittsburgh, Pennsylvania USA, 1996. Article No. 39.

8. *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 1998. P. 70–78.

9. *Khokhar A.A., Prasanna V.K. et al.* Heterogeneous Computing: Challenges and Opportunities // IEEE Computer. 1993. V. 26. № 6. P. 18–27.

10. *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. V. 59. № 2. P. 107–131.

11. *Zhu Y., Ni L. M.* A Survey on Grid Scheduling Systems // Technical Report No. SJTU_CS_TR_200309001, Department of Computer Science and Engineering, Shanghai Jiao Tong University, 2013. 42 p.

12. *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), December 3–6, 2001, Kyoto, Japan. IEEE Computer Society. V. 1. № 1. P. 72–80.

13. *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. 55 p.

14. *Kim S.J.* A general approach to multiprocessor scheduling // Report TR-88-04. Department of Computer Science, University of Texas at Austin, 1988. 155 p.

15. *Yang T., Gerasoulis A.* DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors // Proceedings of the IEEE Transactions on Parallel and Distributed Systems. 1994. V. 5. № 9. P. 951–967.

16. *Yu J., Buyya R., Ramamohanarao K.* Workflow Scheduling Algorithms for Grid Computing. Metaheuristics for Scheduling in Distributed Computing Environments Studies in Computational Intelligence. Springer Berlin Heidelberg. 2008. V. 146. P. 173–214.

17. *Gerasoulis A., Yang T.* A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors // Journal of Parallel and Distributed Computing. 1992. V. 16. № 4. P. 276–291.

18. *Sarkar V.* Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors. The MIT Press, Cambridge, MA, 1989. P. 215.

19. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. Санкт-Петербург: БХВ-Петербург, 2002. 608 с.