АЛГОРИТМ ПЛАНИРОВАНИЯ РЕСУРСОВ POS ДЛЯ РАСПРЕДЕЛЕННЫХ ПРОБЛЕМНО-ОРИЕНТИРОВАННЫХ СРЕД

А.В. Шамакина

Южно-Уральский государственный университет

ВВЕДЕНИЕ

На сегодняшний день предложено большое количество алгоритмов планирования [1-7], ориентированных на использование в распределенных вычислительных средах. Только очень небольшая часть этих алгоритмов учитывает проблемно-ориентированную специфику потоков работ в сложных приложениях, которая выражается в указании времени выполнения задачи и объеме передаваемых данных. Одним из наиболее существенных достижений в этой области является алгоритм DSC [5]. Алгоритм планирования DSC представляет задание в виде ориентированного графа, узлами которого являются задачи, а дуги соответствуют потокам данных. При планировании алгоритм DSC учитывает время выполнения задач и объем передаваемых данных. Существенным ограничением данного алгоритма применительно к современным кластерным вычислительным системам является то, что каждая задача может выполняться только на одном процессорном ядре.

Нами разработан новый алгоритм планирования POS (Problem-Oriented Scheduling) для распределенных кластерных вычислительных сред, который в отличие от алгоритма DSC позволяет планировать запуск одной задачи на нескольких процессорных ядрах с учетом ограничений по масштабируемости данной задачи.

1. ФОРМАЛЬНАЯ МОДЕЛЬ ПРОБЛЕМНО-ОРИЕНТИРОВАННОЙ СРЕДЫ

Перед описанием алгоритма планирования POS приведем формальную модель задания, для которой нам понадобятся следующие базовые определения.

1.1. БАЗОВЫЕ ОПРЕДЕЛЕНИЯ

Ориентированным графом называется четверка $G=\langle V, E, init, fin\rangle$, где V – множество вершин; E – множество дуг; $init: E \to V$ – функция, определяющая *начальную вершину* дуги; $fin: E \to V$ – функция, определяющая *конечную вершину* дуги.

Вершина $v_1, v_2 \in V$ называются *смежными*, если

$$\exists e \in E\left(\left(v_1 = init(e) \wedge v_2 = fin(e)\right) \vee \left(v_1 = fin(e) \wedge v_2 = init(e)\right)\right), \quad (1)$$

другими словами, существует дуга e, соединяющая эти вершины. Если $v_1 = init(e) \wedge v_2 = fin(e)$, мы будем обозначать это следующим образом: $(v_1, v_2) = e \in E$ и говорить, что вершины v_1, v_2 *инцидентны* дуге e.

Пусть $v_1, v_2 \in V$ и $n \geq 1$. Упорядоченная последовательность дуг $(e_1, e_2, \ldots, e_n) \in E^n$ называется *путем длины n*, от вершины v_1 к вершине v_2, если $v_1 = init(e_1)$, $v_2 = fin(e_n)$ и $fin(e_i) = init(e_{i+1})$ для всех $i \in \{1, \ldots, n-1\}$. Если $(e_1, e_2, \ldots, e_n) \in E^n$ – путь от вершины v_1 к вершине v_2, то *обратным путем* от вершины v_2 к вершине v_1 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 $init(e_1), init(e_2), \ldots, init(e_n)$ все различны между собой и если $fin(e_1), fin(e_2), \ldots, fin(e_n)$ тоже все различны. *Циклом* называется простой путь от некоторой вершины к ней самой. Ориентированный граф называется *ациклическим*, если он не содержит циклов [8].

Вершины $v_1, v_2 \in V$ называются *независимыми*, если не существует прямого или обратного пути от вершины v_1 к вершине v_2. В противном случае, вершины v_1, v_2 *зависимы*.

Пусть задан ориентированный граф $G=\langle V, E, init, fin\rangle$. *Взвешиванием графа G* будем называть функцию $\sigma : E \to Z_{\geq 0}$. *Разметкой графа G* будем называть функцию

$$\gamma : V \to N^2. \quad (2)$$

1.2. МОДЕЛЬ ВЫЧИСЛИТЕЛЬНОЙ СРЕДЫ

Теперь мы готовы дать формальное определение задания в распределенной вычислительной среде.

Графом задания называется 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.

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

$$\gamma(v) = (m_v, t_v) \quad (3)$$

определяет максимальное количество процессорных ядер m_v, на которых задача v имеет ускорение, близкое к *линейному*, и время t_v выполнения задачи v на одном ядре. Данная модель предполагает, что *вычислительная стоимость* $\chi(v, j_v)$ задачи v на j_v процессорных ядрах определяется следующей формулой:

$$\chi(v, j_v)=\begin{cases} t_v/j_v, \text{если } 1\le j\le m_v; \\ t_v/m_v, \text{если } m_v< j_v. \end{cases} \quad (4)$$

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

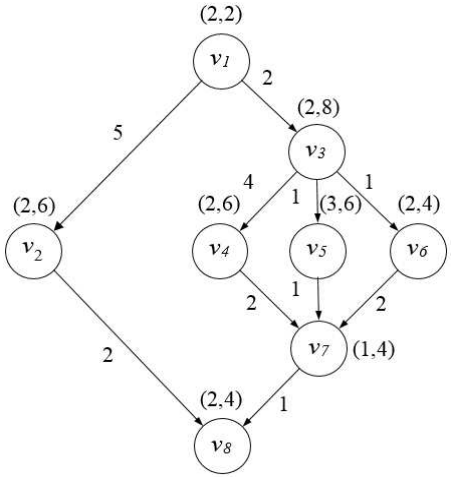

Рис. 1. Граф задания

На рис. 1 приведен пример графа задания, содержащего 8 вершин. Для каждой из вершин указана метка в виде пары (m_v, t_v). Каждой дуге графа сопоставляется вес – объем данных $\delta(e)$, передаваемых по данной дуге.

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

Вычислительной системой называется упорядоченное множество вычислительных узлов $\mathsf{P}=\{P_0, \ldots, P_{k-1}\}$. 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 $\omega: V \to \mathsf{P}$ множества вершин V графа задания G на множество вычислительных узлов P.

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

$$W_i=\{v\in V \mid \omega(v)=P_i\in \mathsf{P}\}. \quad (5)$$

Имеем

$$W_i \cap_{k-1} W_j=\varnothing \text{ для } i\ne j; \quad (6)$$

$$V=\bigcup_{i=0} W_i. \quad (7)$$

Пусть задан граф задания $G=\langle V, E, init, fin, \delta, \gamma\rangle$, для которого определена функция кластеризации $\omega(v)$. 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, близко к 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 $\sigma: E \to Z_{\ge 0}$, вычисляющую *коммуникационную стоимость* (время) передачи данных по дуге $e\in E$, следующим образом:

$$\sigma(e)=\begin{cases} 0, \text{ если } \omega(init(e))=\omega(fin(e)); \\ \delta(e), \text{если } \omega(init(e))\ne\omega(fin(e)). \end{cases} \quad (8)$$

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

$$\xi(v)=(\tau_v, j_v), \quad (9)$$

где τ_v определяет время запуска задачи v, j_v – количество процессорных ядер, выделяемых задаче v. Обозначим через s_v – время останова задачи v. Имеем

$$s_v = \tau_v + \chi(v, j_v) \quad , \tag{10}$$

где χ – функция временной сложности, определенная с помощью формулы (4). Расписание называется *корректным*, если оно удовлетворяет следующим условиям:

$$\forall\ e \in E\ \left(\tau_{fin(e)} \geq \tau_{init(e)} + \chi\left(init(e), j_{init(e)}\right) + \sigma(e)\right)\ ; \tag{11}$$

$$\forall\ v \in V\ \left(j_v \leq m_v\right)\ ; \tag{12}$$

$$\forall t \in \mathbb{N}\left(\forall i \in [0,\ldots,k-1]\left(\sum_{v \in W_i \wedge \tau_v < t \leq s_v} j_v \leqslant |P_i|\right)\right)\ . \tag{13}$$

Условие (11) означает, что для любых двух смежных вершин $v_1 = init(e)$ и $v_2 = fin(e)$ время запуска v_2 не может быть меньше суммы следующих величин: время запуска v_1, время выполнения v_1, коммуникационная стоимость дуги *e*. Условие (12) означает, что количество ядер, выделяемое задаче v_1, не превышает границы линейной масштабируемости, заданной разметкой γ в контексте формулы (3). Условие (13) означает, что в любой момент времени t общее количество процессорных ядер, выделяемых задачам на узле с номером i, не может превосходить количества ядер на этом узле. В дальнейшем мы будем рассматривать только корректные расписания, если явно не оговорено противное.

Кластеризованный граф $G = \langle V, E, init, fin, \delta, \gamma, \omega \rangle$ 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*) [9] 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:

$$\left.\begin{array}{l} V = \bigcup_{i=1}^{r} L_i; \\ \forall i \neq j \in \{1,\ldots,r\}\left(L_i \cap L_j = \varnothing\right); \\ \forall (v_1, v_2) \in E\left(\forall i \neq j \in \{1,\ldots,r\}\left(v_1 \in L_i \wedge v_2 \in L_j \Rightarrow i < j\right)\right). \end{array}\right\} \tag{14}$$

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* [10], 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 путь $y = (e_1, e_2, \ldots, e_n)$. Так как граф G ациклический, то любой путь в нем, в том числе и y, будет простым. *Стоимостью* пути $u(y)$ называется величина

$$u(y) = \chi(fin(e_n), j_{fin(e_n)}) + \sum_{i=1}^{n} \left(\chi(init(e_i), j_{init(e_i)}) + max(\sigma(e_i), \tau_{fin(e_i)} - s_{init(e_i)})\right)\ , \tag{15}$$

где χ – функция, определяемая формулой (4), значением которой является вычислительная стоимость вершины; σ – функция, определяемая формулой (8), значением которой является коммуникационная стоимость дуги; j_v и τ_v определяется по формуле (9); s_v определяется по формуле (10).

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

$$u(\bar{y}) = \max_{y \in Y} u(y)\ , \tag{16}$$

то есть, критический путь обладает максимальной стоимостью.

2. АЛГОРИТМ ПЛАНИРОВАНИЯ POS

В данном разделе описывается новый *алгоритм POS* (*Problem-Oriented Scheduling*) планирования ресурсов в распределенных проблемно-ориентированных средах. Отличительной особенностью алгоритма POS является то, что при планировании ресурсов он учитывает знания о специфике предметной области. В рамках модели, описанной в разделе 1.2, 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.

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.

2.1. ГОЛОВНАЯ ПРОЦЕДУРА

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

$$|V| \leqslant |\mathsf{P}|\ ; \tag{17}$$

$$\forall v \in V\ (\forall P \in \mathsf{P}\ (m_v \leqslant |P|)) \tag{18}$$

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:

$$\forall i, j \in \{1,\ldots,q\}((v_i \in L_a \wedge v_j \in L_b \wedge a < b) \Rightarrow i < j)\ , \tag{19}$$

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+1} **;**

Шаг 4. Если остались нерассмотренные дуги, $i := i+1$ и перейти на шаг 3;

Шаг 5. **Уплотнить** 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.

2.2. ПРОЦЕДУРА ПОСТРОЕНИЯ НАЧАЛЬНОЙ КОНФИГУРАЦИИ G_0

Шаг 1.1. Зададим функцию начальной кластеризации ω_0 следующим образом:

$$\forall i \in \{1,\ldots,q\}(\omega_0(v_i) = P_{i-1})\ , \tag{20}$$

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 рекурсивно по уровням ЯПФ:

$$\left.\begin{array}{l} \forall v \in L_1(\tau_v := 0); \\ \forall v \in L_{i>1}\left(\tau_v := \max\limits_{v'' \in L_i;\, v' \in L_{j<i}} (\lambda(v', v''))\right). \end{array}\right\} \tag{21}$$

Здесь

$$\lambda(v', v'') = \begin{cases} s_v', \text{если} (v', v'') \notin E; \\ s_v' + \sigma((v', v'')), \text{если} (v', v'') \in E; \end{cases} \tag{22}$$

где s_v' вычисляется по формуле (10). Определим количество ядер j_v, выделяемых вершине v, следующим образом:

$$\forall v \in V(j_v = m_v) \tag{23}$$

Шаг 1.3. $G_0 := \langle V, E, init, fin, \delta, \gamma, \omega_0, \xi_0 \rangle$;

Шаг 1.4. Конец процедуры.

Утверждение. Расписание ξ_0, построенное в приведенной процедуре, является корректным.

Доказательство. Нам необходимо и достаточно проверить выполнение условий (11)-(13). Проверим сначала условие (11). Пусть $e \in E$. Из (21) имеем

$$\tau_{fin(e)} \geq s_{init(e)} + \sigma(e) = \tau_{init(e)} + \chi(init(e), j_{init(e)}) + \sigma(e)\ ,$$

то есть условие (11) выполняется. Условие (12) выполняется в силу (23). Условие (13) вытекает из (18), (20) и (23). Утверждение доказано.

2.3. ПРОЦЕДУРА ПОСТРОЕНИЯ КОНФИГУРАЦИИ G_{i+1}

Определим *субкритический путь*, как путь, имеющий максимальную стоимость среди всех путей, содержащих хотя бы одну нерассмотренную дугу.

Шаг 3.1. Найти в G_i субкритический путь $\tilde{y}_i = (e_1,\ldots,e_n)$ (если таких путей несколько, выбрать любой из них);

Шаг 3.2. Найти первую от начала пути нерассмотренную дугу $e_j\ (1 \leqslant j \leqslant n)$ 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 (если таких путей несколько выбрать любой из них);

Шаг 3.11. Найти критический путь $\overline{y_{i+1}}$ в G_{i+1} (если таких путей несколько выбрать любой из них);

Шаг 3.12. Если $u(\overline{y_{i+1}}) \leqslant u(\bar{y}_i)$, то перейти на шаг 3.16;

Шаг 3.13. $G_{i+1} := G_i$;

Шаг 3.14. Если в $\bar{y}_i$ 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.

2.4. 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.

Шаг 3.8.1. $R := W_{\omega_i(v')} \cap L_{T(v'')}$;

Шаг 3.8.2. Если $R = \varnothing$ или $\sum_{v \in R} j_v \leqslant |P_{\omega_i(v')}|$, 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. $L_{T(v'')+1} := \{v''\}$;

Шаг 3.8.5. $L_{T(v'')} := L_{T(v'')} \setminus \{v''\}$;

Шаг 3.8.6. $q := q + 1$;

Шаг 3.8.7. Построить новое расписание ξ_{i+1} путем вычисления времени запуска τ_v всех вершин $v \in V$ с помощью формулы (21);

Шаг 3.8.8. Пометить вершину v'' как зафиксированную;

Шаг 3.8.9. Конец процедуры.

2.5. ПРОЦЕДУРА УПЛОТНЕНИЯ G_{i+1}

Целью процедуры уплотнения является минимизация количества задействованных вычислительных узлов. Данная процедура применяется только к кластерам, содержащим одну вершину.

Шаг 5.1. $\mathsf{W} := \varnothing$;

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 $W \in \mathsf{W}$ перейти к следующей итерации цикла (шаг 5.2);

Шаг 5.2.3. $\mathsf{W} := \mathsf{W} \cup \{W\}$;

Шаг 5.2.4. Конец цикла (шаг 5.2);

Шаг 5.3. $\mathsf{V} := \{W_a \in \mathsf{W} \| W_a| > 1\}$;

Шаг 5.4. Пронумеруем все кластеры в V таким образом, что $a > b \Rightarrow |W_a| \geqslant |W_b|$;

Шаг 5.5. $\mathsf{N} := \mathsf{W} \setminus \mathsf{V}$;

Шаг 5.6. Пронумеруем все кластеры в N таким образом, что $\forall W_a = \{v'\}, W_b = \{v''\}, a > b \Rightarrow j_{v'} \geqslant j_{v''}$;

Шаг 5.7. Для $a := 1$ до $|\mathsf{N}|$ выполнить цикл;

Шаг 5.7.1. Для $b := 1$ до $|\mathsf{V}|$ выполнить цикл;

Шаг 5.7.1.1. Если $\sum_{v' \in W_a \cap W_b} j_{v'} > |P_b|$, перейти к следующей итерации цикла (шаг 5.7.1);

Шаг 5.7.1.2. $W_b := W_b \cup \{v'\}, W_a := W_a \setminus \{v'\}$;

Шаг 5.7.1.3. Выйти из цикла (шаг 5.7.1);

Шаг 5.7.1.4. Конец цикла (шаг 5.7.1);

Шаг 5.7.2. Конец цикла (шаг 5.7);

Шаг 5.8. Конец процедуры.

ЗАКЛЮЧЕНИЕ

Разработанная формальная модель задания позволяет описать поток работ в виде графа задания, предусматривает использование кластеризации вершин графа, возможность указать время выполнения каждой задачи и ее верхнюю границу линейной масштабируемости, а также возможность задать количество процессорных ядер для каждого из вычислительных узлов. Данная формальная модель задания является

универсальной, поскольку предоставляет возможность описать как уже известные алгоритмы кластеризации, так и новые алгоритмы для планирования ресурсов в распределенных вычислительных средах.

Алгоритм планирования POS (Problem-Oriented Scheduling) для распределенных кластерных вычислительных сред позволяет планировать запуск одной задачи на нескольких процессорных ядрах с учетом ограничений по масштабируемости данной задачи. Данный алгоритм планирования предназначен для создания системы интеллектуального планирования и диспетчеризации приложений с потоковой структурой с целью повышения эффективности функционирования суперкомпьютерных комплексов с кластерной архитектурой.

Исследование выполнено при финансовой поддержке РФФИ в рамках научного проекта № 14-07-31159.

ЛИТЕРАТУРА:

1. S.J. Kim. A general approach to multiprocessor scheduling. Report TR-88-04. Department of Computer Science, University of Texas at Austin, 1988.
2. S.J. Kim, J.C. Browne. A general approach to mapping of parallel computation upon multiprocessor architectures // Proceedings of the International Conference on Parallel Processing, 1988. Vol. 3. P. 1-8.
3. V. Sarkar. Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors. The MIT Press, Cambridge, MA, 1989. P. 215.
4. A. Gerasoulis, T. Yang. 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.
5. T. Yang, A. Gerasoulis. 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.
6. A. Shamakina. Brokering Service for Supporting Problem-Oriented Grid Environments // UNICORE Summit 2012 Proceedings, Forschungszentrum Julich, 2012. P. 67-75.
7. А.В. Шамакина. Брокер ресурсов для поддержки проблемно-ориентированных сред // Вестник ЮУрГУ. Серия "Вычислительная математика и информатика". 2012. № 46(305). Вып. 1. С. 88-98.
8. Д.Э. Кнут. Искусство программирования, т. 1. Основные алгоритмы, 3-е изд. М.: Издательский дом «Вильямс», 2000. 720 с.
9. В.В. Воеводин. Математические модели и методы в параллельных процессах. Наука, 1986. 296 с.
10. В.В. Воеводин, Вл.В. Воеводин. Параллельные вычисления. Санкт-Петербург: БХВ-Петербург, 2002. 608 с.