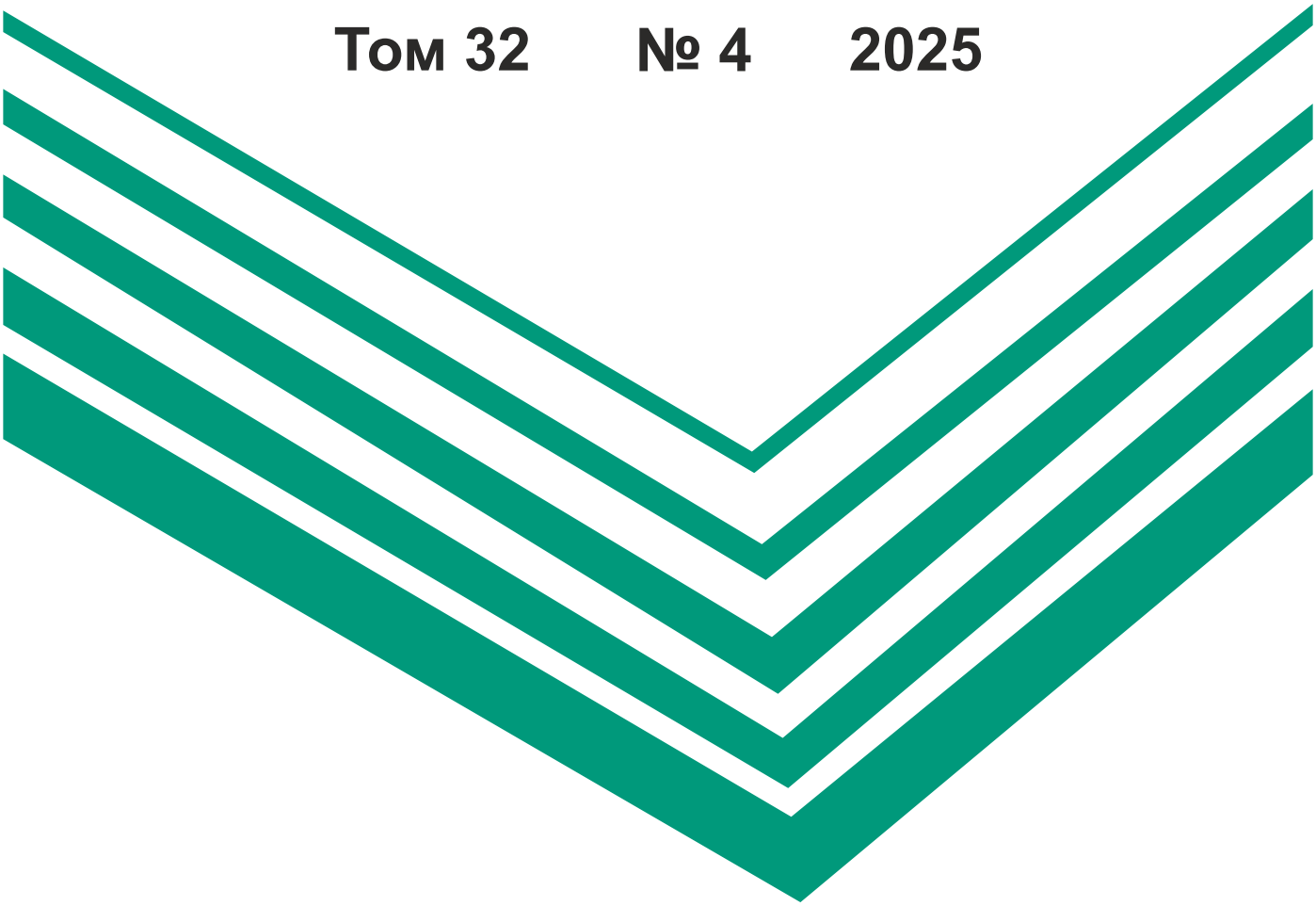
ISSN 2949-5598

ДИСКРЕТНЫЙ АНАЛИЗ И ИССЛЕДОВАНИЕ ОПЕРАЦИЙ

Том 32 № 4 2025



Новосибирск
Издательство Института математики

ДИСКРЕТНЫЙ АНАЛИЗ И ИССЛЕДОВАНИЕ ОПЕРАЦИЙ
Октябрь–декабрь 2025. Т. 32, № 4. С. 118–137

УДК 519.8 DOI: 10.33048/daio.2025.32.816

ПРИБЛИЖЁННЫЙ АЛГОРИТМ РАСПРЕДЕЛЕНИЯ ЗАДАНИЙ ПО НЕОДНОРОДНЫМ ПРОЦЕССОРАМ С ЗАДЕРЖКАМИ ПРИ ПЕРЕДАЧЕ ДАННЫХ

А. В. Демаков[1, a], *А. В. Кононов*[2, b]

[1] Новосибирский гос. университет,
ул. Пирогова, 2, 630090 Новосибирск, Россия
[2] Институт математики им. С. Л. Соболева,
пр. Акад. Коптюга, 4, 630090 Новосибирск, Россия
E-mail: [a] a.demakov@g.nsu.ru, [b] alvenko@math.nsc.ru

Аннотация. Изучается задача распределения заданий по вычислительным серверам с учётом начальной загрузки данных для их выполнения. Назначение задания на сервер, который не содержит нужного блока данных, ведёт к расходам времени на передачу блока по сети. Чем больше блоков передаётся по сети, тем больше добавка к длительности задания. Требуется минимизировать общее время выполнения всех заданий.

Для рассматриваемой задачи предложен 2-приближённый алгоритм, который использует решение задачи линейного программирования и достройку дробного решения до целого. Для вычислительных экспериментов рассмотрена ускоренная версия алгоритма. Проведены вычислительные эксперименты, которые показали, что алгоритм по качеству ответов сопоставим с известными алгоритмами. Ил. 7, библиогр. 16.

Ключевые слова: теория расписаний, приближённый алгоритм, длина расписания.

Введение

Рассмотрим задачу параллельной обработки данных, в которой требуется распределить задания по параллельным неоднородным процессорам с целью минимизировать общее время обслуживания. Обработка каждого задания требует наличия на сервере определённого блока данных. Назначение задания на сервер, который не содержит нужного блока, ведёт к расходам времени на передачу блока по сети. Такое задание будем называть *удалённым*. Если сервер содержит блок данных, то задание, назначенное на такой сервер, называется *локальным*. Чем больше блоков

передаётся по сети, тем больше добавка к длительности задания. Данная задача в упрощённой форме моделирует различные процессы параллельных вычислений, например обработку большого набора изображений, где каждое изображение может быть обработано независимо.

Задачи оптимального распределения заданий по параллельным процессорам интенсивно изучаются с 1970-х гг. Задача построения расписания минимальной длины NP-трудна уже в системе из параллельных идентичных процессоров [1]. В задачах с неоднородными процессорами время выполнения задания зависит и от задания, и от того процессора, на который оно назначено. В 1985 г. Поттс [2] предложил 2-приближённый алгоритм, основанный на округлении точного дробного решения релаксации соответствующей задачи целочисленного линейного программирования (ЦЛП), в случае, когда число машин фиксировано. Ленстра, Шмойс и Тардош [3] усилили этот результат и показали, что дробное решение может быть преобразовано в хорошее целочисленное за время, ограниченное полиномом от размера входа, и в том случае, когда число машин не фиксированно и является частью входа. Алгоритм, полученный в [3], также 2-приближённый. В той же статье авторы показали, что аппроксимация этой задачи с относительной погрешностью меньше $3/2$ влечёт совпадение классов P и NP.

Задачи, в которых требуется учесть предварительное распределение данных на серверах, изучались в статьях [4, 5], посвящённых оптимизации назначения заданий в системе Hadoop, разработанной в рамке вычислительной парадигмы MapReduce. Процесс MapReduce состоит из двух стадий. На первой стадии Map один из компьютеров получает входные данные и разделяет их на другие компьютеры для обработки. На второй стадии Reduce происходит свёртка обработанных данных. Главный сервер получает ответы и формирует результат. Вторая стадия не может начаться, пока не выполнены все задания на первой стадии.

Для эффективного распределения задач на стадии Map Фишер, Су и Ин [4] представили идеализированную модель Hadoop, которая называется задачей назначения заданий Hadoop. Для упрощения модели авторы предположили, что все серверы одинаковы, а время выполнения заданий на каждом сервере совпадает с загрузкой этого сервера. С учётом размещения входных блоков по серверам цель задачи — найти назначение, минимизирующее общее время выполнения всех заданий. В [4] авторы рассмотрели задачу с одинаковыми длительностями локальных заданий, доказали её NP-трудность и предложили полиномиальный алгоритм с абсолютной оценкой точности. Доказано, что алгоритм может ошибаться от оптимума не больше, чем на длительность одного удалённого задания в оптимальном решении. В [5] рассмотрена эта же задача с дополнением в виде начальной нагрузки серверов и предложен полиномиальный

эвристический алгоритм BAR. Оба алгоритма используют решение задачи о максимальном потоке для назначения локальных заданий и жадное улучшение решения. С учётом специфики модели Hadoop авторы обеих статей предполагают длительности всех заданий одинаковыми, что позволяет строить начальное решение потоковыми алгоритмами.

В [6] указывается, что хотя принято считать, что в модели Hadoop длительности всех заданий одинаковы, на практике часто случается перекос при выполнении заданий разными процессорами, который негативно влияет на общее время выполнения стадии Map. Существуют различные причины возникновения такого перекоса. Например, задачи Map обычно обрабатывают коллекцию записей в форме пар ключ-значение, одну за другой. В идеале время обработки не сильно различается от записи к записи. Однако в зависимости от приложения для обработки некоторых записей может потребоваться больше ресурсов ЦП и памяти, чем для других, например такая ситуация возникает при использовании приложения PageRank [7]. Другая причина в том, что хотя MapReduce — унарный оператор, его можно использовать для эмуляции n-арной операции путём логического объединения нескольких наборов данных в качестве одного входа [8]. Каждый набор данных может потребовать различной обработки, что приводит к многомодальному распределению времени выполнения задач. В [6] перечисляется и ряд других причин, ведущих к перекосу при выполнении заданий на первой стадии. Похожая причина указывается также в работе [9], где авторы рассмотрели обобщённую задачу для модели MapReduce с обеими стадиями и неоднородными серверами. В [9] предложены приближённые онлайн и оффлайн алгоритмы и проведено теоретическое исследование задачи.

В нашей работе рассматривается задача о назначениях заданий на стадии Map с неоднородными серверами и произвольными длительностями. В разд. 1 вводится формальная постановка задачи. В разд. 2 описывается приближённый алгоритм, который использует идею, предложенную в [3]. Метод состоит в решении задачи линейного программирования и достройке дробного решения до целого. В разд. 3 обсуждаются модификации алгоритма с целью улучшения его трудоёмкости, приводятся результаты вычислительных экспериментов и сравнение полученных результатов с результатами известных алгоритмов.

1. Постановка задачи

Пусть $J = \{1, \ldots, n\}$ — множество заданий, $S = \{1, \ldots, m\}$ — множество серверов. Общее время выполнения всех заданий на сервере не зависит от того, в каком порядке они выполняются, поэтому для составления расписания достаточно найти назначение заданий по серверам. Назначением заданий назовём отображение $A\colon J \to S$.

Длительность задания определяется функцией $w\colon S \times J \times \mathbb{Z}^+ \to \mathbb{Z}^+$. Длительность зависит не только от сервера и задания, но и от нагрузки сети, т. е. от количества данных, перемещаемых с одного сервера на другой. Если данные для задания находятся на выбранном сервере, то время выполнения задания будет ниже, чем если их требуется передать по сети с удалённого сервера. Локальное задание t на сервере s имеет длительность w_{ts}, а удалённое — $w_{ts} + f(p)$, где p — число удалённых заданий. Без ограничения общности будем считать, что функция f принимает в целых точках целочисленные значения. Так как количество передаваемых по сети данных увеличивает нагрузку сети, будем считать, что $f(p)$ увеличивается с ростом p. Тогда для задания t, сервера s и назначения A с количеством удалённых заданий p выполняется соотношение

$$w'_{ts} = \begin{cases} w_{ts}, & \text{если задание } t \text{ локальное для сервера } s, \\ w_{ts} + f(p) & \text{иначе.} \end{cases}$$

Размещение данных задано двудольным графом $G(J \cup S, E)$. Между вершинами $t \in J$ и $s \in S$ имеется ребро $(t, s) \in E$, если данные для задания t находятся на сервере s.

Нагрузка сервера s вычисляется по формуле $L_s^A = \sum\limits_{t\colon A(t)=s} w'_{ts} + L_s^{\text{init}}$, где L_s^{init} — нагрузка сервера в начальный момент времени. Требуется найти назначение A, минимизирующее время $C_{\max}(A) = \max\limits_{s \in S} L_s^A$ завершения всех работ. Назовём задачу поиска оптимального назначения задачей П. Отметим, что задача П является обобщением классической задачи минимизации длины расписания на различных параллельных приборах [11], которая NP-трудна в сильном смысле [12].

2. Алгоритм

В этом разделе для решения задачи П предлагается 2-приближённый алгоритм, основанный на решении серии задач линейного программирования и последующем преобразовании дробных решений в целочисленные. Сформулируем задачу П в виде задачи целочисленного линейного программирования (ЦЛП). Для этого введём переменные

$$x_{ts} = \begin{cases} 1, & \text{если задание } t \text{ назначено на сервер } s, \\ 0 & \text{иначе.} \end{cases}$$

С использованием введённых обозначений задача ЦЛП может быть записана следующим образом:

$$C_{\max} \to \min, \tag{1}$$

$$\sum_{t=1}^{|J|} x_{ts} w'_{ts} + L_s^{\text{init}} \leqslant C_{\max}, \quad s \in S, \tag{2}$$

$$\sum_{s=1}^{|S|} x_{ts} = 1, \quad t \in J, \tag{3}$$

$$x_{ts} \in \{0, 1\}, \quad t \in J,\ s \in S. \tag{4}$$

Левая часть неравенств (2) задаёт нагрузку каждого сервера. Равенства (3) гарантируют, что каждое задание назначено ровно на один сервер.

Так как длительность удалённых задач зависит от их числа p, при его изменении требуется пересчитать значения параметров w'_{ts}, поэтому будем решать задачу Π для каждого p отдельно. Обозначим через Π_p задачу Π, в которой число удалённых заданий не превышает p.

Введём двоичную матрицу $\Lambda = (\lambda_{ts})_{t\in J}^{s\in S}$, где

$$\lambda_{ts} = \begin{cases} 1, & \text{если задание } t \text{ локальное для сервера } s, \\ 0 & \text{иначе,} \end{cases}$$

и добавим в задачу (1)–(4) ограничение на число удалённых заданий:

$$\sum_{t=1}^{|J|} \sum_{s=1}^{|S|} x_{ts}(1 - \lambda_{ts}) \leqslant p. \tag{5}$$

Для фиксированного числа p задачу (1)–(5) обозначим через ЦЛП(p). Допустимое решение задачи ЦЛП(p) взаимно однозначно соответствует допустимому решению задачи Π_p, хотя значения целевых функций могут и отличаться. Действительно, пусть $\overline{x} = (x_{ts})_{t\in J}^{s\in S}$ — произвольное допустимое решение задачи ЦЛП(p). Положим $A(t) = s$ для всех $t \in J$ и $s \in S$ таких, что $x_{ts} = 1$. Ограничения (3) и (5) влекут, что назначение A является допустимым решением задачи Π_p. Пусть $C_{\max}(p, \overline{x})$ — значение целевой функции задачи ЦЛП(p) на решении $\overline{x}$. Поскольку $f(p)$ — неубывающая функция, имеем $C_{\max}(A) \leqslant C_{\max}(p, \overline{x})$. Более того, существует $p' \leqslant p$ такое, что $C_{\max}(A) = C_{\max}(p', \overline{x})$, где p' равно числу удалённых заданий в назначении A.

2.1. Параметрическое сокращение. Одним из широко известных методов построения приближённых алгоритмов для комбинаторных задач является решение релаксации задачи ЦЛП с последующим округлением оптимального дробного решения до целочисленного.

К сожалению, как показано в [13], задача (1)–(4) имеет неограниченный разрыв целочисленности, что верно и для задачи ЦЛП(p). Например, пусть есть одно задание, которое имеет одинаковую длительность s

на каждом сервере, и s серверов. В этом случае оптимальное решение релаксированной задачи имеет длину 1, а оптимальное решение целочисленной — s.

Чтобы обойти эту трудность, используем параметрическое сокращение. Пусть $T \in \mathbb{Z}^+$ — верхняя оценка длины расписания. Будем считать, что задание t не может выполняться на сервере s, если $w'_{ts} > T$. Пусть $S_T = \{(t,s) \mid w'_{ts} \leqslant T\}$. Тогда для фиксированных p и T будем искать решение следующей задачи:

$$C_{\max} \to \min, \tag{6}$$

$$\sum_{t\colon (t,s)\in S_T} x_{ts} w'_{ts} + L_s^{\text{init}} \leqslant C_{\max}, \quad s \in S, \tag{7}$$

$$\sum_{s\colon (t,s)\in S_T} x_{ts} = 1, \quad t \in J, \tag{8}$$

$$\sum_{t=1}^{|J|} \sum_{s=1}^{|S|} x_{ts}(1-\lambda_{ts}) \leqslant p, \tag{9}$$

$$x_{ts} \geqslant 0, \quad (t,s) \in S_T, \tag{10}$$

$$C_{\max} \leqslant T. \tag{11}$$

Задачу (6)–(11) обозначим через $\mathrm{LP}(T,p)$. Используя бинарный поиск, можно найти наименьшее значение параметра T, при котором существует допустимое решение. Пусть T^* и есть такое значение. Оно является нижней оценкой оптимального решения задачи Π_p. Найти допустимое решение можно, например, с помощью алгоритма Хачияна [14], который находит экстремальное решение задачи линейного программирования.

2.2. Свойства экстремальных решений. Пусть $|S_T| = m$, $|J| = n$. Определим, какими свойствами обладают экстремальные решения.

Лемма 1. *Любое экстремальное решение* $\mathrm{LP}(T,p)$ *содержит не более* $n+m+1$ *ненулевых переменных.*

Доказательство. Заметим, что решение экстремально, если $|S_T|+1$ линейно независимых ограничений обращаются в равенство. По крайней мере $|S_T|+1-(n+m+2)$ из них будут выбраны из четвёртого множества ограничений. Значит, $|S_T|+1-(n+m+2)$ переменных равны нулю. Следовательно, экстремальное решение имеет не более $n+m+1$ ненулевых переменных. Лемма 1 доказана.

Пусть x^* — экстремальное решение $\mathrm{LP}(T,p)$. Представим его двудольным графом $H = ((J,S),E)$ с множеством вершин $J \cup S$, при этом ребро

$(t, s) \in E$, если $x^*_{ts} > 0$. Далее вершины в доле S будем называть *вершинами-серверами*, а вершины в доле J — *вершинами-заданиями*.

Лемма 2. *Пусть n_c — число вершин-заданий, m_c — число вершин-серверов в некоторой компоненте связности C графа H. Тогда число рёбер в C не превышает $n_c + m_c + 1$.*

Доказательство. Пусть $S_c \subset S$ и $J_c \subset J$ — подмножества вершин-заданий и вершин-серверов, попавших в компоненту связности C. Положим $y_{ts} = x^*_{ts}$, $i \in J_c$, $j \in S_c$. Тогда y — экстремальное решение задачи $LP_C(T)$, ограниченной множествами S_c и J_c. По лемме 1 число рёбер в C не превышает $n_c + m_c + 1$. Лемма 2 доказана.

Если в решении x^* переменная x^*_{ts} равна 1 для некоторого s, то задание t называется *целым*, иначе — *дробным*. Соответствующие им вершины также будем называть *целыми* или *дробными*. Пусть задание t целое и $x^*_{ts} = 1$. Положим $A(t) = s$, т. е. назначим целое задание на ту же машину, что и в решении x^*.

Каждая вершина графа H, соответствующая целому заданию, инцидентна ровно одному ребру. Удалим из графа H все целые вершины-задания с инцидентными им рёбрами. Назовём полученный граф H'. Так как каждая компонента связности C' в графе H' получена из соответственной компоненты C связности графа H удалением одинакового числа вершин и рёбер, утверждение леммы 2 остаётся в силе и для компоненты C'.

Ребра (t, s) графа H' будем называть *лёгкими*, если $1 - x_{ts} \leqslant 1/2$, иначе — *тяжёлыми*. Заметим, что каждая вершина-задание имеет не более одного инцидентного тяжёлого ребра. Сначала докажем две вспомогательные леммы.

Лемма 3. *Пусть C — произвольный цикл в графе H'. Тогда либо тяжёлые и лёгкие рёбра в нём чередуются, либо существует вершина-сервер с двумя лёгкими рёбрами.*

Доказательство. По построению граф H двудольный, следовательно, все циклы в нём содержат чётное число вершин и рёбер. Предположим, что в цикле C нет вершины-сервера с двумя лёгкими рёбрами. Напомним, что каждая вершина-задание содержит по крайней мере одно лёгкое ребро. Занумеруем все рёбра по циклу в направлении от вершины-задания к вершине-серверу, начиная с этого ребра. Тогда по предположению второе ребро, идущее от вершины-сервера, будет тяжёлым, а следующее за ним ребро, идущее от следующей вершины-задания, опять лёгким. В итоге получим, что все нечётные рёбра лёгкие, а чётные — тяжёлые. Лемма 3 доказана.

Лемма 4. *Пусть C — произвольная цепь в графе H' с чётным числом рёбер. Пусть крайние вершины цепи — вершины-серверы, а крайние рёбра тяжёлые. Тогда существует вершина-сервер с двумя лёгкими рёбрами.*

Доказательство. Так как вершины-задания не крайние, каждому из них инцидентны два ребра, и у них нет общих инцидентных рёбер. Поскольку каждая вершина-задание имеет не более одного инцидентного тяжёлого ребра, тяжёлых рёбер в C не больше, чем лёгких рёбер. Пусть не существует вершины-сервера с двумя лёгкими рёбрами. Тогда у каждой вершины-сервера кроме крайних одно ребро тяжёлое и одно лёгкое, т. е. количество тяжёлых и лёгких внутренних рёбер совпадает. Добавляя к количеству тяжёлых рёбер два крайних ребра, получаем противоречие с утверждением, что тяжёлых рёбер в C не больше, чем лёгких. Лемма 4 доказана.

Покажем, что для всех s существует назначение дробных заданий такое, что

$$\sum_{t\colon A(t)=s} (1 - x_{ts}) \leqslant 1. \tag{12}$$

Назовём такое назначение *равномерным*.

Лемма 5. *Пусть x^* — экстремальное решение $\mathrm{LP}(T, p)$. Тогда существует равномерное назначение дробных заданий.*

Доказательство. Достаточно показать, что на каждый сервер можно назначить не более одного дробного задания, связанного с сервером тяжёлым ребром, или не более двух заданий, связанных с сервером лёгкими рёбрами.

Рассмотрим произвольную связную непустую компоненту $C \subseteq H'$. Пусть r — число вершин в компоненте C. Лемма 2 влечёт, что число рёбер компоненты C ограничено числом $r + 1$. Следовательно, компонента C является либо деревом, либо псевдодеревом, либо графом с $r+1$ рёбрами. В последнем случае каждый такой граф может быть получен добавлением двух рёбер к некоторому остовному дереву на вершинах из C. После добавления первого ребра образуется псевдодерево, т. е. связный граф с одним циклом. Добавление второго ребра либо породит новый цикл, либо соединит путём две вершины первого цикла. Граф на r вершинах назовём *двуциклическим*, если он содержит $r+1$ рёбер и в нём нет вершин степени 1. Варианты таких графов представлены на рис. 1.

Построение равномерного назначения разобьём на несколько этапов.

Этап 1. Если в графе H' есть вершина степени 1, то это вершина-сервер, скажем, s. Тогда в решении x^* существует ровно одно дробное

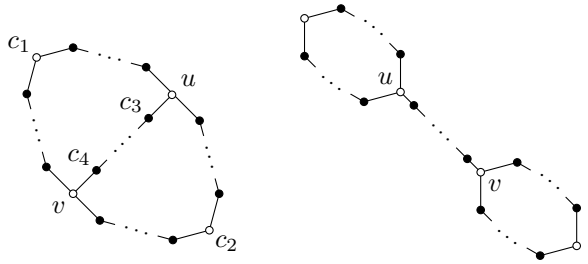


Рис. 1. Виды компонент связности графа H':
вершины-задания выделены чёрным, вершины-серверы — белым

задание t такое, что $0 < x^*_{ts} < 1$. Назначим это задание на сервер s и удалим из H' вершину-сервер, вершину-задание и все инцидентные ей рёбра. Повторим эту процедуру, пока в графе H' не останется вершин степени 1. Назовём полученный граф H''.

При удалении вершин и рёбер не образуется новых циклов. Следовательно, каждая связная компонента графа H'' либо простой цикл, либо двуциклический граф.

Этап 2. Если в результате удаления вершин степени 1 получился простой цикл, то выберем в нём произвольное совершенное паросочетание M и назначим каждое задание на тот сервер, который связан с ним ребром в M.

Осталось разобрать случай, когда после удаления вершин степени 1 получился двуциклический граф. Обозначим через u и v вершины степени больше чем 2 и назовём их *выделенными*. Отметим, что возможна ситуация, когда вершины u и v совпадают. В этом случае имеем одну вершину степени 4, скажем, v. Если выделенные вершины различны, то степень каждой из них равна 3.

Этап 3. Пусть одна из этих вершин (или обе) является вершиной-заданием. Тогда рассмотрим произвольный цикл, в который она входит. Найдём в этом цикле совершенное паросочетание M и назначим каждое задание на тот сервер, который связан с ним ребром в M. После удаления всех вершин цикла и инцидентных им рёбер получим компоненту, которая является либо деревом, либо псевдодеревом. Последовательно повторяя этапы 1 и 2, получим требуемое назначение.

Пусть обе вершины будут вершинами-серверами. Назовём вершину-сервер *особенной*, если ей инцидентно два лёгких ребра.

Этап 4. Предположим, что в двуциклическом графе вершины u и v лежат в разных циклах (правый граф на рис. 1) и в нём нет особенной вершины. Тогда по лемме 3 тяжёлые и лёгкие рёбра в циклах чередуются, следовательно, в каждом цикле существует паросочетание, состоящее

из лёгких рёбер. Назначим задания на серверы согласно двум таким паросочетаниям. Рассмотрим вершины задания на u-v-пути. По лемме 4 одно из крайних рёбер в этом пути лёгкое. Выберем совершенное паросочетание, содержащее это ребро, и назначим задания на серверы согласно этому паросочетанию.

Покажем, что если в двуциклическом графе вершины u и v лежат в одном цикле (левый граф на рис. 1), то всегда существует особенная вершина. Пусть это не так. Рассмотрим произвольный цикл C, в котором лежат вершины u и v. По предположению в нём нет особенной вершины. Тогда по лемме 3 тяжёлые и лёгкие рёбра в цикле C чередуются. Следовательно, в этом цикле в каждую вершину входят одно лёгкое и одно тяжёлое ребро. Однако, вершина v имеет степень три и входит в три цикла. При этом каждая пара рёбер, инцидентных вершине v, попадает в один из трёх циклов. Следовательно, существует цикл, в котором тяжёлые и лёгкие рёбра не чередуются, и согласно лемме 3 получаем противоречие с предположением, что в рассматриваемом графе нет особенной вершины.

Этап 5. Пусть в двуциклическом графе есть особенная вершина z, отличная от u и v. Рассмотрим путь из вершины z в одну из выделенных вершин, который не содержит вторую выделенную вершину. Пусть это будет путь $P_{[zu]}$ из вершины z в вершину u. Так как u и z — вершины-серверы, в $P_{[zu]}$ чётное число рёбер, следовательно, в нём существует два совершенных паросочетания. Выберем то паросочетание, которое содержит лёгкое ребро, инцидентное z. Назначим задания на серверы согласно этому паросочетанию и удалим все назначенные вершины-задания и инцидентные им рёбра. В результате получится либо псевдограф, либо две отдельные компоненты — псевдограф и простой цикл. Последовательно повторяя этапы 1 и 2, получим требуемое назначение.

Этап 6. Пусть в двуциклическом графе вершины u и v лежат в разных циклах и нет особенных вершин, отличных от u и v. По леммам 3 и 4 тяжёлые и лёгкие рёбра чередуются в каждом из циклов и на пути, соединяющем вершины u и v. Следовательно, в каждом цикле и пути есть совершенное паросочетание, состоящее из лёгких рёбер. Назначим задания на серверы согласно этим паросочетаниям. При этом на одну из выделенных вершин придётся два лёгких ребра, а все остальные вершины-серверы получат по одному лёгкому ребру.

Этап 7. Пусть в двуциклическом графе вершины u и v лежат в одном цикле, u — особенная вершина и нет особенных вершин, отличных от u и v. Так как u — особенная вершина, существует цикл C, в котором вершине u инцидентно два лёгких ребра. Пусть P_1 и P_2 — два пути из вершины v в вершину u в этом цикле. Оба пути имеют чётное число

рёбер. Выберем в каждом из них паросочетание, которое содержит ребро, инцидентное вершине u. Назначим задания на серверы согласно этим паросочетаниям. Пусть P_3 — путь из вершины v в вершину u, не лежащий в цикле C. Так как вершины u и v — вершины-серверы, в P_3 чётное число рёбер, следовательно, в нём существует два совершенных паросочетания. Выберем то паросочетание, которое не содержит ребра, инцидентного u. Назначим задания на серверы согласно этому паросочетанию.

В итоге получаем, что по экстремальному решению x^* на каждый сервер можно назначить не более одного дробного задания, связанного с сервером тяжёлым ребром, или не более двух заданий, связанных с сервером лёгкими рёбрами. Лемма 5 доказана.

Положим

$$\alpha = \frac{1}{m}\sum_{t\in J}\min_{s\in S} w_{ts}, \quad \beta = \max_{s\in S}\Big\{\sum_{t\colon (t,s)\in E} w_{ts} + L_s^{\text{init}}\Big\}.$$

Легко проверить, что α и β являются нижней и верхней оценкой длины оптимального расписания соответственно.

Алгоритм 1.

Вход: J, S, G, w_{ij}, f.

1: **for** $p \in [0, n]$ **do**
2: используя бинарный поиск, найти наименьшее значение $T \in [\alpha, \beta]$, для которого $\mathrm{LP}(T, p)$ имеет допустимое решение;
3: обозначить найденное значение через $T^*(p)$;
4: найти экстремальное решение задачи $\mathrm{LP}(T^*(p), p)$;
5: все целые работы назначить по найденному решению;
6: построить равномерное назначение дробных работ;
7: обозначить полученное назначение через A_p;
8: **return** $A^* = \{A_r \mid C_{\max}(A_r) = \min\limits_{p\in\overline{1,n}} C_{\max}(A_p)\}$;

Пусть

$$W(p) = \max_{s,t} w_{ts} + f(p). \tag{13}$$

Обозначим через OPT значение целевой функции в оптимальном решении.

Теорема 1. *Пусть число удалённых заданий в оптимальном решении равно p. Тогда $C_{\max}(A^*) \leqslant \min\{\mathrm{OPT} + W(p), 2\mathrm{OPT}\}$.*

Доказательство. Рассмотрим назначение работ A_p, найденное алгоритмом. Заметим, что если для фиксированного p существует решение x задачи $\mathrm{LP}(T', p)$ такое, что $C_{\max}(x) < T'$, то оно также является

допустимым решением задачи $\mathrm{LP}(T,p)$ с $T = C_{\max}(x)$. Следовательно, согласно шагу 2 алгоритма $T^*(p) \leqslant \mathrm{OPT}$.

Оценим нагрузку каждого сервера s в решении A_p. Пусть x^* — экстремальное решение $\mathrm{LP}(T^*(p),p)$. Имеем

$$L_s^{A_p} = L_s^{\mathrm{init}} + \sum_{t\colon A_p(t)=s} w'_{ts} = \\ = L_s^{\mathrm{init}} + \sum_{t\colon A_p(t)=s} w'_{ts}x^*_{ts} + \sum_{t\colon A_p(t)=s} w'_{ts}(1-x^*_{ts}) \leqslant \\ \leqslant T^*(p) + \sum_{t\colon A_p(t)=s} w'_{ts}(1-x^*_{ts}). \quad (14)$$

Последнее неравенство следует из ограничений (1) и (5) задачи $\mathrm{LP}(T,p)$. Так как на сервер s могут быть назначены только задания из множества $S_{T^*(p)}$, для любого задания t такого, что $A_p(t) = s$, имеем $w'_{ts} \leqslant T^*(p)$. Подставляя это неравенство в (14) и учитывая (12) и (13), получим

$$L_s^{A_p} \leqslant T^*(p) + \min\{T^*(p), W(p)\}.$$

Поскольку последнее неравенство выполнено для нагрузки каждого сервера, имеем

$$C_{\max}(A^*) \leqslant C_{\max}(A_p) \leqslant T^*(p) + \min\{T^*(p), W(p)\} \leqslant \\ \leqslant \mathrm{OPT} + \min\{\mathrm{OPT}, W(p)\},$$

откуда следует утверждение теоремы. Теорема 1 доказана.

Замечание 1. Отметим, что первый результат теоремы 1 обобщает результат из [4], полученный для случая, когда $w_{ts} = 1$ для всех s и t.

Трудоёмкость алгоритма определяется трудоёмкостью решения задачи линейного программирования на каждой итерации бинарного поиска для каждого возможного значения параметра p. Пусть $O(\tau)$ — трудоёмкость решения $\mathrm{LP}(T,p)$. Тогда алгоритм 1 имеет трудоёмкость $O(\tau n \times \log(\beta-\alpha))$, которая ограничена полиномом от размера входа задачи.

2.3. Модификации алгоритма при решении тестовых примеров. В предыдущем разделе было доказано, что алгоритм 1 строит расписание, длина которого не превышает двух длин оптимального расписания, и время работы алгоритма ограничено полиномом от размера входа индивидуальной задачи. Однако с практической точки зрения алгоритм 1 имеет два недостатка. Во-первых, он универсален и не учитывает специфику конкретного примера. Во-вторых, время его работы достаточно велико.

В алгоритме 1 после решения задачи линейного программирования происходит округление дробного решения до целого. Дробное решение представляется двудольным графом. Округление производится с помощью разбора висячих вершин. В тестовых примерах часто возникает ситуация, когда вершина-задание смежна с несколькими висячими вершинами-серверами. В этом случае выберем из них сервер с минимальной нагрузкой. Эта простая модификация не улучшает теоретическую оценку точности алгоритма, но приводит к улучшению решений для большинства примеров. Назовём алгоритм 1 с такой модификацией алгоритмом 2.

Более существенный недостаток алгоритма 1 — его трудоёмкость. Перебор решений для каждого значения параметра p требует много времени. В целях ускорения алгоритма было решено заменить перебор количества удалённых заданий жадной балансировкой нагрузки. На первой стадии решается задача $\mathrm{LP}(T, 0)$, т. е. строится дробное решение, в котором нет удалённых заданий. Далее по дробному решению одним из ранее описанных методов строится целочисленное решение. Затем применяется процедура локального спуска. Процедура состоит в перемещении задания с самого нагруженного сервера на любой менее нагруженный, которое не увеличивает длины расписания. Такую модификацию назовём ускоренным алгоритмом 2.

Эффект от обеих модификаций обсуждается в разд. 3.

3. Вычислительные эксперименты

В этом разделе представлены результаты вычислительных экспериментов для оценки эффективности представленного алгоритма.

Во всех тестовых примерах число серверов равняется 20, а число заданий изменяется от 60 до 200. Предполагается, что начальная нагрузка серверов равна нулю. Каждое задание выступает локальным ровно для трёх серверов. Для каждого задания случайно генерируем числа из усечённого нормального распределения со средним 10 и дисперсией 4. Числа округляем вниз до целого и применяем в качестве номеров серверов для размещения блоков данных. Выбор нормального распределения для размещения данных обусловлен необходимостью получить неравномерную загрузку серверов. В случае равномерной загрузки серверов данными задача сильно упрощается и, как правило, все применяемые далее алгоритмы находят оптимальное решение.

Рассматриваются два класса примеров: с одинаковыми локальными длительностями и с произвольными локальными длительностями. В первом случае все локальные длительности равны 20, длительность удалённых заданий вычисляется как $w'_{ij} = w_{ij} + C \cdot r$, где C — фактор сети, а r — число удалённых заданий в назначении. В построенных примерах

фактор сети изменяется от 0,1 до 3. Для примеров с произвольными длительностями значения параметров w_{ij} генерируются на отрезке $[1, 50]$ с равномерным распределением.

Алгоритмы для сравнения реализованы в виде компьютерной программы на языке C++ и решателя SCIP 8.0.3 под лицензией Apache 2.0 [15]. Расчёты проведены на персональном компьютере с процессором Intel (R) Core (TM) i5 9400 2,9 ГГц и 8 ГБ ОЗУ.

3.1. Сравнение методов округления дробного решения. В п. 2.3 описана модификация округления дробного решения. Как видно из графиков, представленных на рис. 2 и 3, за счёт более аккуратного назначения дробных заданий алгоритм 2 улучшает решения в среднем на 3% для задач с одинаковыми локальными длительностями работ и на 6% — для задач с произвольными локальными длительностями работ.

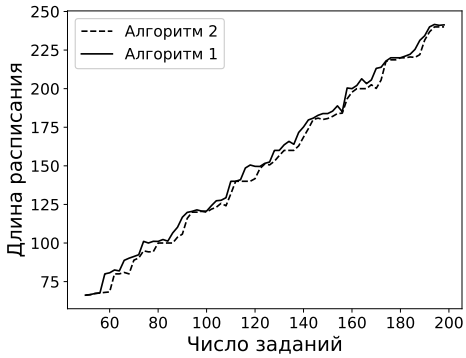


Рис. 2. Сравнение решений, полученных алгоритмами 1 и 2: одинаковые длительности

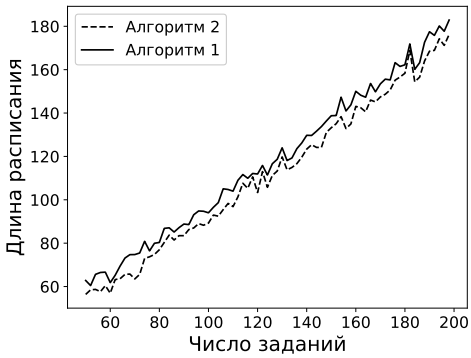


Рис. 3. Сравнение решений, полученных алгоритмами 1 и 2: произвольные длительности

3.2. Сравнение эффективности алгоритмов. Так как для рассматриваемой задачи в литературе известны алгоритмы только для примеров с единичными длительностями работ, проведём сравнение алгоритмов, представленных в этой работе, со стандартным алгоритмом системы Hadoop.

Стандартный алгоритм системы Hadoop (Hadoop Default Scheduler, HDS) [16] работает в режиме онлайн. Когда сервер простаивает, алгоритм выбирает локальное задание. Если такого нет, то случайное. Для сравнения с остальными алгоритмами была реализована оффлайн версия этого алгоритма. Для назначения очередного задания алгоритм выбирает наименее загруженный сервер. Затем нагрузки обновляются.

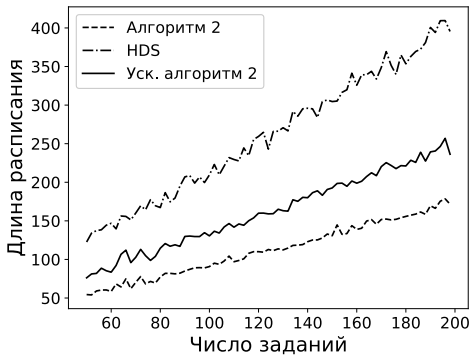


Рис. 4. Сравнение точности решений, $C = 0{,}1$

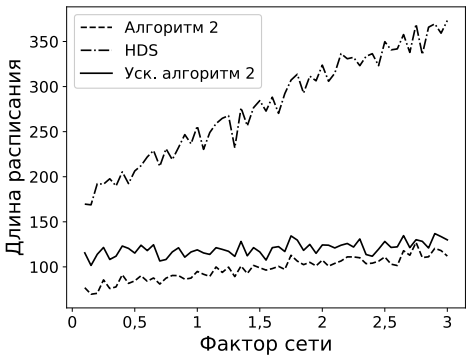


Рис. 5. Влияние фактора сети на точность различных алгоритмов

На рис. 4 приведено сравнение алгоритма 2 и его ускоренной версии с алгоритмом HDS. На диаграмме хорошо видно, что алгоритм HDS сильно проигрывает в точности даже ускоренной версии алгоритма 2.

Чтобы исследовать влияние перегруженности сети, сравниваем точность алгоритмов, изменяя сетевой коэффициент C от 0,1 до 3,0. Число работ в этих тестовых примерах равнялось 80. На рис. 7 видно, что увеличение коэффициента C приводит приводит к увеличению разрыва между точностью решений, полученных алгоритмом HDS и алгоритмом 2, в то время как разница целевых значений, полученных алгоритмом 2 и его ускоренной версией, снижается.

Для полноты картины сравним алгоритмы 1 и 2 со специализированными алгоритмами, разработанными для примеров с единичными длительностями работ.

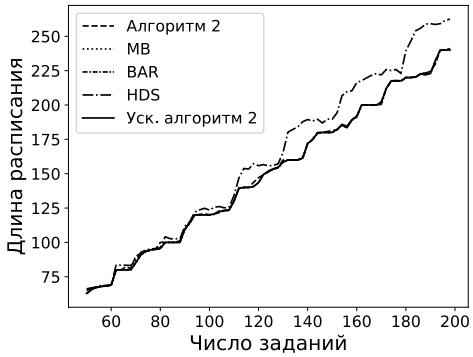


Рис. 6. Сравнение точности решений: одинаковые длительности

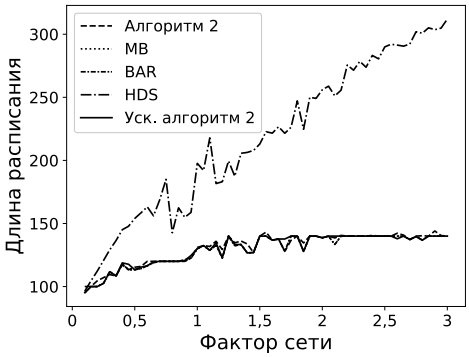


Рис. 7. Влияние фактора сети на точность различных алгоритмов: одинаковые длительности

Алгоритм Balance-Reduce (BAR) [5] стартует с решения задачи о максимальном потоке для назначения локальных заданий. Затем с помощью локальной процедуры Reduce решение улучшается перемещением заданий с наиболее загруженных серверов на менее загруженные. Алгоритм применим только для случая одинаковых локальных длительностей.

Алгоритм MaxCover-BalAssign (MB) [4] работает итеративно, создавая назначения, и выбирает из них наилучшее с точки зрения длины расписания. Каждая итерация состоит из двух частей. На первом этапе фиксируется ограничение на количество локальных заданий для серверов и решается задача о максимальном потоке. На втором этапе неназначенные задания распределяются жадной эвристикой. Алгоритм применим только для случая одинаковых локальных длительностей.

Проведённые эксперименты показывают, что алгоритм 2 и его ускоренная версия для примеров с одинаковой локальной длительностью заданий не уступают в точности алгоритмам HDS и BAR. Однако необходимо признать, что алгоритм 2 и даже его ускоренная версия сильно проигрывают потоковым алгоритмам в скорости нахождения решений.

Заключение

В настоящей работе рассмотрена задача распределения заданий по вычислительным серверам, в которой время обработки задания зависит как от нагрузки сети при передачи исходных данных, так и от машины, на которую назначается задание. Для рассматриваемой задачи предложен 2-приближённый алгоритм полиномиальной трудоёмкости, основанный на решении соответствующей задачи линейного программирования и последующем округлении полученного решения. Проведённые вычислительные эксперименты показали, что алгоритм сопоставим по качеству получаемых решений с известными специализированными алгоритмами, даже если локальные длительности всех работ одинаковы. Так как время работы алгоритма значительно проигрывает времени работы алгоритмов, основанных на решении задачи о максимальном потоке, предложена и протестирована ускоренная версия алгоритма.

Финансирование работы

Работа первого автора выполнена в рамках подготовки ВКР магистра под руководством второго автора в Новосибирском гос. университете. Работа второго автора выполнена в рамках гос. задания Института математики им. С. Л. Соболева (проект № FWNF–2022–0019). Дополнительных грантов на проведение или руководство этим исследованием получено не было.

Конфликт интересов

Авторы заявляют, что у них нет конфликта интересов.

Литература

1. **Garey M. R., Johnson D. S.** "Strong" NP-completeness results: Motivation, examples, and implication // J. ACM. 1978. V. 25, No. 3. P. 499–508.
2. **Potts C. N.** Analysis of a linear programming heuristic for scheduling unrelated parallel machines // Discrete Appl. Math. 1985. V. 10, No. 2. P. 155–164.
3. **Lenstra J. K., Shmoys D. B., Tardos É.** Approximation algorithms for scheduling unrelated parallel machines // Math. Program. 1990. V. 46. P. 259–271.
4. **Fischer M. J., Su X., Yin Y.** Assigning tasks for efficiency in Hadoop: Extended abstract // Proc. 22nd Annu. ACM Symp. Parallelism in Algorithms and Architectures (Thira Santorini, Greece, June 13–15, 2010). New York: ACM, 2010. P. 30–39. DOI: 10.1145/1810479.1810484.
5. **Jin J., Luo J., Song A., Dong F., Xiong R.** BAR: An efficient data locality driven task scheduling algorithm for cloud computing // Proc. 11th IEEE/ACM Int. Symp. Cluster, Cloud and Grid Computing (Newport Beach, USA, May 23–26, 2011). Washington: IEEE Comput. Soc., 2011. P. 295–304. DOI: 10.1109/CCGrid.2011.55.
6. **Kwon Y., Balazinska M., Howe B., Rolia J.** A study of skew in MapReduce applications // Proc. Open Cirrus Summit 2011 (Moscow, Russia, June 1–3, 2011). Piscataway: IEEE, 2011. P. 1–5.
7. **Brin S., Page L.** The anatomy of a large-scale hypertextual Web search engine // Comput. Netw. ISDN Syst. 1998. V. 30, No. 1–7. P. 107–117.
8. **Gates A. F., Natkovich O., Chorpa S.** [et al.]. Building a high-level dataflow system on top of Map-Reduce: The Pig experience // Proc. VLDB Endow. 2009. V. 2, No. 2. P. 1414–1425.
9. **Moseley B., Dasgupta A., Kumar R., Sarlós T.** On scheduling in map-reduce and flow-shops // Proc. 23rd Annu. ACM Symp. Parallelism in Algorithms and Architectures (San Jose, CA, USA, June 4–6, 2011). New York: ACM, 2011. P. 289–298. DOI: 10.1145/1989493.1989540.
10. **Dean J., Ghemawat S.** MapReduce: Simplified data processing on large clusters // Commun. ACM. 2008. V. 51, No. 1. P. 107–113. DOI: 10.1145/1327452.1327492.
11. **Танаев В. С., Гордон В. С., Шафранский Я. М.** Теория расписаний: Одностадийные системы. М.: Наука, 1984. 384 с.
12. **Гэри М., Джонсон Д.** Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982. 416 с.
13. **Vazirani V. V.** Approximation algorithms. Heidelberg: Springer, 2001. 396 p.
14. **Хачиян Л. Г.** Полиномиальные алгоритмы в линейном программировании // Журн. вычисл. математики и мат. физики. 1980. Т. 20, № 1. С. 51–68.
15. **Bestuzheva K., Besançon M., Chen W.-K.** [et al.]. Enabling research through the SCIP Optimization Suite 8.0 // ACM Trans. Math. Softw. 2023. V. 49, No. 2. Article ID 22. 21 p. DOI: 10.1145/3585516.

16. White T. Hadoop: The definitive guide. Sebastopol, CA: O'Reilly Media, 2015. 756 p.

Демаков Алексей Владимирович
Кононов Александр Вениаминович

Статья поступила
22 октября 2024 г.
После доработки —
20 августа 2025 г.
Принята к публикации
22 сентября 2025 г.

DISKRETNYI ANALIZ I ISSLEDOVANIE OPERATSII
/DISCRETE ANALYSIS AND OPERATIONS RESEARCH/
October–December 2025. Vol. 32, No. 4. P. 118–137

UDC 519.8 DOI: 10.33048/daio.2025.32.816

AN APPROXIMATE ALGORITHM FOR TASK ASSIGNMENT TO HETEROGENEOUS PROCESSORS WITH DELAYS IN DATA TRANSMISSION

A. V. Demakov[1, a] and *A. V. Kononov*[2, b]

[1] Novosibirsk State University,
2 Pirogov Street, 630090 Novosibirsk, Russia
[2] Sobolev Institute of Mathematics,
4 Koptyug Avenue, 630090 Novosibirsk, Russia
E-mail: [a] a.demakov@g.nsu.ru, [b] alvenko@math.nsc.ru

Abstract. We consider a problem of assigning tasks to computing servers taking into account the initial download of data for their execution. Assigning a task to a server that does not contain a required data block will take time to pass the block over the network. The more blocks are transmitted over the network, the more time is added to the task. The total time of all tasks is to be minimized.

For the problem under consideration, we propose a 2-approximate algorithm. Using a solution of some linear programming problem, this algorithm completes a fractional solution to an integer one. For computational experiments an enhanced version of the algorithm is considered. The computational experiments showed that the algorithm is comparable with some known algorithms in the quality of solution obtained. Illustr. 7, bibliogr. 16.

Keywords: scheduling theory, approximate algorithm, makespan.

References

1. **S. Brin** and **L. Page,** The anatomy of a large-scale hypertextual Web search engine, *Comput. Netw. ISDN Syst.* **30** (1–7), 107–117 (1998).
2. **M. R. Garey** and **D. S. Johnson,** "Strong" NP-completeness results: Motivation, examples, and implication, *J. ACM* **25** (3), 499–508 (1978).
3. **A. F. Gates, O. Natkovich, S. Chorpa,** [et al.], Building a high-level dataflow system on top of Map-Reduce: The Pig experience, *Proc. VLDB Endow.* **2** (2), 1414–1425 (2009).

English transl.: *Journal of Applied and Industrial Mathematics* **19** (4) (2025).

4. **J. Dean** and **S. Ghemawat,** MapReduce: Simplified data processing on large clusters, *Commun. ACM* **51** (1), 107–113 (2008), DOI: 10.1145/1327452.1327492.
5. **M. J. Fischer, X. Su,** and **Y. Yin,** Assigning tasks for efficiency in Hadoop: Extended abstract, in *Proc. 22nd Annu. ACM Symp. Parallelism in Algorithms and Architectures* (Thira Santorini, Greece, June 13–15, 2010) (ACM, New York, 2010), pp. 30–39, DOI: 10.1145/1810479.1810484.
6. **J. Jin, J. Luo, A. Song, F. Dong,** and **R. Xiong,** BAR: An efficient data locality driven task scheduling algorithm for cloud computing, in *Proc. 11th IEEE/ACM Int. Symp. Cluster, Cloud and Grid Computing* (Newport Beach, USA, May 23–26, 2011) (IEEE Comput. Soc., Washington, 2011), pp. 295–304, DOI: 10.1109/CCGrid.2011.55.
7. **Y. Kwon, M. Balazinska, B. Howe,** and **J. Rolia,** A study of skew in MapReduce applications, in *Proc. Open Cirrus Summit 2011* (Moscow, Russia, June 1–3, 2011) (IEEE, Piscataway, 2011), pp. 1–5.
8. **B. Moseley, A. Dasgupta, R. Kumar,** and **T. Sarlós,** On scheduling in map-reduce and flow-shops, in *Proc. 23rd Annu. ACM Symp. Parallelism in Algorithms and Architectures* (San Jose, CA, USA, June 4–6, 2011) (ACM, New York, 2011), pp. 289–298, DOI: 10.1145/1989493.1989540.
9. **J. K. Lenstra, D. B. Shmoys,** and **É. Tardos,** Approximation algorithms for scheduling unrelated parallel machines, *Math. Program.* **46**, 259–271 (1990).
10. **C. N. Potts,** Analysis of a linear programming heuristic for scheduling unrelated parallel machines, *Discrete Appl. Math.* **10** (2), 155–164 (1985).
11. **V. S. Tanaev, V. S. Gordon,** and **Ya. M. Shafranskii,** *Scheduling Theory: Single-Stage Systems* (Nauka, Moscow, 1984) [Russian].
12. **M. R. Garey** and **D. S. Johnson,** *Computers and Intractability. A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, 1979; Mir, Moscow, 1982 [Russian]).
13. **V. V. Vazirani,** *Approximation Algorithms* (Springer, Heidelberg, 2001).
14. **L. G. Khachiyan,** Polynomial algorithms in linear programming, *Zh. Vychisl. Mat. Mat. Fiz.* **20** (1), 51–68 (1980) [Russian] [*USSR Comput. Math. Math. Phys.* **20** (1), 53–72 (1980), DOI: 10.1016/0041-5553(80)90061-0].
15. **K. Bestuzheva, M. Besançon, W.-K. Chen,** [et al.], Enabling research through the SCIP Optimization Suite 8.0, *ACM Trans. Math. Softw.* **49** (2), ID 22 (2023), DOI: 10.1145/3585516.
16. **T. White,** *Hadoop: The Definitive Guide* (O'Reilly Media, Sebastopol, CA, 2015).

Aleksey V. Demakov
Aleksandr V. Kononov

Received October 22, 2024
Revised August 20, 2025
Accepted September 22, 2025