УДК 004.942 **DOI: 10.14529/cmse250302**

МЕТОД ПРЕДСТАВЛЕНИЯ ТРЕХ WORK-STEALING ДЕКОВ В ДВУХУРОВНЕВОЙ ПАМЯТИ

© 2025 Е.А. Аксёнова[1], Е.А. Барковский, А.В. Соколов[1]

[1] *Институт прикладных математических исследований Карельского научного центра Российской академии наук*

(185910 Петрозаводск, ул. Пушкинская, д. 11)

E-mail: aksenova@krc.karelia.ru, barkevgen@gmail.com, avs@krc.karelia.ru

Поступила в редакцию: 21.07.2025

Эффективность работы распределенных систем во многом зависит от равномерной загруженности потоков, что обеспечивается за счет различных стратегий балансировки задач между ними. Одним из методов децентрализованной динамической стратегии балансировки, где каждый поток участвует в распределении задач, является «work-stealing». Принцип метода в том, что поток не имеющий задач перехватывает («steal») их у других потоков. В основе процесса лежит специальная структура данных, work-stealing дек, в котором находятся указатели на задачи. При работе с деками возникает проблема их эффективного расположения в памяти. В двухуровневой памяти дек можно расположить разделив его на концы и середину: в быстрой памяти (первый уровень) находятся вершина и основание дека; в медленной памяти (второй уровень) — середина. Таким образом, система может быстро обращаться к концам дека, где элементы активно добавляются и удаляются. В статье анализируется новый метод представления трех work-stealing деков в двухуровневой памяти, где концы деков находятся в отдельных участках памяти. Для него строится имитационная модель на основе метода Монте-Карло, с помощью которой исследуются задачи оптимального разделения памяти. В качестве критерия оптимальности используется максимальное среднее время работы системы до перераспределения памяти.

Ключевые слова: двухуровневая память, метод Монте-Карло, структуры данных, work-stealing деки.

ОБРАЗЕЦ ЦИТИРОВАНИЯ

Аксёнова Е.А., Барковский Е.А., Соколов А.В. Метод представления трех work-stealing деков в двухуровневой памяти // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2025. Т. 14, № 3. С. 28–41. DOI: 10.14529/cmse250302.

Введение

Эффективность работы распределенных систем во многом зависит от равномерной загруженности потоков. Ярким примером этого являются параллельные [1, 2] и облачные [3, 4] вычисления, где равномерная нагрузка потоков обеспечивается за счет различных стратегий балансировки задач между ними. Для задач, характеристики которых заранее известны, используют стратегию статической балансировки. Если характеристики задачи не известны или могут меняться (например, система работает в реальном времени), используют стратегию динамической балансировки [5]. Такие стратегии бывают централизованными (распределением задач занят специальный поток) и децентрализованными (каждый поток участвует в распределении задач) [6].

Work-stealing является распространенным методом децентрализованной динамической стратегии балансировки задач [7, 8] и реализован в широко используемых балансировщиках: Cilk; dotNET TPL; Intel TBB; X10. Принцип метода в том, что поток без задач перехватывает («steal») их у других потоков [9]. Для этого используется специальная структура данных, work-stealing дек, в котором находятся указатели на задачи.

В целом, work-stealing дек является деком («double ended queue» или «deque») с ограниченным входом [10]: в один конец элементы добавляются и удаляются по принципу LIFO (Last In, First Out) или «последним пришел, первым вышел», из другого конца только удаляются по принципу FIFO (First In, First Out) — «первым пришел, первым вышел». Каждый поток имеет свой work-stealing дек. Таким образом, когда создается новая задача, указатель помещается (операция «push») на вершину дека. Когда поток готов выполнить задачу, он берет (операция «pop») указатель на нее из вершины своего дека. Когда поток готов выполнить задачу, но его дек пуст, он перехватывает (операция «steal») указатель из основания другого дека. На рис. 1 представлена схема работы такого дека.

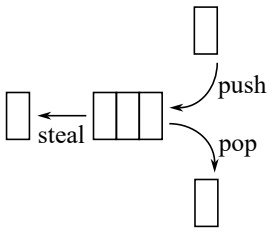

Рис. 1. Принцип работы work-stealing дека

Для эффективного применения стратегии децентрализованной динамической балансировки задач, исследуются как способы реализации метода work-stealing [11, 12], так и способы организации work-stealing деков [13, 14]: в [15] исследуются различные способы представления деков в памяти; в [16, 17] описывается разработка и проводится анализ динамического балансировщика, построенного на основе этого метода.

В этой работе рассматривается следующий способ расположения дека в памяти:

- В быстрой памяти (первый уровень) находятся вершина и основание;
- В медленной памяти (второй уровень) находится середина.

Таким образом, система может быстро обращаться к концам дека, где элементы активно добавляются и удаляются. Для этого способа представления дека возникают задачи разделения и перераспределения памяти.

Модель работы такого work-stealing дека была предложена в [18, 19]. В статьях была решена задача перераспределения двухуровневой памяти для критерия оптимальности: минимизация средних затрат на перераспределение. Модели работы и методы управления двумя work-stealing деками в двухуровневой памяти были рассмотрены в [20]. Модель работы трех work-stealing деков в двухуровневой памяти была предложена в [21]. В предыдущей статье была решена задача разделения памяти для метода представления, где концы деков растут на встречу друг другу в общем участке памяти. В этой статье предлагается модель и исследуется задача разделения двухуровневой памяти для альтернативного метода представления трех work-stealing деков, где концы деков находятся в отдельных участках памяти.

Статья организована следующим образом. В разделе 1 дается описание нового метода и ставятся задачи разделения памяти между тремя деками. В разделе 2 рассматривается имитационная модель процесса и приводится пример ее работы. Результаты исследований, проведенных на основе модели, представлены в разделе 3. В заключении предложены некоторые варианты использования метода и направление дальнейших исследований.

1. Постановка задачи

Пусть в памяти компьютерной системы расположены три work-stealing дека De_1, De_2 и De_3. Память системы разбита на два уровня имеющие разную скорость: память первого уровня быстрая, второго — медленная. Чтобы расположить дек в двухуровневой памяти, он разделен на три части (рис. 2):

1. конец Sk_n, куда указатели на задачи добавляются и удаляются;
2. середина Md_n, где указатели хранятся;
3. конец Qe_n, из которого указатели только удаляются (перехватываются методом work-stealing).

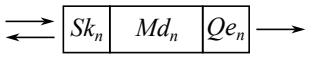

Рис. 2. Разбиение work-stealing дека De_n

Середины трех деков (Md_1, Md_2, Md_3) находятся в памяти второго уровня. В памяти первого уровня находятся концы этих деков: активные (Sk_1, Sk_2, Sk_3) и концы из которых происходит перехват (Qe_1, Qe_2, Qe_3), объединенные в одну очередь Qe. Таким образом, двухуровневая память компьютерной системы, в которой расположены три work-stealing дека, имеет следующий вид (рис. 3):

- m — общий размер быстрой памяти первого уровня;
- s — участок памяти первого уровня, где находятся объединенная очередь Qe и активный конец первого дека Sk_1, представленный в виде стека;
- d и $m - s - d$ — участки памяти, в которых расположены активные концы Sk_2 и Sk_3 соответственно.

Характеристики памяти второго уровня и находящихся в ней частей деков Md_1, Md_2 и Md_3 в этой статье не рассматриваются.

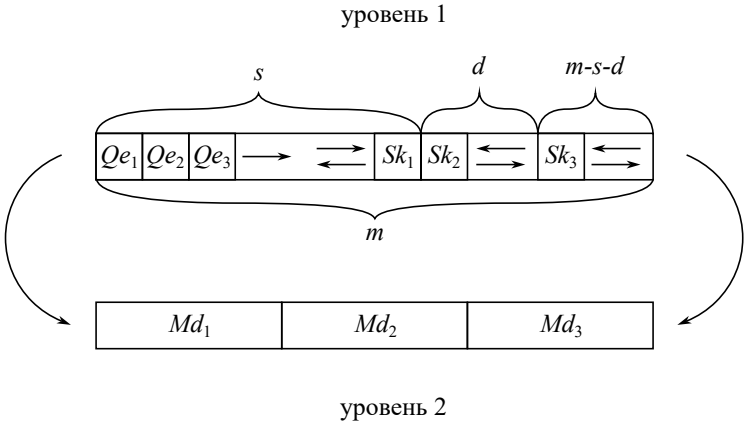

Рис. 3. Расположение деков в двухуровневой памяти

Пусть один указатель на задачу занимает одну единицу памяти. Весь размер быстрой памяти составляет m указателей, размеры концов каждого дека равны между собой $Sk_1 = Qe_1 = Sk_2 = Qe_2 = Sk_3 = Qe_3$. Система работает в дискретном времени, где на каждом шаге происходят операции влияющие на размер деков. Вероятности этих операций для каждого из деков указаны в табл. 1 ($p_n + q_n + w_n + pw_n + qw_n + r_n = 1$).

Таблица 1. Вероятности операций с деком De_n

Обозначение	Описание вероятности
p_n	Добавление одного указателя в рабочий конец Sk_n дека De_n
q_n	Удаление одного указателя из рабочего конца Sk_n дека De_n
w_n	Перехват одного указателя из общей work-stealing очереди Qe, в то время как дек De_n обрабатывает задачу
pw_n	Параллельное добавление указателя в Sk_n и перехват указателя из Qe
qw_n	Параллельное удаление указателя из Sk_n и перехват указателя из Qe
r_n	Операция не изменяющая размеры концов дека De_n (обработка задачи)

Время работы системы заранее не ограничено, но она прекращает работу если возникает одно из условий перераспределения памяти: дек De_n пытается забрать указатель из уже пустого конца Sk_n ($Sk_n < 0$); общая очередь work-stealing концов деков опустошена $Qe < 0$; один из участков памяти первого уровня переполнен $Qe + Sk_1 > s$, $Sk_2 > d$ или $Sk_3 > m - s - d$.

Для вышеописанной системы рассматриваются следующие задачи:

1. нахождение оптимального разделения s памяти первого уровня m, если участкам d и $m - s - d$ выделена половина оставшейся памяти ($d = m - s - d = (m - s)/2$);
2. нахождение оптимальных значений s и d разделения памяти между тремя деками.

Критерий оптимальности: максимальное среднее время работы системы (среднее количество операций) до перераспределения памяти.

2. Имитационная модель

Поставленные задачи были решены с помощью имитационного моделирования. Для корректной работы этой модели, основанной на методе Монте-Карло, некоторые из характеристик системы должны быть заранее известны. Так, значения вероятностей операций с деками должны быть получены в ходе предварительных статистических исследований. Размер общей памяти m фиксирован, в то время как значение разделения s всей памяти m и значение разделения d участка памяти $m - s$ вычисляются в ходе работы модели. На каждом шаге дискретного времени для каждого дека генерируется случайное значение от нуля до единицы, соответствующее одной из вероятности операций с ним. Среднее количество шагов, которые модель проделывает до перераспределения памяти, является средним временем работы t такой системы. Комбинации s и d при которых t принимает наибольшее значение, считаются оптимальными значениями разделения памяти первого уровня m.

Для примера, рассмотрим систему с характеристиками $m = 100$, $s = 50$, $d = 25$, $m - s - d = 25$ и значениями вероятностей $p_n = q_n = w_n = pw_n = qw_n = 0.16$ и $r_n = 0.20$ (рис. 4).

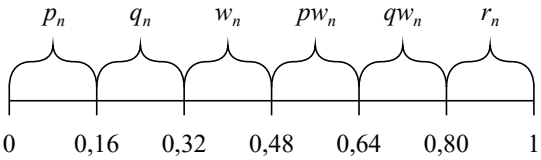

Рис. 4. Пример разбиения отрезка от нуля до одного между вероятностями операций с деками

Допустим, на шаге моделирования n эта система имеет следующий вид:

- $Qe = 1$;
- $Sk_1 = 10$;
- $Sk_2 = 0$;
- $Sk_3 = 25$.

Затем генерируется случайные значения, на основе которых совершаются операции с деками. Например: для дека De_1 было получено число 0.24, соответствующие вероятности q_1 удаления элемента из Sk_1; для дека De_2 число 0.08, что соответствует вероятности p_2 добавления элемента в Sk_2; и для дека De_3 число 0.72 и вероятность qw_3 — операция одновременного удаления элемента из Sk_3 и перехвата элемента из Qe.

Таким образом, на шаге $n + 1$ рабочие концы деков принимают следующие размеры:

- $Qe = 1 - 1 = 0$;
- $Sk_1 = 10 - 1 = 9$;
- $Sk_2 = 0 + 1 = 1$;
- $Sk_3 = 25 - 1 = 24$.

Так как переполнение памяти ($Qe + Sk_1 \not> 50$, $Sk_2 \not> 25$, $Sk_3 \not> 25$) или опустошение концов деков ($Qe \not< 0$, $Sk_1 \not< 0$, $Sk_2 \not< 0$, $Sk_3 \not< 0$) не произошло, то система продолжает свою работу. Полученные размеры рабочих концов становятся стартовыми размерами для следующего шага моделирования.

Рассмотрим ситуации, при которых такая система прекратила бы свою работу на шаге моделирования n:

1. Для дека De_2 сгенерировано случайное значение, соответствующее вероятности q_2, что приводит к опустошению рабочего конца дека ($Sk_2 = 0 - 1 = -1$, $Sk_2 < 0$);
2. Для дека De_3 получено число, соответствующие вероятности p_3 — память, отведенная рабочему концу дека, переполняется ($Sk_3 = 25 + 1 = 26$, $Sk_3 > 25$);
3. Произошло две подряд операции перехвата элемента (w_n, pw_n и/или qw_n) из общей очереди work-stealing концов деков, что приводит к ее опустошению ($Qe = 1-1-1 = -1$, $Qe < 0$).

В этой статье моделируется система, где размер общей памяти первого уровня равен $m = 100$ единиц и начальные размеры концов деков составляют $Sk_1 = Qe_1 = Sk_2 = Qe_2 = Sk_3 = Qe_3 = 10$ единиц памяти. Таким образом, участки памяти могут иметь следующие размеры: участок, в котором расположены конец Sk_1 и очередь work-stealing концов Qe — $s = 40 \dots 80$ единиц; участок, в котором расположен конец Sk_2 — $d = 10 \dots 50$ единиц и, соответственно, участок для конца Sk_3, $m - s - d = 10 \dots 50$ единиц.

3. Численный анализ

В результате работы имитационной модели были получены оптимальные значения разделения памяти (s, d) и среднее время работы системы (t) для различный вероятностей операций с деками. Вероятности имеют теоретические значения для более легкого сравнения результатов.

В следующих таблицах приведены результаты для задачи нахождения оптимального разделения s памяти m, где участки памяти d и $m - s - d$ имеют одинаковый размер ($d = m - s - d = (m - s)/2$). В табл. 2 рассматривается система, где указатели чаще добавляются в первый дек ($p_1 > p_2$, $p_1 > p_3$); в табл. 3 — чаще во второй ($p_2 > p_1$, $p_2 > p_3$). Система, в которой указатели с большей вероятностью добавляются в два дека ($p_1 > p_3$,

$p_2 > p_3$ и $p_2 > p_1$, $p_3 > p_1$), анализируется в табл. 4 и 5. Система, где указатели добавляются равномерно в три дека ($p_1 = p_2 = p_3$) рассматривается в табл. 6.

Таблица 2. Среднее время работы системы при $d = (m - s)/2$, $p_1 > p_2$ и $p_1 > p_3$

Вероятности операций	Метод разделения памяти	
	Разделение пополам ($s = 50$, $d = 25$)	Оптимальное разделение ($s = 66$, $d = 17$)
$p_1 = 0.50$, $p_2 = p_3 = 0.26$, $q_1 = 0.02$, $q_2 = q_3 = 0.26$, $w_1 = w_2 = w_3 = 0.01$, $pw_1 = pw_2 = pw_3 = 0.01$, $qw_1 = qw_2 = qw_3 = 0.01$, $r_1 = r_2 = r_3 = 0.45$	27.82	56.59
$p_1 = 0.60$, $p_2 = p_3 = 0.31$	22.23	46.11
$p_1 = 0.70$, $p_2 = p_3 = 0.36$	18.50	38.91
$p_1 = 0.80$, $p_2 = p_3 = 0.41$	15.83	33.66
$p_1 = 0.90$, $p_2 = p_3 = 0.46$	13.83	29.69

Таблица 3. Среднее время работы системы при $d = (m - s)/2$, $p_2 > p_1$ и $p_2 > p_3$

Вероятности операций	Метод разделения памяти	
	Разделение пополам ($s = 50$, $d = 25$)	Оптимальное разделение ($s = 46$, $d = 27$)
$p_2 = 0.50$, $p_1 = p_3 = 0.26$, $q_2 = 0.02$, $q_1 = q_3 = 0.26$, $r_1 = r_2 = r_3 = 0.45$	32.78	36.73
$p_2 = 0.60$, $p_1 = p_3 = 0.31$	27.18	30.31
$p_2 = 0.70$, $p_1 = p_3 = 0.36$	23.23	25.85
$p_2 = 0.80$, $p_1 = p_3 = 0.41$	20.31	22.54
$p_2 = 0.90$, $p_1 = p_3 = 0.46$	18.05	19.98

В целом, оптимальные значения разделения памяти s для каждого набора вероятностей совпадают. Для примера, опишем первый результат из табл. 2. Вероятность добавления указателя в активный конец Sk_1 дека De_1 составляет $p_1 = 0.50$. Вероятность того, что указатель будет добавлен в конец (Sk_2, Sk_3) другого дека (De_2, De_3) равна $p_2 = p_3 = 0.26$. Если память первого уровня ($m = 100$) разделена пополам, то участкам памяти выделены следующие размеры: участок, в котором расположены Sk_1 и Qe, $s = 50$ единиц; участок, в котором расположен Sk_2, $d = 25$ единиц; участок, в котором расположен Sk_3, $m - s - d = 25$ единиц. Если память разделена оптимально, то размеры участков равны $s = 66$, $d = 17$ и $m - s - d = 17$. Среднее время работы системы до перераспределения памяти первого уровня, где память разделена пополам — $t = 27.82$ операций; системы, где память разделена оптимально — $t = 56.59$ операций. Таким образом, система с оптимально разделенной памятью совершает на 28.77 операций больше, чем система у которой память разделена пополам.

Таблица 4. Среднее время работы системы при $d = (m - s)/2$, $p_1 > p_3$ и $p_2 > p_3$

Вероятности операций	Метод разделения памяти	
	Разделение пополам ($s = 50$, $d = 25$)	Оптимальное разделение ($s = 54$, $d = 23$)
$p_1 = p_2 = 0.50$, $p_3 = 0.26$, $q_1 = q_2 = 0.02$, $q_3 = 0.26$, $r_1 = r_2 = r_3 = 0.45$	25.78	27.89
$p_1 = p_2 = 0.60$, $p_3 = 0.31$	21.08	23.17
$p_1 = p_2 = 0.70$, $p_3 = 0.36$	17.84	19.89
$p_1 = p_2 = 0.80$, $p_3 = 0.41$	15.48	17.48
$p_1 = p_2 = 0.90$, $p_3 = 0.46$	13.66	15.66

Таблица 5. Среднее время работы системы при $d = (m - s)/2$, $p_2 > p_1$ и $p_3 > p_1$

Вероятности операций	Метод разделения памяти	
	Разделение пополам ($s = 50$, $d = 25$)	Оптимальное разделение ($s = 45$, $d = 27$)
$p_2 = p_3 = 0.50$, $p_1 = 0.26$, $q_2 = q_3 = 0.02$, $q_1 = 0.26$, $r_1 = r_2 = r_3 = 0.45$	29.44	33.43
$p_2 = p_3 = 0.60$, $p_1 = 0.31$	24.68	27.91
$p_2 = p_3 = 0.70$, $p_1 = 0.36$	21.34	24.04
$p_2 = p_3 = 0.80$, $p_1 = 0.41$	18.90	21.19
$p_2 = p_3 = 0.90$, $p_1 = 0.46$	17.07	19.04

Таблица 6. Среднее время работы системы при $d = (m - s)/2$, $p_1 = p_2 = p_3$

Вероятности операций	Метод разделения памяти	
	Разделение пополам ($s = 50$, $d = 25$)	Оптимальное разделение ($s = 52$, $d = 24$)
$p_1 = p_2 = p_3 = 0.50$, $q_1 = q_2 = q_3 = 0.02$, $r_1 = r_2 = r_3 = 0.45$	24.87	25.97
$p_1 = p_2 = p_3 = 0.60$	20.55	21.74
$p_1 = p_2 = p_3 = 0.70$	17.54	18.81
$p_1 = p_2 = p_3 = 0.80$	15.32	16.69
$p_1 = p_2 = p_3 = 0.90$	13.60	15.12

В табл. 7 и 8 приведены результаты для задачи нахождения оптимальных значений разделения памяти s и d. В этих таблицах рассматриваются только системы с большой вероятность добавления указателя во второй дек ($p_2 > p_1$, $p_2 > p_3$ и $p_1 > p_3$, $p_2 > p_3$): оптимальное разделение s, разделение d и время работы t для других наборов вероятностей

совпадает с результатами в предыдущих таблицах. Это можно объяснить тем, что значение разделения памяти d зависит от вероятностных характеристик дека De_2 и не имеет влияния, если вероятность добавления указателя во второй дек маленькая.

Таблица 7. Среднее время работы системы при $d = max$, $p_2 > p_1$ и $p_2 > p_3$

Вероятности операций	Размер участка памяти $m - s$ ($s = 46$)	
	Половина участка ($d = 27$)	Оптимальный размер ($d = 37$)
$p_2 = 0.50$, $p_1 = p_3 = 0.26$, $q_2 = 0.02$, $q_1 = q_3 = 0.26$, $w_1 = w_2 = w_3 = 0.01$, $pw_1 = pw_2 = pw_3 = 0.01$, $qw_1 = qw_2 = qw_3 = 0.01$, $r_1 = r_2 = r_3 = 0.45$	36.73	51.92
$p_2 = 0.60$, $p_1 = p_3 = 0.31$	30.31	43.02
$p_2 = 0.70$, $p_1 = p_3 = 0.36$	25.85	36.72
$p_2 = 0.80$, $p_1 = p_3 = 0.41$	22.54	32.03
$p_2 = 0.90$, $p_1 = p_3 = 0.46$	19.98	28.42

Таблица 8. Среднее время работы системы при $d = max$, $p_1 > p_3$ и $p_2 > p_3$

Вероятности операций	Размер участка памяти $m - s$ ($s = 54$)	
	Половина участка ($d = 23$)	Оптимальный размер ($d = 29$)
$p_1 = p_2 = 0.50$, $p_3 = 0.26$, $q_1 = q_2 = 0.02$, $q_3 = 0.26$, $r_1 = r_2 = r_3 = 0.45$	27.89	34.03
$p_1 = p_2 = 0.60$, $p_3 = 0.31$	23.17	27.95
$p_1 = p_2 = 0.70$, $p_3 = 0.36$	19.89	23.74
$p_1 = p_2 = 0.80$, $p_3 = 0.41$	17.48	20.67
$p_1 = p_2 = 0.90$, $p_3 = 0.46$	15.66	18.31

Аналогично с предыдущей задачей, система с оптимально разделенной памятью работает дольше. Например, проанализируем первый результат из табл. 7. Если участок памяти первого уровня $m - s = 54$ разделен пополам ($d = m - s - d = 27$), то система совершает в среднем $t = 36.73$ операций до перераспределения памяти. Если память разделена оптимально: участку, в котором расположены Sk_1 и Qe выделяется $s = 46$ единиц; участку, в котором расположен Sk_2 — $d = 37$ единиц; участку, в котором находится Sk_3 выделяется $m - s - d = 17$ единиц. Среднее время работы такой системы составляет $t = 51.92$ операций, т.е. она работает в среднем на 15.19 операций дольше. Зависимость среднего времени работы этой системы t от разделения d участка памяти $m - s$ представлена в виде графика на рис. 5.

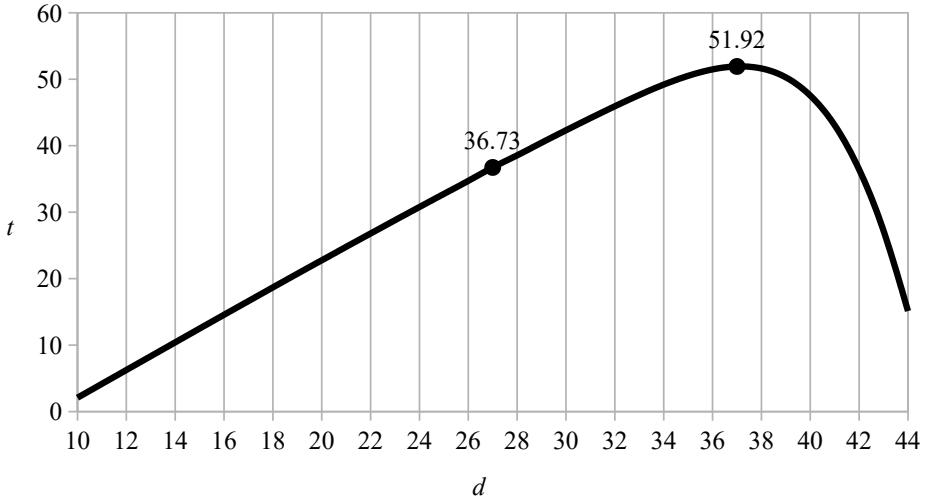

Рис. 5. Зависимость времени работы t от разделения памяти d

Заключение

В статье описывается один из методов представления трех work-stealing деков в двухуровневой памяти, где концы деков расположены в отдельных участках памяти. На основе этого метода была построена имитационная модель и решены задачи оптимального разделения памяти первого уровня между тремя деками для различных теоретических компьютерных систем. В качестве критерия оптимальности используется максимальное среднее время работы системы до перераспределения памяти.

Системы, где участки памяти разделены оптимально, работают дольше систем, память которых разделена поровну. Так, в зависимости от вероятностных характеристик, среднее время работы может увеличится на треть или в два раза.

Задача разделения памяти между деками может возникнуть во время разработки системного программного обеспечения. В таком случае, на основе предложенной модели и предварительного статистического исследования системы, можно найти оптимальные значения разделения быстрой памяти первого уровня.

В будущем можно провести новые исследования не только для других наборов входных данных, но и для других критериев оптимальности, например, минимизация затрат на перераспределение памяти. Также стоит заметить, что в этом методе концы деков расположены в отдельных участках памяти — возможно увеличить число структур данных не изменяя количества участков: концы новых деков следует располагать в участках памяти вместе с уже существующими, пустив их на встречу друг другу.

Литература

1. Dinu S., Raicu G. Load Balancing in Parallel Computing: An Evolutionary Approach // Advanced Topics in Optoelectronics, Microelectronics, and Nanotechnologies XI. Vol. 12493 / ed. by M. Vladescu, R.D. Tamas, I. Cristea. International Society for Optics, Photonics. SPIE, 2023. P. 124931M. DOI: 10.1117/12.2643056.

2. Alam M., Varshney A.K. A New Approach of Dynamic Load Balancing Scheduling Algorithm for Homogeneous Multiprocessor System // International Journal of Applied Evolutionary Computation. 2016. Vol. 7, no. 2. P. 61–75. DOI: 10.4018/IJAEC.2016040104.

3. Shafiq D., Jhanjhi N., Abdullah A. Load Balancing Techniques in Cloud Computing Environment: A Review // Journal of King Saud University – Computer and Information Sciences. 2021. Vol. 34. DOI: 10.1016/j.jksuci.2021.02.007.

4. Амелина Н.О., Корнивец А.Д., Иванский Ю.В., Тюшев К.И. Применение консенсусного протокола для балансировки загрузки стохастической децентрализованной сети при передаче данных // XII Всероссийское совещание по проблемам управления: Труды научной конференции, Москва, 16–19 июня 2014. Москва: ИПУ РАН, 2014. С. 8902–8911.

5. Alakeel A.M. A Guide to Dynamic Load Balancing in Distributed Computer Systems // International Journal of Computer Science and Network Security. 2010. Vol. 10, no. 6. P. 153–160.

6. Xia Y., Prasanna V.K., Li J. Hierarchical Scheduling of DAG Structured Computations on Manycore Processors with Dynamic Thread Grouping // Job Scheduling Strategies for Parallel Processing / ed. by E. Frachtenberg, U. Schwiegelshohn. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. P. 154–174. DOI: 10.1007/978-3-642-16505-4_9.

7. Yang J., He Q. Scheduling Parallel Computations by Work Stealing: A Survey // International Journal of Parallel Programming. 2018. Vol. 46, no. 2. P. 173–197. DOI: 10.1007/s10766-016-0484-8.

8. Hwu W.-m. GPU Computing Gems, Jade Edition. 2012. DOI: 10.1016/C2010-0-68654-8.

9. Arora N.S., Blumofe R.D., Plaxton C.G. Thread Scheduling for Multiprogrammed Multiprocessors // Proceedings of the Tenth Annual ACM Symposium on Parallel Algorithms and Architectures. Puerto Vallarta, Mexico: Association for Computing Machinery, 1998. P. 119–129. SPAA '98. DOI: 10.1145/277651.277678.

10. Knuth D.E. The Art of Computer Programming, Volume 1 (3rd ed.): Fundamental Algorithms. USA: Addison Wesley Longman Publishing Co., Inc., 1997. DOI: 10.5555/260999.

11. Lin C.-X., Huang T.-W., Wong M.D.F. An Efficient Work-Stealing Scheduler for Task Dependency Graph // 2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS). 2020. P. 64–71. DOI: 10.1109/ICPADS51040.2020.00018.

12. Li J., Agrawal K., Elnikety S., *et al.* Work Stealing for Interactive Services to Meet Target Latency // SIGPLAN Notices. New York, NY, USA, 2016. Vol. 51, no. 8. Article 14. DOI: 10.1145/3016078.2851151.

13. Gmys J., Leroy R., Mezmaz M., *et al.* Work Stealing with Private Integer-Vector-Matrix Data Structure for Multi-Core Branch-and-Bound Algorithms // Concurrency and Computation: Practice and Experience. 2016. Vol. 28, no. 18. P. 4463–4484. DOI: 10.1002/cpe.3771.

14. Wimmer M., Versaci F., Traff J.L., *et al.* Data Structures for Task-Based Priority Scheduling // SIGPLAN Notices. New York, NY, USA, 2014. Vol. 49, no. 8. P. 379–380. DOI: 10.1145/2692916.2555278.

15. Аксенова Е.А., Соколов А.В. Методы управления work-stealing деками в динамических планировщиках многопроцессорных параллельных вычислений // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2023. Т. 12, № 4. С. 76–93. DOI: 10.14529/cmse230403.

16. Kuchumov R., Sokolov A., Korkhov V. Staccato: Cache-Aware Work-Stealing Task Scheduler for Shared-Memory Systems // Computational Science and Its Applications – ICCSA 2018 / ed. by O. Gervasi, B. Murgante, S. Misra, *et al.* Cham: Springer International Publishing, 2018. P. 91–102. DOI: 10.1007/978-3-319-95171-3_8.

17. Kuchumov R., Sokolov A., Korkhov V. Staccato: Shared-Memory Work-Stealing Task Scheduler with Cache-Aware Memory Management // International Journal of Web and Grid Services. 2019. Vol. 15, no. 4. P. 394–407. DOI: 10.1504/IJWGS.2019.103233.

18. Аксенова Е.А., Лазутина А.А., Соколов А.В. Минимизация средних затрат на перераспределение при работе с work-stealing деком в двухуровневой памяти // Программные системы: теория и приложения. 2021. Т. 12, № 2. С. 53–71. DOI: 10.25209/2079-3316-2021-12-2-53-71.

19. Лазутина А.А., Соколов А.В. Об оптимальном управлении Work-Stealing деками в двухуровневой памяти // Вестник компьютерных и информационных технологий. 2020. Т. 17, № 4. С. 51–60. DOI: 10.14489/vkit.2020.04.pp.051-060.

20. Aksenova E.A., Lazutina A.A., Sokolov A.V. About Optimal Management of Work-Stealing Deques in Two-Level Memory // Lobachevskii Journal of Mathematics. 2021. Vol. 42, no. 7. P. 1475–1482. DOI: 10.1134/S1995080221070027.

21. Аксенова Е.А., Барковский Е.А., Соколов А.В. Оптимальное управление тремя work-stealing деками в двухуровневой памяти // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2024. Т. 13, № 3. С. 47–60. DOI: 10.14529/cmse240303.

Аксёнова Елена Алексеевна, к.ф.-м.н., лаборатория информационных компьютерных технологий, Институт прикладных математических исследований Карельского научного центра Российской академии наук (Петрозаводск, Российская Федерация)

Барковский Евгений Александрович, независимый исследователь (Петрозаводск, Российская Федерация)

Соколов Андрей Владимирович, д.ф.-м.н., профессор, лаборатория информационных компьютерных технологий, Институт прикладных математических исследований Карельского научного центра Российской академии наук (Петрозаводск, Российская Федерация)

DOI: 10.14529/cmse250302

A REPRESENTATION METHOD OF THREE WORK-STEALING DEQUES IN TWO-LEVEL MEMORY

© 2025 E.A. Aksenova[1], E.A. Barkovsky, A.V. Sokolov[1]

[1] *Institute of Applied Mathematical Research of the Karelian Research Centre of the Russian Academy of Sciences (str. Pushkinskaya 11, Petrozavodsk, 185910 Russia)*

E-mail: aksenova@krc.karelia.ru, barkevgen@gmail.com, avs@krc.karelia.ru

Received: 21.07.2025

The efficiency of distributed systems largely depends on the uniformity of thread workload. To achieve this, various strategies for balancing tasks between threads are utilized. One of the methods of dynamic distributed load balancing, where each thread participates in task distribution, is called "work-stealing." In this method, a thread without tasks steals them from other threads. This process is based on a special data structure, a work-stealing deque, where pointers to tasks are stored. Thus, the problem of efficient placement of deques in memory arises. A deque can be represented in two-level memory in a divided form: the often-accessed ends of a deque are placed in the fast memory of the first level, while its middle is left in the slow memory of the second level. This way, the system can quickly access the ends of the deque where elements are being actively added and removed. In this paper, a new representation method of three work-stealing deques in two-level memory where the ends of the deques are placed in separate memory areas is described. We propose a simulation model of this approach based on the Monte Carlo method and solve several problems of optimal partitioning of memory. The optimality criterion is the maximum mean system operating time to memory reallocation.

Keywords: data structures, Monte Carlo method, two-level memory, work-stealing deques.

FOR CITATION

Aksenova E.A., Barkovsky E.A., Sokolov A.V. A Representation Method of Three Work-Stealing Deques in Two-Level Memory. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2025. Vol. 14, no. 3. P. 28–41. (in Russian) DOI: 10.14529/cmse250302.

References

1. Dinu S., Raicu G. Load Balancing in Parallel Computing: An Evolutionary Approach. Advanced Topics in Optoelectronics, Microelectronics, and Nanotechnologies XI. Vol. 12493 / ed. by M. Vladescu, R.D. Tamas, I. Cristea. International Society for Optics, Photonics. SPIE, 2023. P. 124931M. DOI: 10.1117/12.2643056.
2. Alam M., Varshney A.K. A New Approach of Dynamic Load Balancing Scheduling Algorithm for Homogeneous Multiprocessor System. International Journal of Applied Evolutionary Computation. 2016. Vol. 7, no. 2. P. 61–75. DOI: 10.4018/IJAEC.2016040104.
3. Shafiq D., Jhanjhi N., Abdullah A. Load Balancing Techniques in Cloud Computing Environment: A Review. Journal of King Saud University – Computer and Information Sciences. 2021. Vol. 34. DOI: 10.1016/j.jksuci.2021.02.007.
4. Amelina N.O., Kornivec A.D., Ivanskij J.V., Tjushev K.I. Primenenie konsensusnogo protokola dlja balansirovki zagruzki stohasticheskoj decentralizovannoj seti pri peredache dan-

nyh. XII Vserossijskoe soveshhanie po problemam upravlenija: Scientific Conference Proceedings, Moscow, Russia, June 16–19, 2014. Moscow: ICS of RAS, 2014. P. 8902–8911. (in Russian).

5. Alakeel A.M. A Guide to Dynamic Load Balancing in Distributed Computer Systems. International Journal of Computer Science and Network Security. 2010. Vol. 10, no. 6. P. 153–160.
6. Xia Y., Prasanna V.K., Li J. Hierarchical Scheduling of DAG Structured Computations on Manycore Processors with Dynamic Thread Grouping. Job Scheduling Strategies for Parallel Processing / ed. by E. Frachtenberg, U. Schwiegelshohn. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010. P. 154–174. DOI: 10.1007/978-3-642-16505-4_9.
7. Yang J., He Q. Scheduling Parallel Computations by Work Stealing: A Survey. International Journal of Parallel Programming. 2018. Vol. 46, no. 2. P. 173–197. DOI: 10.1007/s10766-016-0484-8.
8. Hwu W.-m. GPU Computing Gems, Jade Edition. 2012. DOI: 10.1016/C2010-0-68654-8.
9. Arora N.S., Blumofe R.D., Plaxton C.G. Thread Scheduling for Multiprogrammed Multiprocessors. Proceedings of the Tenth Annual ACM Symposium on Parallel Algorithms and Architectures. Puerto Vallarta, Mexico: Association for Computing Machinery, 1998. P. 119–129. SPAA '98. DOI: 10.1145/277651.277678.
10. Knuth D.E. The Art of Computer Programming, Volume 1 (3rd ed.): Fundamental Algorithms. USA: Addison Wesley Longman Publishing Co., Inc., 1997. DOI: 10.5555/260999.
11. Lin C.-X., Huang T.-W., Wong M.D.F. An Efficient Work-Stealing Scheduler for Task Dependency Graph. 2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS). 2020. P. 64–71. DOI: 10.1109/ICPADS51040.2020.00018.
12. Li J., Agrawal K., Elnikety S., *et al.* Work Stealing for Interactive Services to Meet Target Latency. SIGPLAN Notices. New York, NY, USA, 2016. Vol. 51, no. 8. Article 14. DOI: 10.1145/3016078.2851151.
13. Gmys J., Leroy R., Mezmaz M., *et al.* Work Stealing with Private Integer-Vector-Matrix Data Structure for Multi-Core Branch-and-Bound Algorithms. Concurrency and Computation: Practice and Experience. 2016. Vol. 28, no. 18. P. 4463–4484. DOI: 10.1002/cpe.3771.
14. Wimmer M., Versaci F., Traff J.L., *et al.* Data Structures for Task-Based Priority Scheduling. SIGPLAN Notices. New York, NY, USA, 2014. Vol. 49, no. 8. P. 379–380. DOI: 10.1145/2692916.2555278.
15. Aksenova E.A., Sokolov A.V. Control Methods of Work-Stealing Deques in Dynamic Schedulers of Multiprocessor Parallel Computations. Bulletin of the SUSU. Series: Computational Mathematics and Software Engineering. 2023. Vol. 12, no. 4. P. 76–93. (in Russian) DOI: 10.14529/cmse230403.
16. Kuchumov R., Sokolov A., Korkhov V. Staccato: Cache-Aware Work-Stealing Task Scheduler for Shared-Memory Systems. Computational Science and Its Applications – ICCSA 2018 / ed. by O. Gervasi, B. Murgante, S. Misra, *et al.* Cham: Springer International Publishing, 2018. P. 91–102. DOI: 10.1007/978-3-319-95171-3_8.

17. Kuchumov R., Sokolov A., Korkhov V. Staccato: Shared-Memory Work-Stealing Task Scheduler with Cache-Aware Memory Management. International Journal of Web and Grid Services. 2019. Vol. 15, no. 4. P. 394–407. DOI: 10.1504/IJWGS.2019.103233.

18. Aksenova E.A., Lazutina A.A., Sokolov A.V. About Optimal Managment of Work-Stealing Deques in Two-Level Memory. Program Systems: Theory and Applications. 2021. Vol. 12, no. 2. P. 53–71. (in Russian) DOI: 10.25209/2079-3316-2021-12-2-53-71.

19. Lazutina A.A., Sokolov A.V. About Optimal Management of Work-Stealing Deques in Two-Level Memory. Herald of Computer and Information Technologies. 2020. Vol. 17, no. 4. P. 51–60. (in Russian) DOI: 10.14489/vkit.2020.04.pp.051-060.

20. Aksenova E.A., Lazutina A.A., Sokolov A.V. About Optimal Management of Work-Stealing Deques in Two-Level Memory. Lobachevskii Journal of Mathematics. 2021. Vol. 42, no. 7. P. 1475–1482. DOI: 10.1134/S1995080221070027.

21. Aksenova E.A., Barkovsky E.A., Sokolov A.V. Optimal Control of Three Work-Stealing Deques Located in Two-Level Memory. Bulletin of the South Ural State University. Series: Computational Mathematics and Software Engineering. 2024. Vol. 13, no. 3. P. 47–60. (in Russian) DOI: 10.14529/cmse240303.