Modeling of the memory management process for dynamic work-stealing schedulers

Elena A. Aksenova, Andrew V. Sokolov

Institute of Applied Mathematical Research
Karelian Research Centre RAS
Petrozavodsk, Russia

Introduction

Strategies for parallel computing planning are divided into static and dynamic.

Static schedulers are used when almost all the tasks are known. In this case, we can determine in advance the optimal task schedule. Such a problem is considered NP-complete and is quite rare.

When dynamic scheduling the scheduler uses a relatively simple planning strategy that produces a result that is close to optimal.

Introduction

- **Work-sharing** — the scheduler moves jobs from the most loaded processor to the least loaded one.

- **Work-stealing** — processor that has become empty tries to steal jobs from another processor. Work-stealing strategy is implemented in a large number of systems, e.g.: Cilk, TBB, TPL, X10 and others. In this method of load-balancing processor uses deques for information storing.

There are different ways for implementing of several deques in the shared memory:

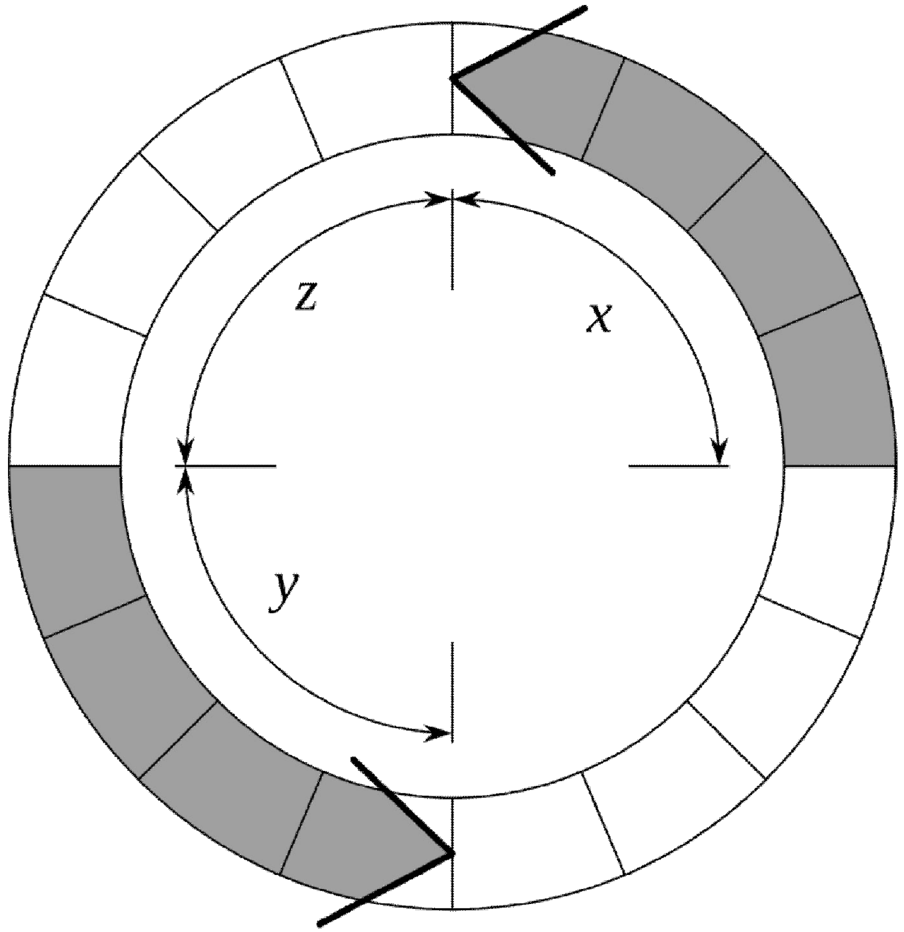
- One can use the “linked list” implementation. A model of this method for deques will be similar to the already constructed models for stacks and queues [*Sokolov A.V., Drac A.V., 2013*].

- In [Aksenova, E.A., Lazutina, A.A., Sokolov, A.V*, 2003*] another method was proposed and analyzed, where stacks and queues are represented as a linked list of arrays (“paged implementation”).

- In [*Hendler D., Lev Y., Moir M., Shavit N., 2006*] it was suggested to use this method for deques.

- In [*Mitzermatchen M., 1998*] the model of work-stealing load balancer (built on the basis of queuing theory) was proposed, but specific ways of representing deques in memory were not considered.

The implementation method of work-stealing deques was proposed, where deques are allocated "one by one in a circle“. (Patent application № 2016143800)

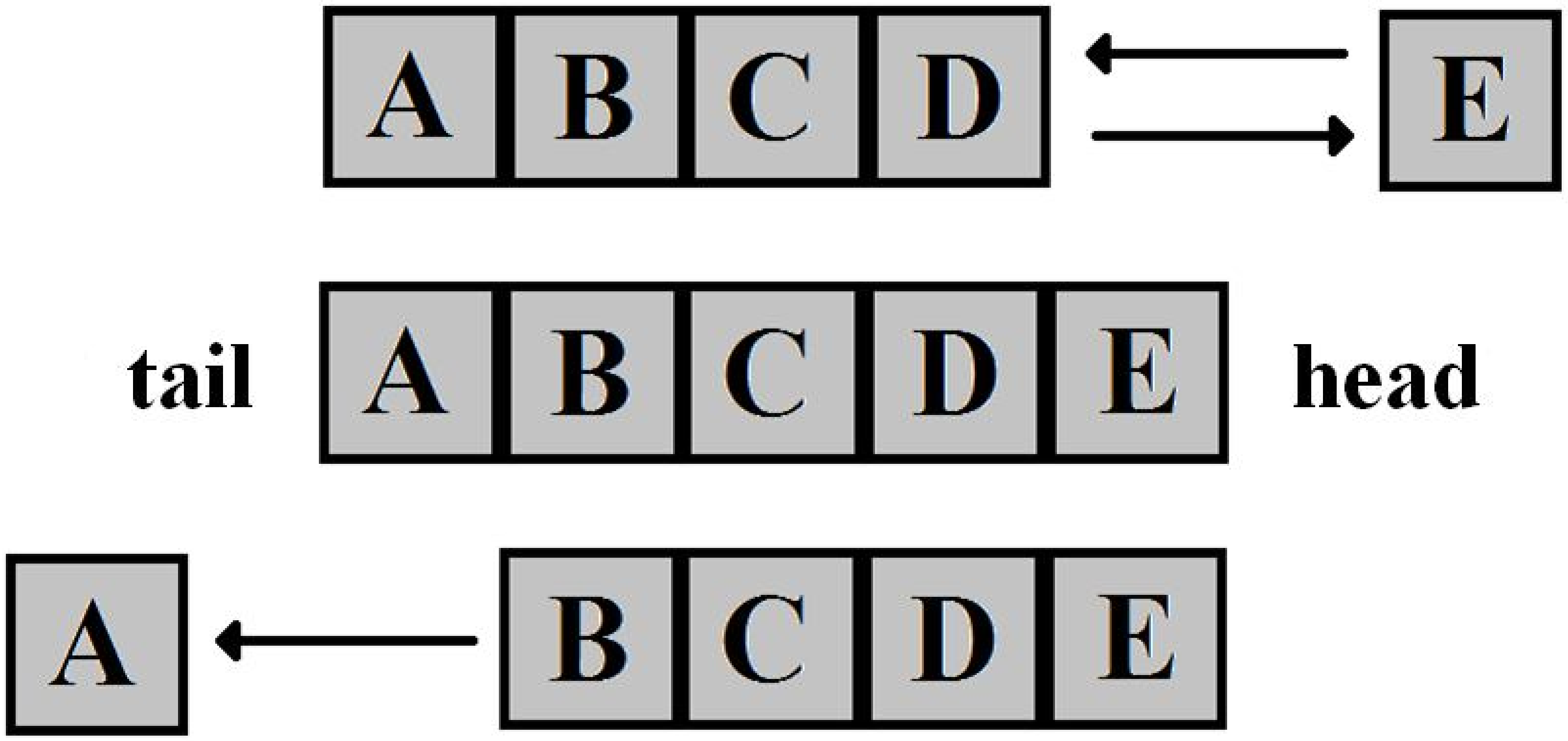


In this paper, we propose a mathematical model for the sequential cyclic representation of n work-stealing deques, where each deque is located in one's memory area.

At each step of discrete time, operations with specified probabilities can occur.

Previously, such models were built by our team to represent some dynamic data structures: stacks, queues, priority queues and others.

Work-stealing deque



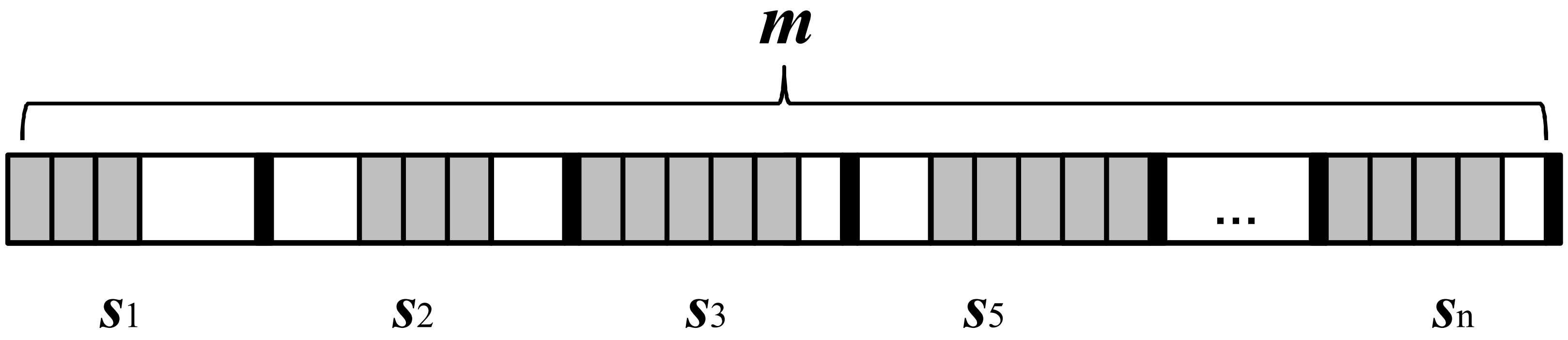
If one processor finds that its deque is empty, it steals tasks from another processors deque.

Problem definition

- m – the size of shared memory;
- n – number of deques, $n<m$;
- s_i – the size of memory for *i*-th deque,

  $s_1+s_2+\ldots+s_n=m, \quad i=1,\ldots,n;$
- x_i – the length of *i*-th deque at a some step,

$$0 \le x_i \le s_i, \quad i=1,\ldots n.$$



At each step, parallel operations of insertion, deletion and processing of elements are possible.

Suppose that the probabilistic characteristics of each deque are known:

- p_i – the probability of insertion an element to the *i*-th deque,
- q_i – is the probability of deletion the element from the *i*-th deque,
- r_i – is the probability that the length of the *i*-th deque has not changed (processing),

$$p_i+q_i+r_i=1,\ \ i=1,\dots,n.$$

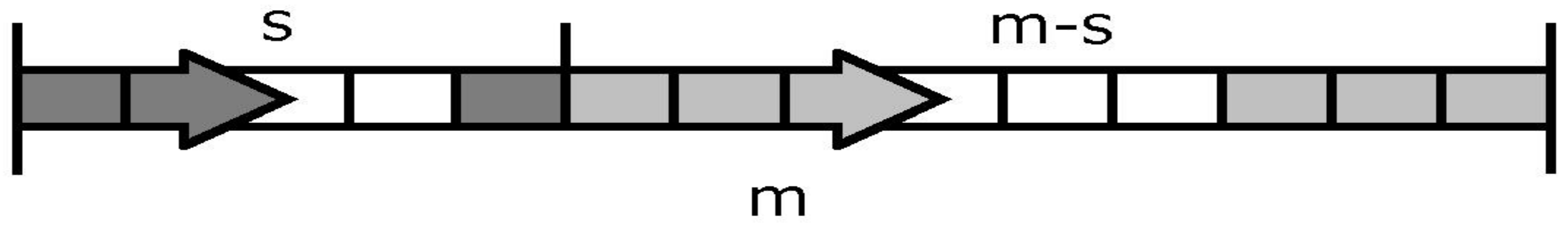
If the *j*-th deque becomes empty $x_j=0$, $j=1,\ldots,n$, then it can steals the $K > 0$ elements from any deque in which $x_i > K$, $i \neq j$, $i=1,\ldots,n$. At each step one deque can steals elements from only one deque.

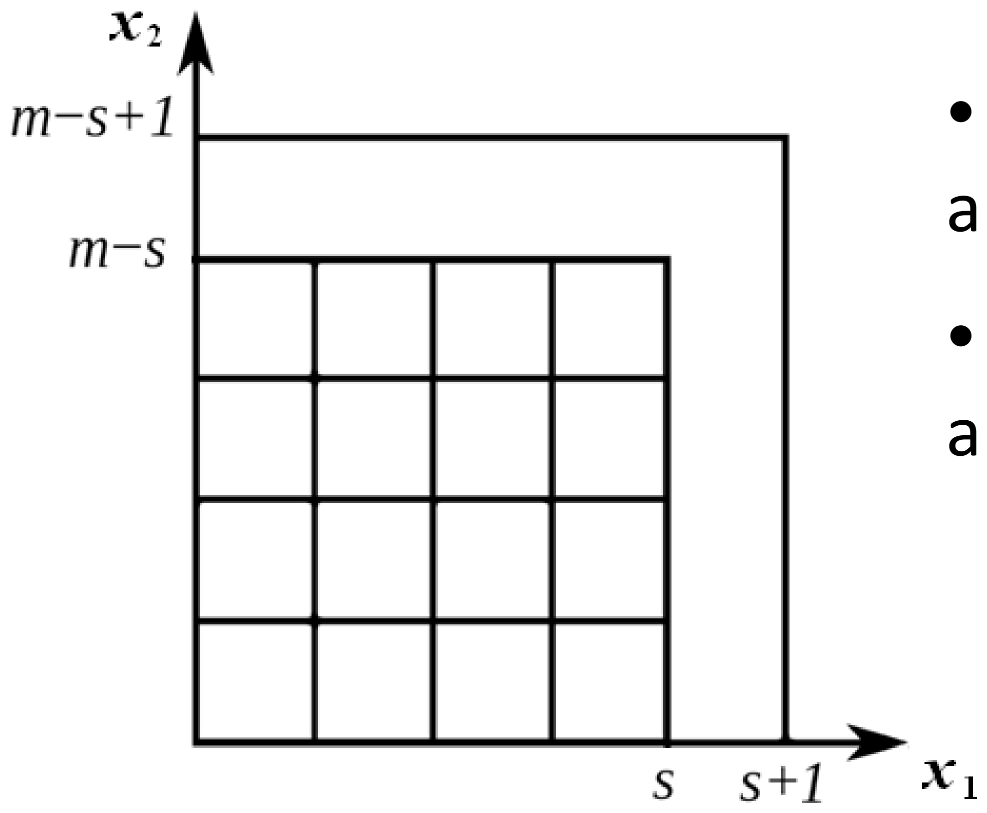
The problem is to choose the values s_i, $\Sigma s_i=m$, $i=1,\ldots,n$ and $K > 0$ so, that the average time before the overflow of any deque would be maximal.

As a mathematical model, we consider a random walk on an integer lattice in the *n*-dimensional space $0 \leq x_i \leq s_i$, $i=1,\ldots,n$, where x_i – is the length of the *i*-th deque. We will examine a random walk as an absorbing Markov chain.

That optimality criterion can be useful, for example, in applications of real time, where the overflow of memory can lead to an emergency shutdown of the program. It is appropriate to note here, that among the multi-core architectures there are such, where cache memory is absent. For example, in the AsAP-II architecture each core has two FIFO-queues, and in the SEAforth architecture (designed by Charles Moore (IntellaSys)) each core has two LIFO-stacks. In these architectures stacks and queues are implemented cyclically and separated from each other with the possibility to lose elements due to overflow. We assume, that it is possible to implement work-stealing deques in hardware in a similar way, and in some cases our model can minimize a number of lost elements.

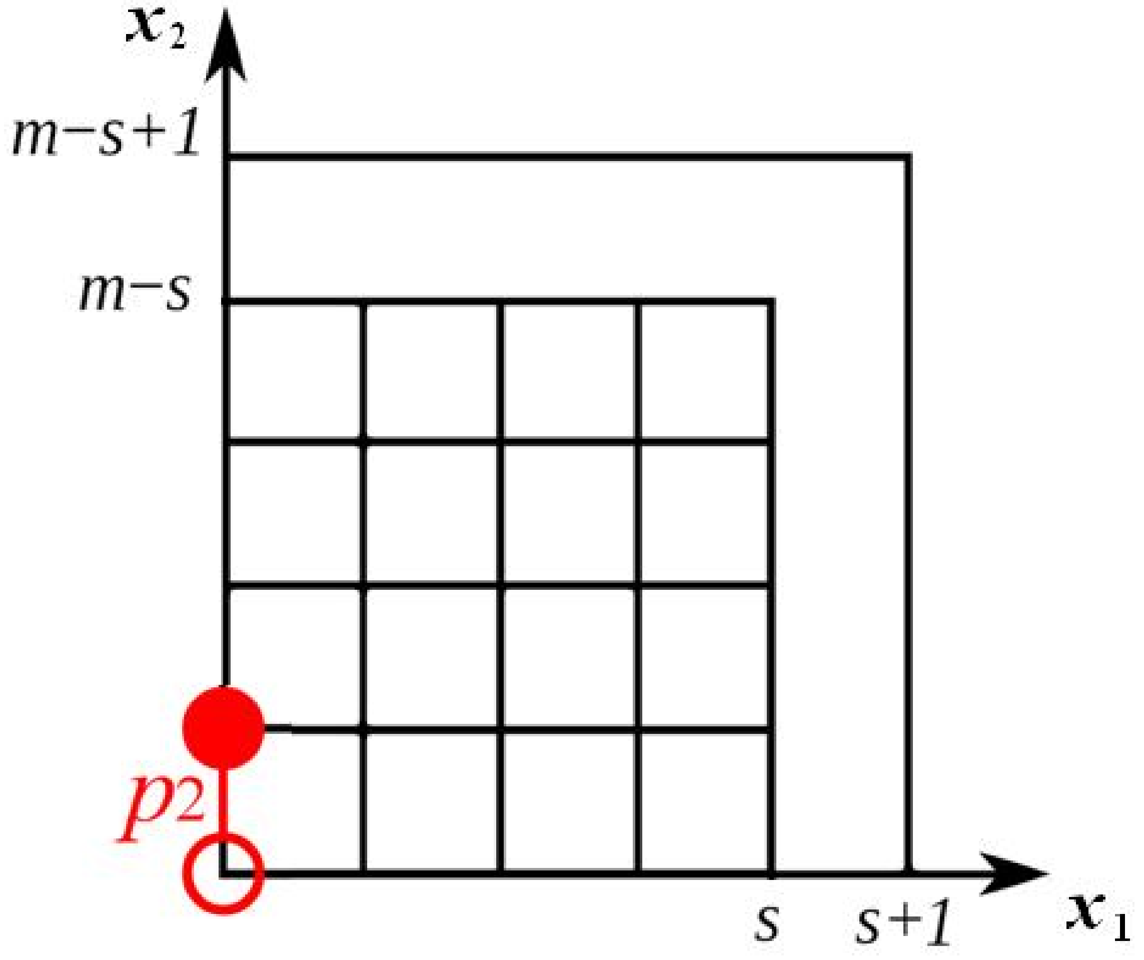
The Model of two deques



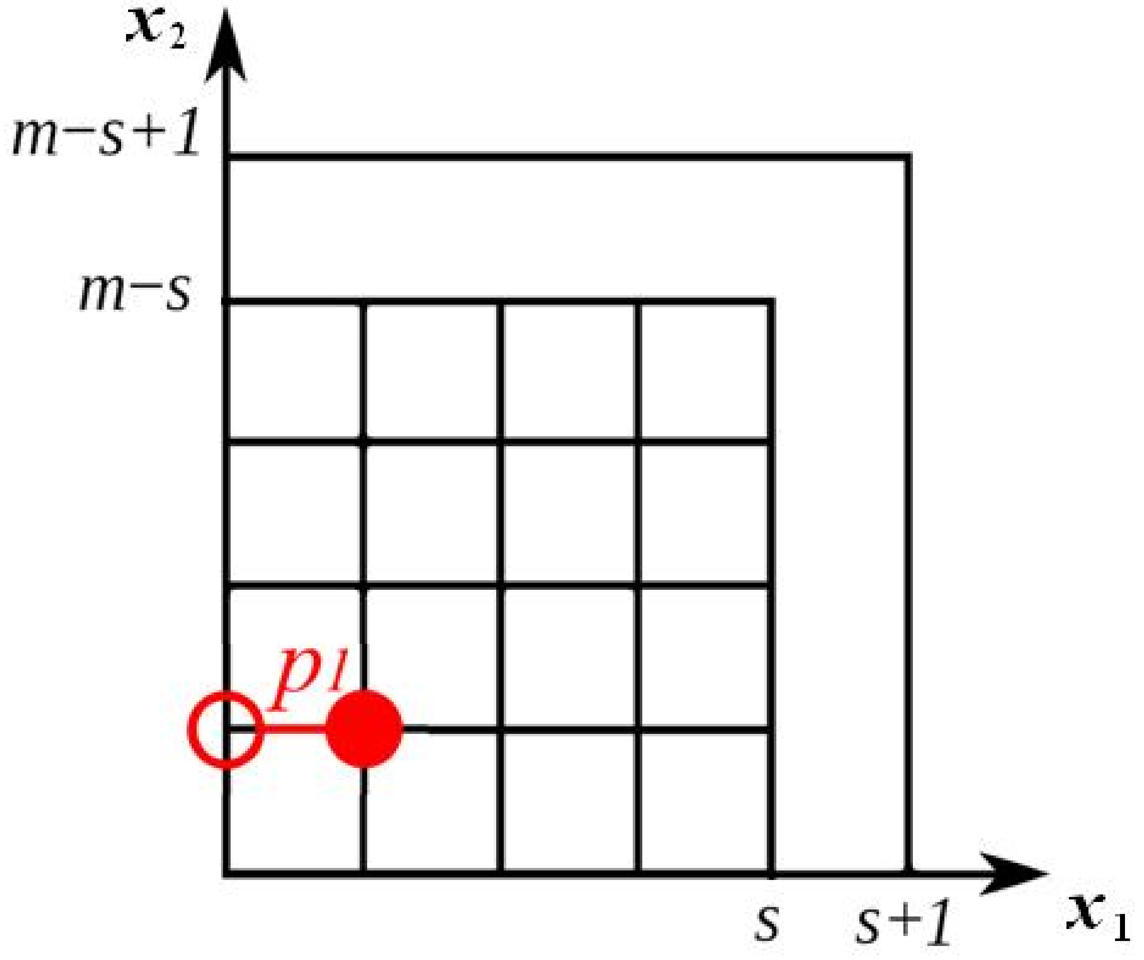


- x_1 и x_2 – the lenghts of 1 and 2 deques;
- $x_1 = s+1$, $x_2 = m-s+1$ - absorbing barriers.

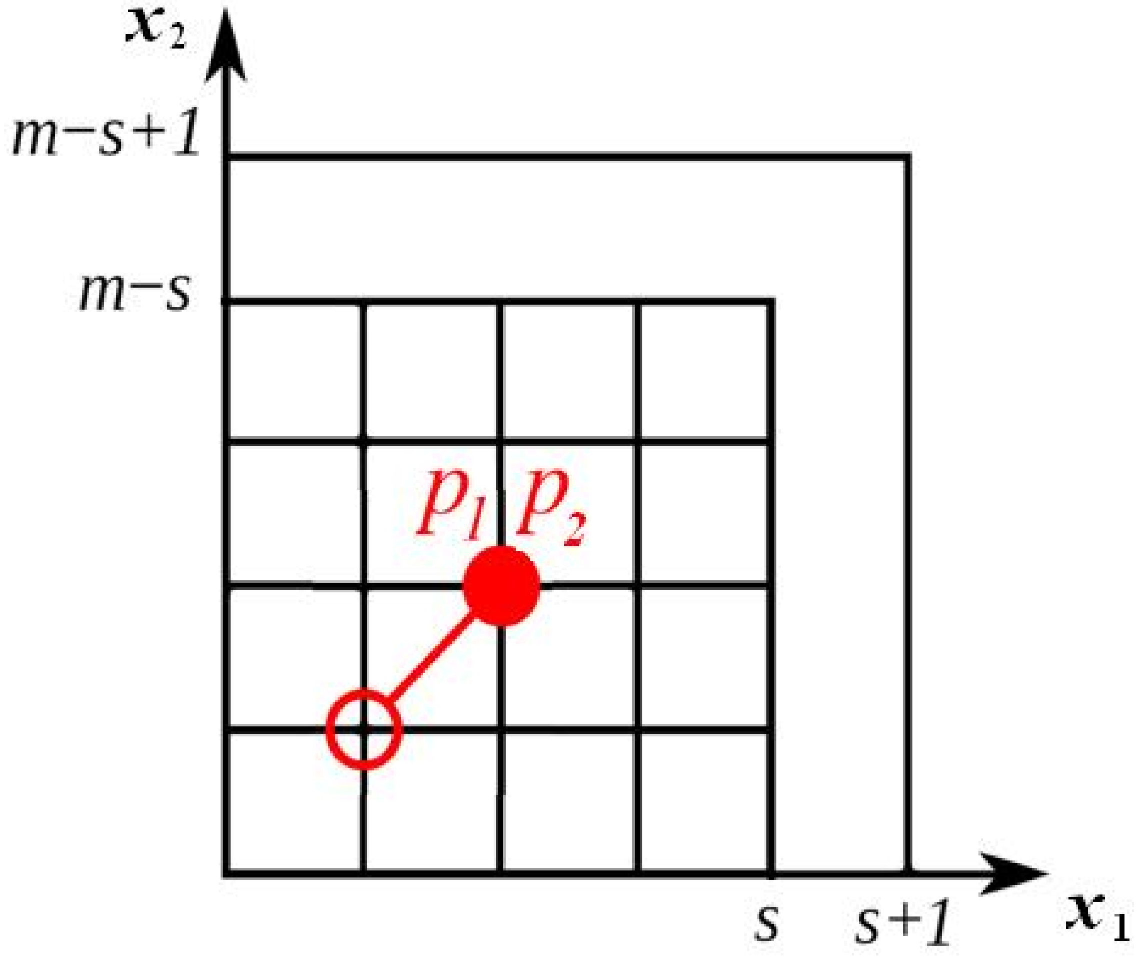
Random walk



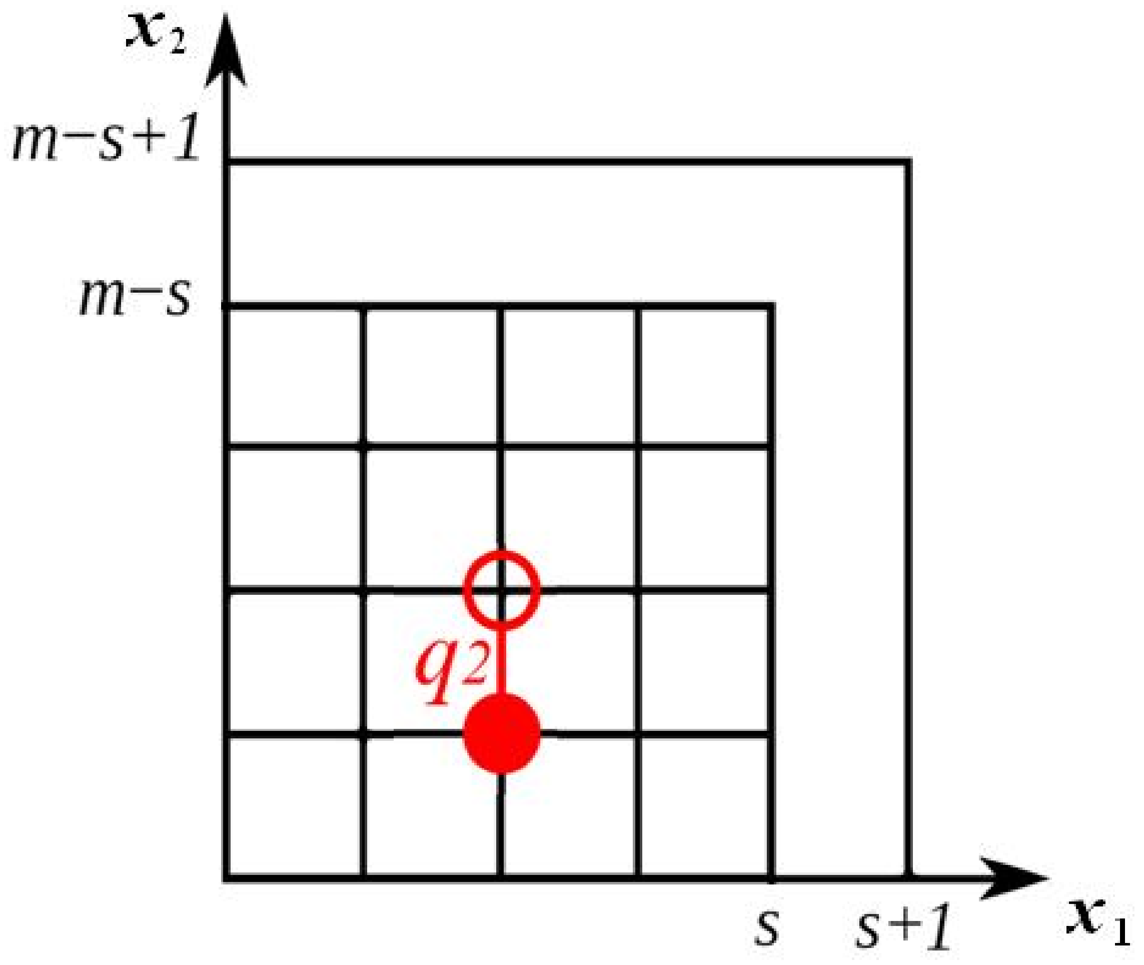
Random walk



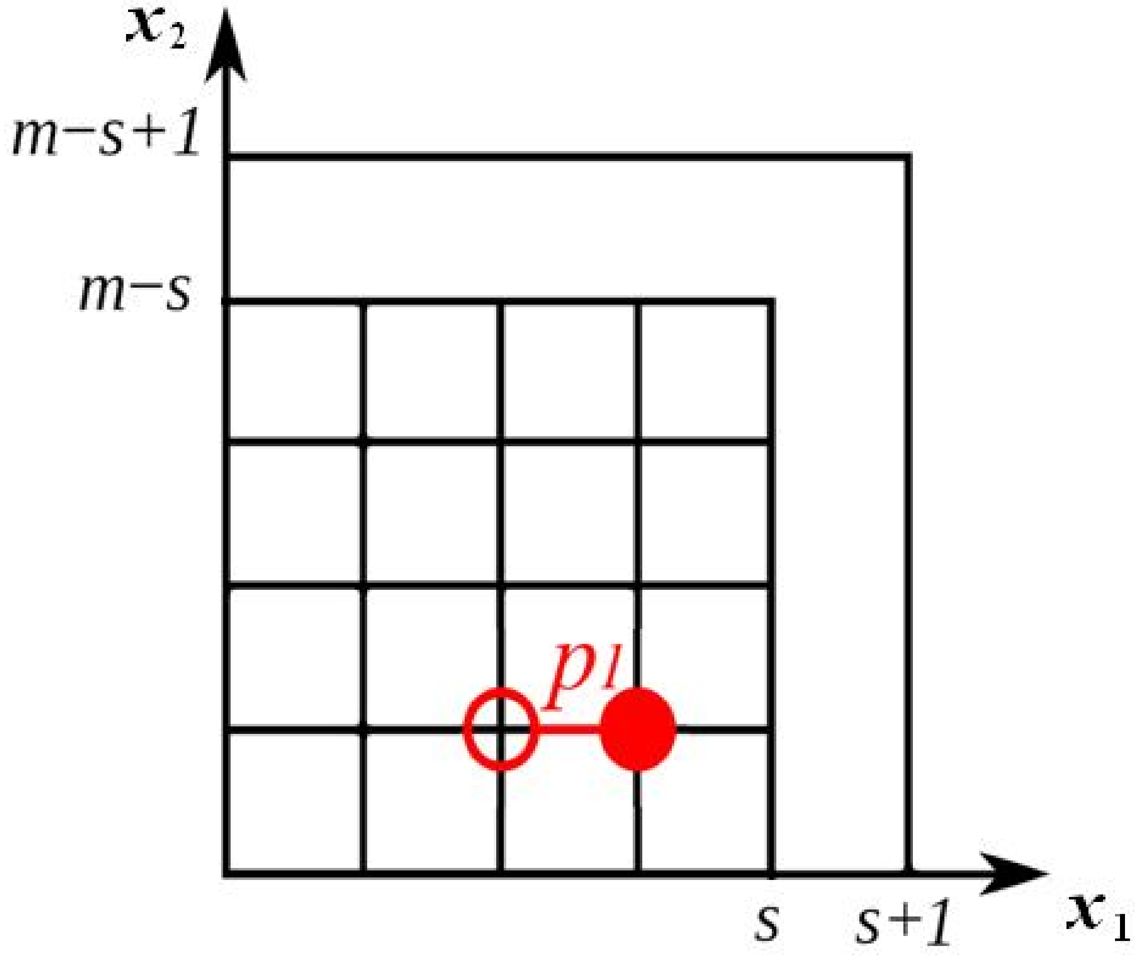
Random walk



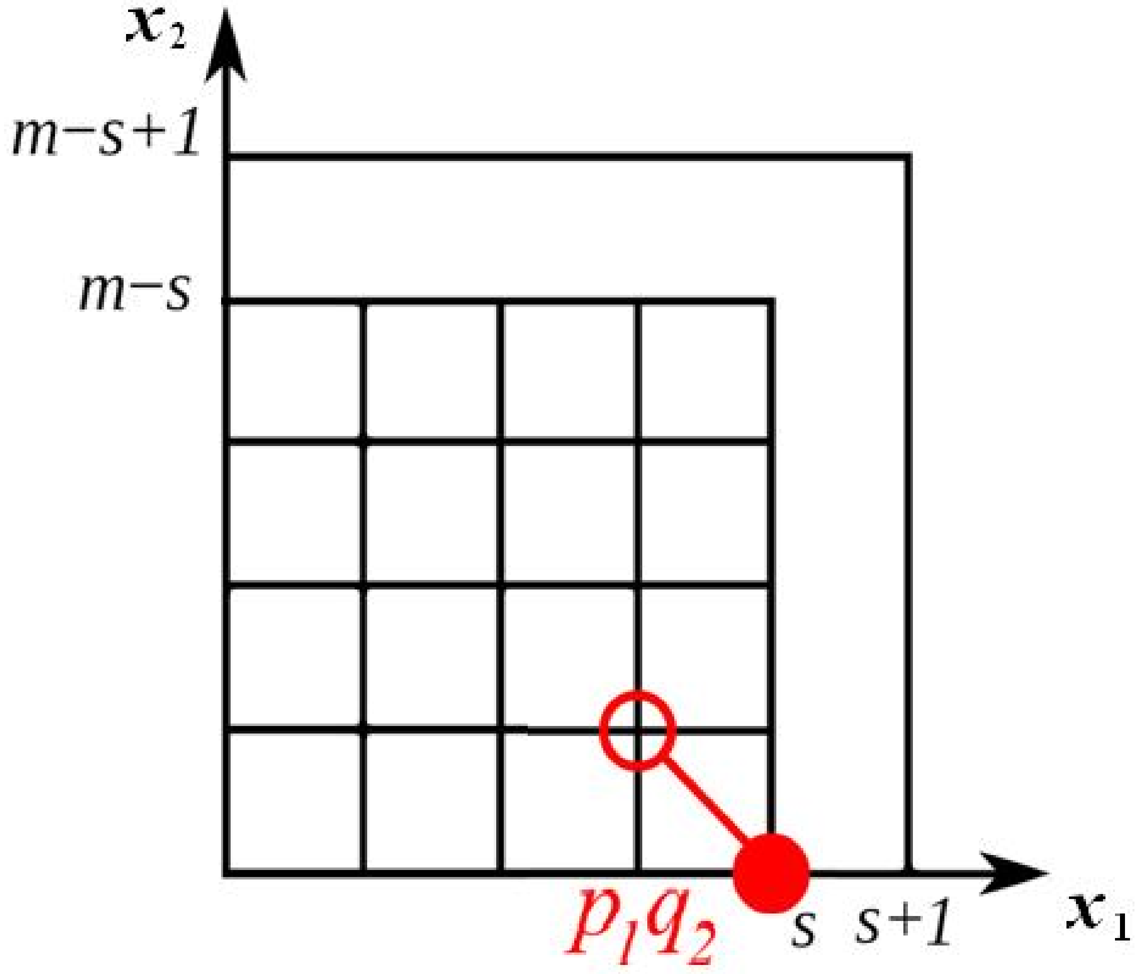
Random walk



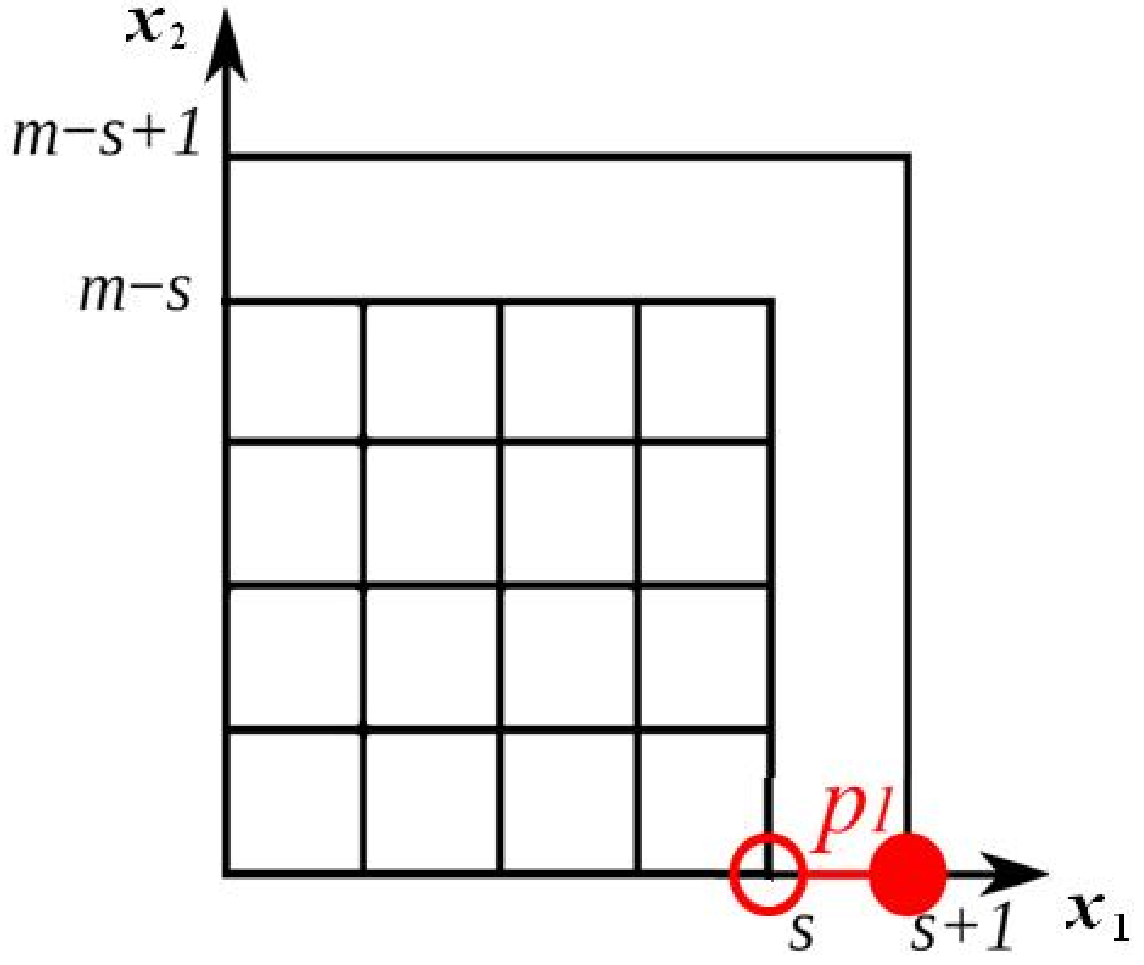
Random walk



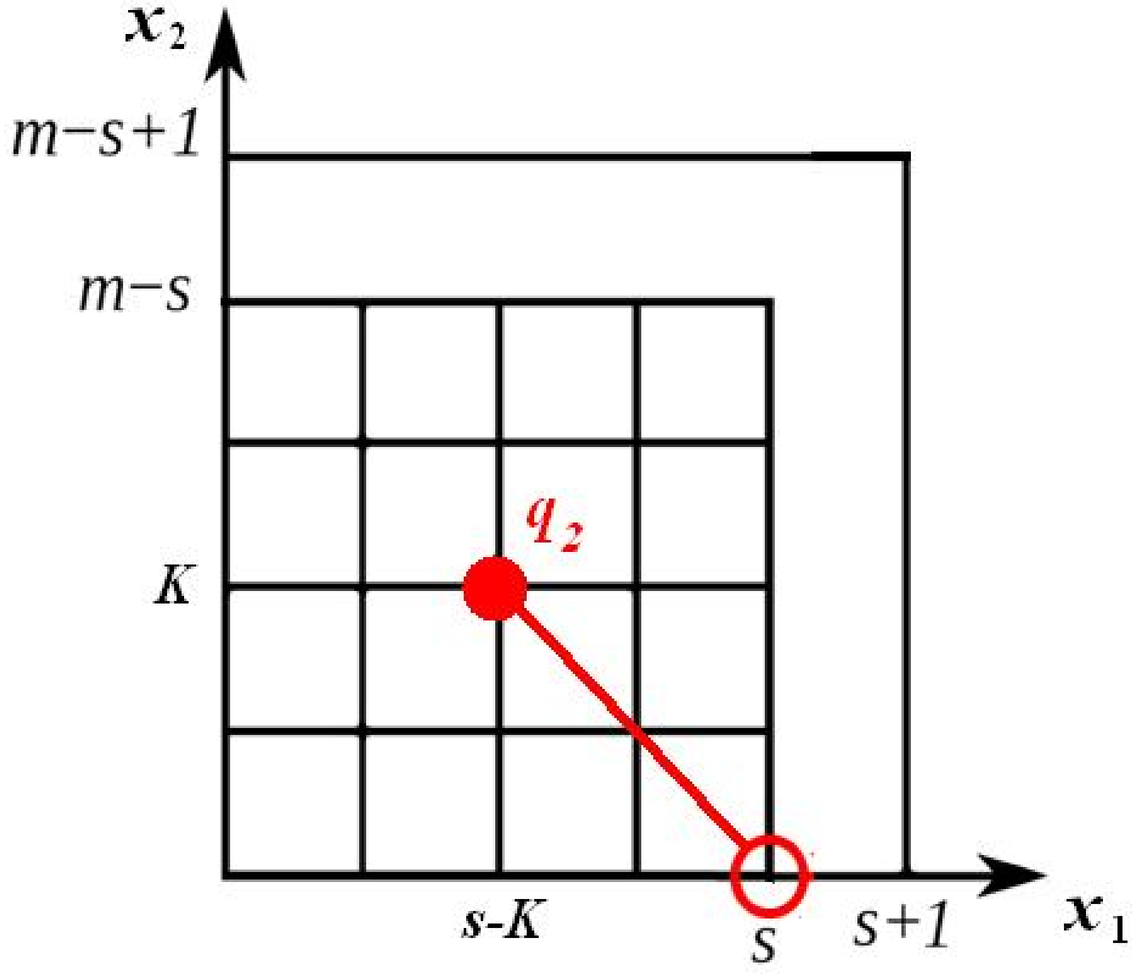
Random walk



Overflow



Stealing of *K*>0 elements (tasks)



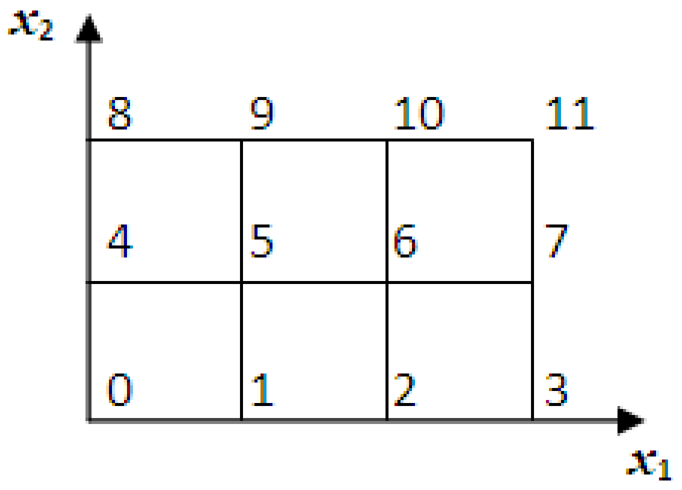
The point with coordinates $(x_1,x_2,x_3,\dots,x_{n-1},x_n)$ is the state of the Markov chain, which describes the system of deques at each step. The process of work (random walk) starts from the point $(0,0,\dots,0)$.

The total number of states in the Markov chain $(s_1+1)(s_2+1)\dots(s_n+1)$.

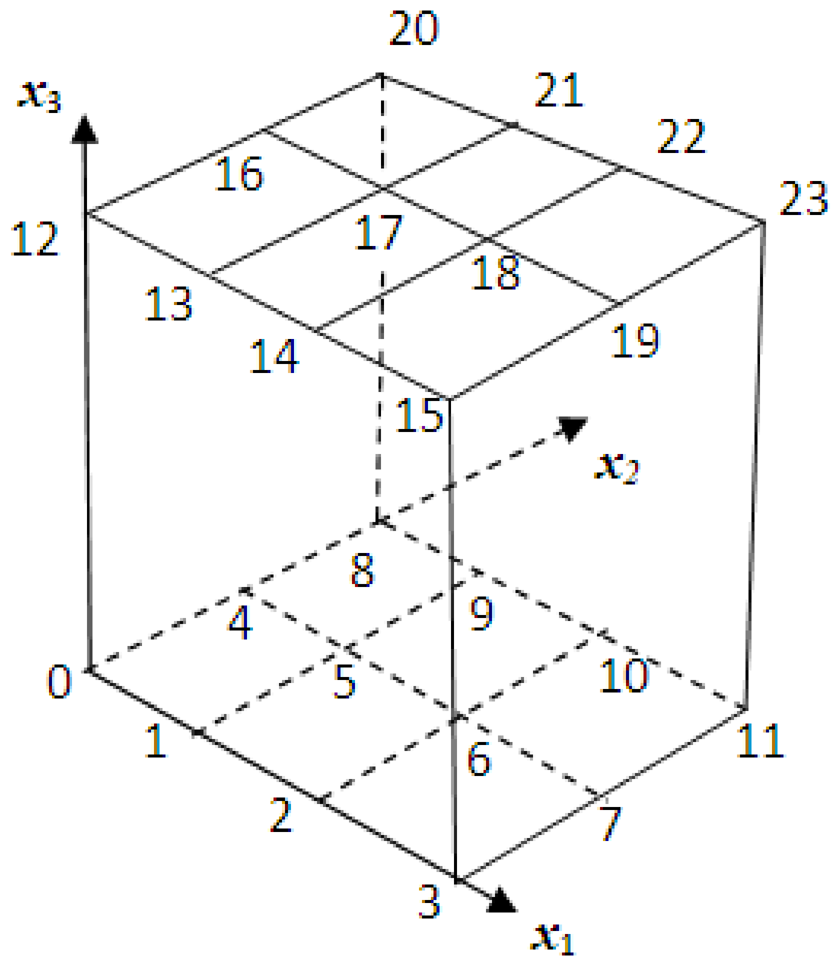
Let's number the states of the Markov chain. By the method of mathematical induction this formula of state numbering for any value of *n* can be proved:

$$M(x_1,x_2,\dots,x_n) = x_1 + x_2(s_1+1) + x_3(s_1+1)(s_2+1) +\dots+$$

$$+x_n(s_1+1)(s_2+1)\dots(s_{n-1}+1).$$



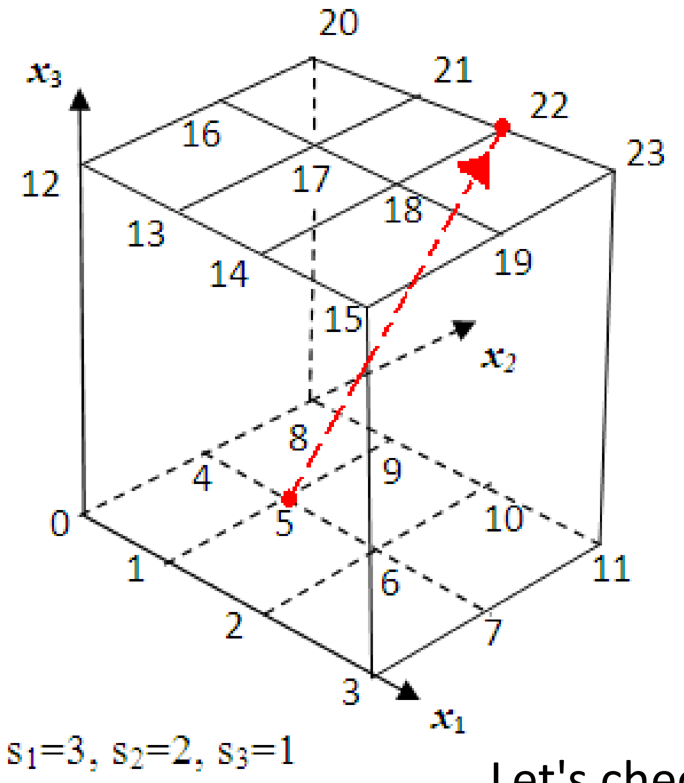
The random walk area and the numbering of states for two deques s1 = 3, s2 = 2



The random walk area and the numbering of states for three deques s1=3, s2=2, s3=1

$$M(x_1,x_2,\ldots,x_n) = x_1 + x_2(s_1+1) + x_3(s_1+1)(s_2+1) + \ldots + x_n(s_1+1)(s_2+1)\ldots(s_{n-1}+1).$$

Consider the transition from the state 5 to the state 22 and determine the lengths of deques x_1, x_2, x_3 for that step by algorithm (this algorithm is based on the formula of numbering of states):



$s_1=3$, $s_2=2$, $s_3=1$

***a*=5:**

$\mathbf{x_3}$:=[*a* / ((3+1)(2+1))]=**0**

a:=*a* % ((3+1)(2+1))=5

$\mathbf{x_2}$:=[*a* / (3+1)]=**1**

a:=*a* % (3+1)=1

$\mathbf{x_1}$:=*a*=**1**

***b*=22:**

$\mathbf{x_3}$:=[*b* / ((3+1)(2+1))]=**1**

b:=*b* % ((3+1)(2+1))=10

$\mathbf{x_2}$:=[*b* / (3+1)]=**2**

b:=*b* % (3+1)=2

$\mathbf{x_1}$:=*b*=**2**

Let's check:

$a=M(1,1,0)=1+1(3+1)+0(3+1)(2+1)=5$,

$b=M(2,2,1)=2+2(3+1)+1(3+1)(2+1)=22$.

At each step in the system of n parallel deques there are n operations of insertion, deletion and processing. In total, 3^n event variants are possible, where each event is a set of n operations of three possible for each deque.

The probabilities of transitions to the Markov chain :

- for each deque $p_i+q_i+r_i=1$, $i=1,...,n$.
- for the system of n deques

$$(p_1+q_1+r_1)(p_2+q_2+r_2)...(p_n+q_n+r_n)=1,$$

where each summand is the probability of the Marcov chain transition at each step, describing what happened with the deques (which n operations were performed).

Suppose that at some step the process was in the state $a=M(x_1,x_2,...,x_n)$ and in the next step the process was got into the state $b=M(x'_1,x'_2,...,x'_n)$. To determine what operations were performed at this step in the deque system, consider the increment values

$\Delta_i=(x'_i - x_i)$ for $i=1,...,n$:

- **1** – the insertion of an element in the i-th deque, $x_i \geq 0$;
- **0** − the length of the i-th deque has not changed, $x_i \geq 0$ (processing or deletion from the empty deque, when steal is impossible);
- **−1** − deletion of the element from the i-th deque, $x_i > 0$;
- ***K*** − *i*-th deque stole *K* elements, $x_i = 0$;
- ***−K*** − there was stealing of K elements from the *i*-th deque, $x_i > K$;
- **(1−*K*)** − there was an insertion of 1 element and stealing of *K* elements, $x_i > K$;
- **(−1−*K*)** − one element has been deleted and *K* elements are stolen, $x_i > K$.

We obtain a set of n elements consisting of $\{0, 1, -1, (-1-K), (1-K), -K, K\}$.

The steal of $K > 0$ elements from the *i*-th deque is possible, if the length of the deque is $x_i > K$, $i=1,...,n$. If there are no deques in the system from which the elements can be stolen, then the length of the deque does not change. The number of deques that have stolen the elements should be equal to the number of deques from which the elements were stolen.

With the help of the defined set of elements we determine the probability of the event that occurred:

$$P(x_i, x'_i) = \prod d_i, \quad i=1,\ldots,n, \text{ where}$$

$$d_i=\{\ p_i,\ \textit{if}\ \Delta_i=1,\ x_i \geq 0\ \textit{or}\ \Delta_i = 1-K,\ x_i>K;$$
$$q_i,\ \textit{if}\ \Delta_i = -1,\ x_i > 0\ \textit{or}\ \Delta_i = -1-K, x_i>K\ \textit{or}\ \Delta_i = K, x_i = 0;$$
$$r_i,\ \textit{if}\ \Delta_i = 0,\ x_i > 0\ \textit{or}\ \Delta_i = -K,\ x_i>K;$$
$$q_i+r_i,\ \textit{if}\ \Delta_i = 0,\ x_i = 0 \text{ and } x_j \leq K,\ i\neq j,\ j=1,\ldots,n;$$
$$0 \text{ for all other cases } \}.$$

Transition probability matrix

$$P = \begin{bmatrix} Q & R \\ O & I \end{bmatrix}$$

- **Q** – transition matrix from the nonrecurring states to the nonrecurring ones;
- **R** – transition matrix from the nonrecurring states to the absorbing ones;
- **I** – identity matrix;
- **O** – zero matrix.

$$Q = \begin{pmatrix} D & A & O & \cdots & O \\ B & C & A & \cdots & O \\ O & B & C & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & A \\ O & O & \cdots & B & F \end{pmatrix}$$

Matrix $\mathbf{Q}$ in the case of two deques has block form, where $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$, $\mathbf{F}$ – submatrixes.

For the case of *n* deques, the matrix of transition probabilities from the nonrecurring states to the nonrecurring ones is given algorithmically. The function **double Qij(int i, int j)** receives state numbers, from which and where the transition occurs, calculates the deque length increments and transition probability.

For decision of the problem, we shall find and sum the elements of the fundamental matrix $\boldsymbol{N = (I-Q)^{-1}}$ in the row corresponding to the initial state $(0,0,\ldots,0)$ for all possible values $\sum s_{\mathrm{i}}=m$, $s_{\mathrm{i}}>0$, $\mathrm{i}=1,\ldots,\mathrm{n}$.

The number of options for partitioning *m* units of memory into *n* deques is C_{m-1}^{n-1} .

Since we need to sum elements in a certain row of the matrix *N* (in our case this is the first row), then we can use the representation of the fundamental matrix as a series, calculating only the elements of the required row

$$N = (I - Q)^{-1} = I + Q + Q^2 + Q^3 + \ldots = \sum_{k=0}^{\infty} Q^k$$

In the mathematical model we consider the value $K \geq 3$, since for values $K = 1,2$ in increments Δ_i of the ***i*-**th deque it is impossible to determine stealing.

Definition of the optimization problem

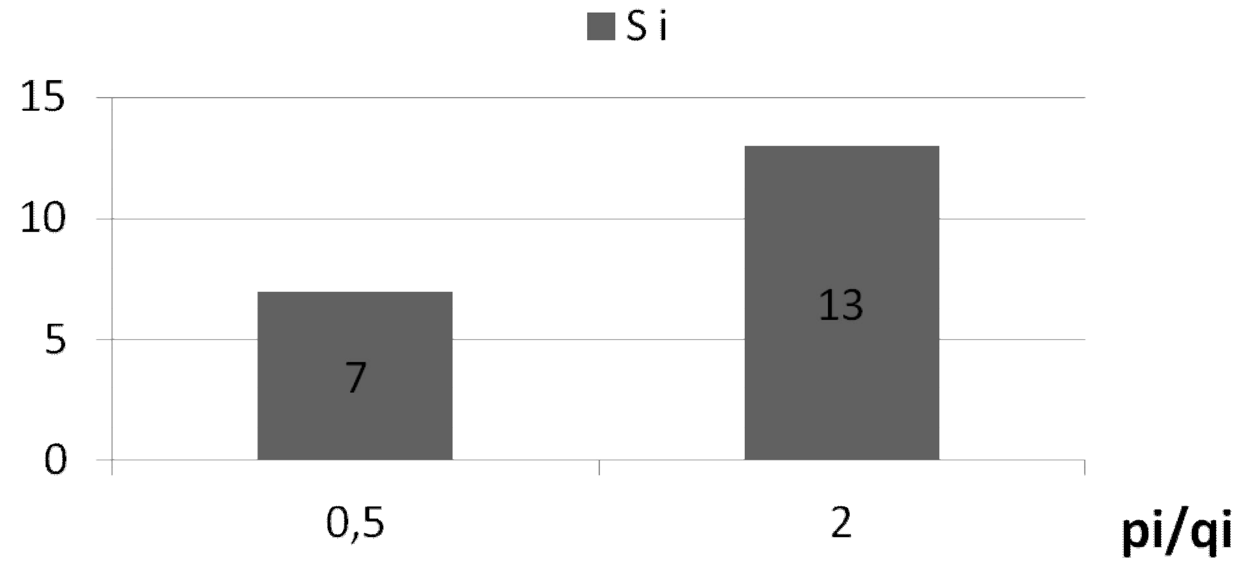
Since a Markov chain is constructed for each set of values si, i = 1, ..., n and K> 0, we can say that we solve the problem of finding the optimal Markov chain for a given optimality criterion or, in other words, we solve the problem of integer nonlinear programming, where the optimality criterion function is algorithmic.

Also, a simulation model of parallel work with Work-stealing deques was built. With the help of a random number generator, at each step we generated a sequence of operations with deques until overflowing of one of deques. The experiment was repeated a selected number of times. For the simulation model the value K > 0.

Results of numerical experiments

m=20 n=2

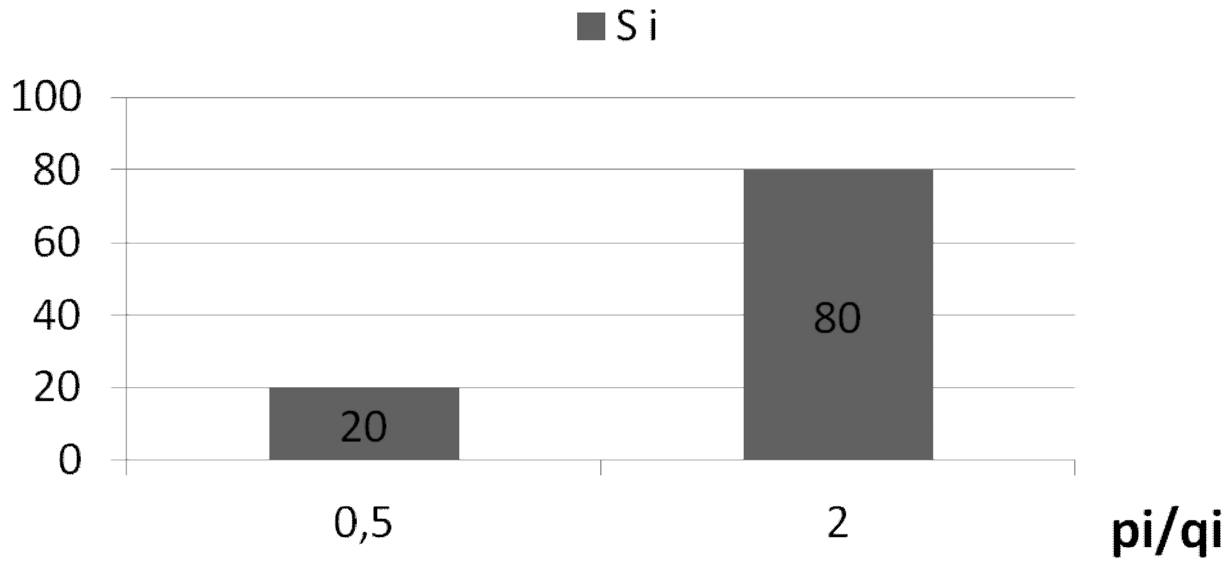
N deque	p_i	q_i	r_i	s_i	K_{opt}	Time
1	0.3	0.6	0.1	7	3	92.4
2	0.6	0.3	0.1	13		



Results of numerical experiments

$m=100$ $n=2$

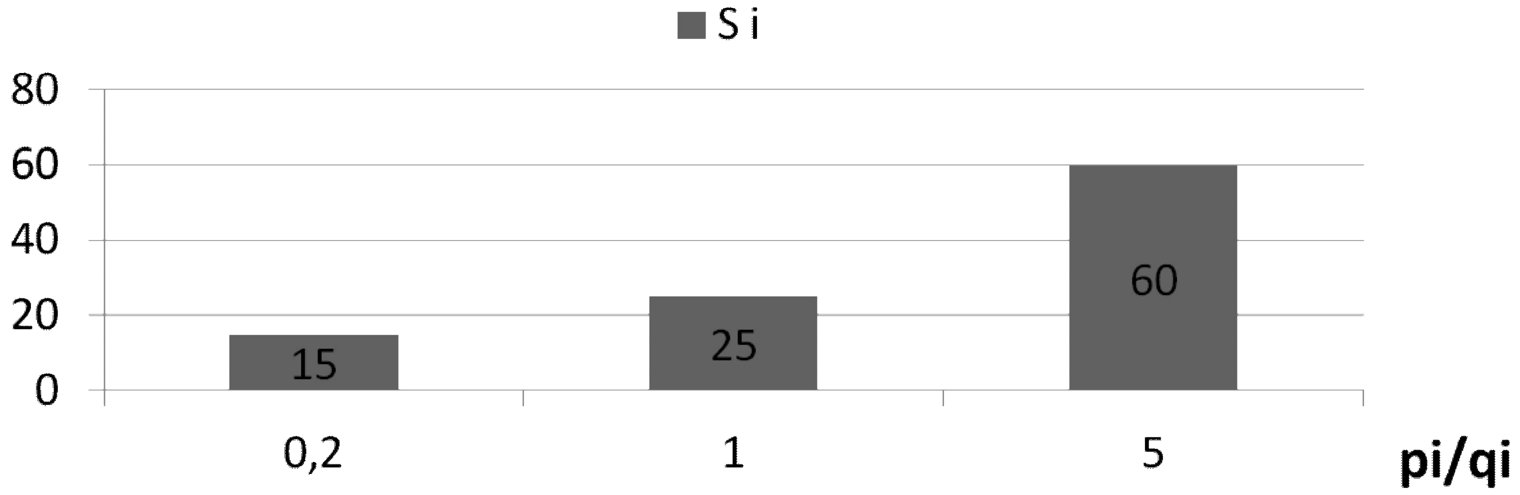
N deque	p_i	q_i	r_i	s_i	K_{opt}	Time
1	0.3	0.6	0.1	20	12	1709.7
2	0.6	0.3	0.1	80		



Results of numerical experiments

m=100 n=3

N deque	p_i	q_i	r_i	s_i	K_{opt}	Time
1	0.1	0.5	0.4	15	12	1055.01
2	0.3	0.3	0.4	25		
3	0.5	0.1	0.4	60		



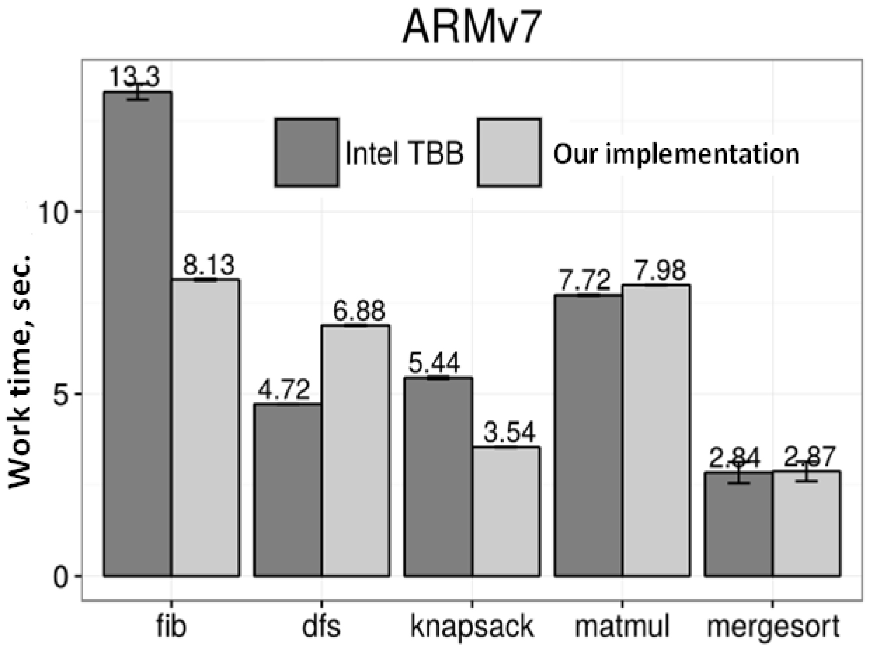
Implementing of the scheduler

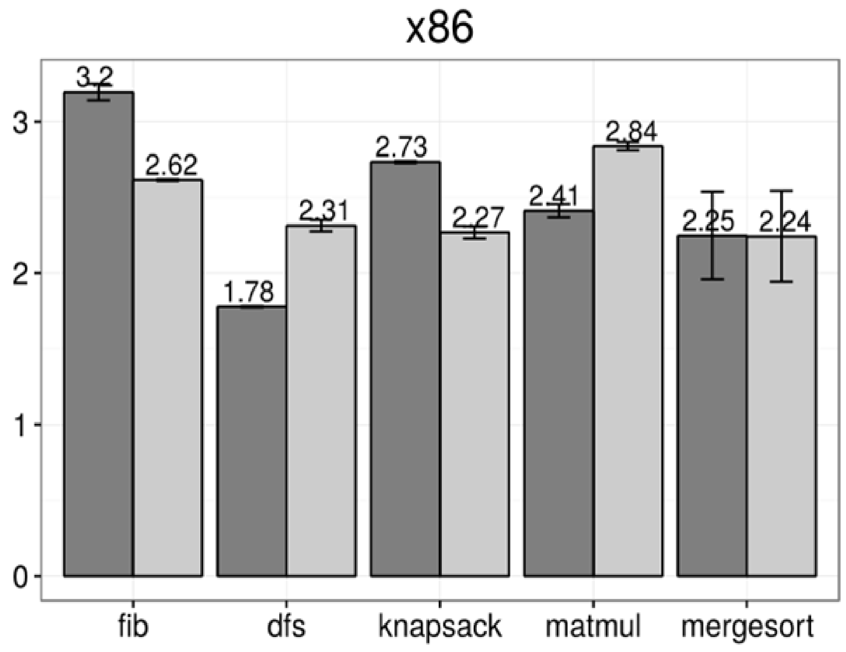
Statistical studies were conducted to assess the probabilities of operations with deques for several types of tasks performed in the scheduler (multiplication of matrices and knapsack problem). For this purpose, within the framework of our RFBR grant experimental work-stealing task scheduler was implemented in C ++ 11, using the implementation of deques based on cyclic arrays with non-blocking synchronizations of operations with the possibility of dynamic resizing. Source codes and implementation on the site https://github.com/rkuchumov/staccato. In comparison with the task scheduler from Intel TBB, which uses the same algorithm of planning and implementation of deques, our implementation showed acceptable results on some tests.

To compare the work of the schedulers, the following tests were used:

- **fib** - calculation of the 35th Fibonacci number by the recurrent formula;
- **knapsack** - solution of the knapsack problem by branch-and-bound method for 26 elements;
- **matmul** - multiplication of 256*256 size matrixes;
- **mergesort** - sorting of 64 MB integers by the method of merging;
- **dfs** – the task graph traversal, in which tasks create 300 subtasks and complete when the depth is 3.

Implementing of the scheduler





Comparisons was carried out on:

- Intel Pentium N3530 2.16GHz, 4 CPU, 64L1, Linux 4.4.0
- ARMv7l rev 5, 4 CPU, 32 L1, Linux 4.4.8

with the same compilation parameters.

In practice, including in the implementation of the balancer described in the report, classical methods of working with a heap in C / C ++ translators are often used to work with deques in the form of cyclic arrays. For each deque, an array is requested from the heap. When it becomes large enough, a new array is requested, for example, twice as large - the elements of the deque are copied there, and the old array is given to the list of free heap blocks. If on the contrary, when the deque has become sufficiently small, then the same actions are performed with an array smaller than the old one twice. Perhaps a modified variant of the Garwick algorithm will be useful here, where when a overflows, a local redistribution of a part of the deques located next to the overflowing deque is made.

In the model presented in the report, we consider a random walk starting from the origin of coordinates(at the beginning of the work, the deques are empty). We also started work on the development of models in which the problem was solved not of the initial division of memory, but of an optimal redistribution of memory after overflow. There, the initial point of the walk is one in which we are overwhelmed, or the point obtained from it due to the optimal interception. This model can be used if we consider time intervals with different probabilities of operations with deques. To be used in practice, such a model should be applied several times with different transition matrices, if statistical studies have revealed intervals of time at which the probabilities of operations with deques remain practically unchanged (since we are considering a homogeneous Markov chain when the probabilities do not depend from time).

References

- Blumofe R.D., Leiserson C.E. Scheduling multithreaded computations by work stealing // Journal of the ACM, 1999. N.46, P.720–748.
- Herlihy M., Shavit N. The Art of Multiprocessor Programming. Elsevier, 2008.
- Knuth D. The Art of Computer Programming. V.1. Addison-Wesley, 2001.
- Hendler D., Shavit N. Non-blocking Steal-half Work Queues // ACM Symposium on Principles of Distributed Computing, 2002. P.280–289.
- Sokolov A.V., Barkovsky E.A. The Mathematical Model and The Problem of Optimal Partitioning of Shared Memory for Work-Stealing Deques // PaCT, LNCS, 2015. V. 9251, P.102–106.
- Eugene Barkovksy, Ruslan Kuchumov, Andrew Sokolov. “Optimal control of two deques in shared memory with various work-stealing strategies”, Program systems: Theory and applications, 2017,8:1(32), pp. 83–103. (In Russian). URL:http://psta.psiras.ru/read/ psta2017_1_83-103.pdf

Thank you for attention!

The research was supported by RFBR № 15-01-03404-a