Епифанцев Егор Витальевич, группа 9305

Тема: «Реализация разделяемых структур данных в модели MPI RMA»

Цель работы: Разработка распределенного связного списка, стека и очереди в модели удаленного доступа к памяти (MPI RMA).

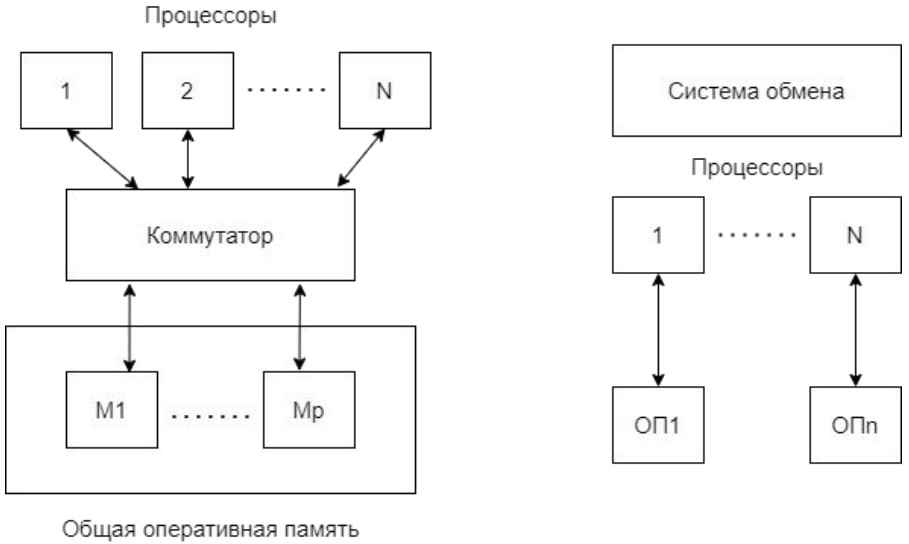
Объект и предмет исследования: объект - разделяемые структуры данных. Предмет исследования - связный список с использованием блокировок, неблокирующая очередь Майкла и Скотта и неблокирующий стек Трайбера.

Техническое задание: Структуры должны быть реализованы на языке программирования Си с использованием библиотеки OpenMPI и корректно выполняться на кластере.

Актуальность темы

В настоящее время вычислительные кластеры применяются во многих областях науки, например, в физике, химии, астрономии, биологии, фармакологии для решения сложных вычислительных задач и моделирования. Такие системы используют модель распределенной памяти, в отличие от обычных ПК.

Распределенная же память приводит к созданию распределенных структур данных



Блокирующая синхронизация

Примитивы синхронизации:

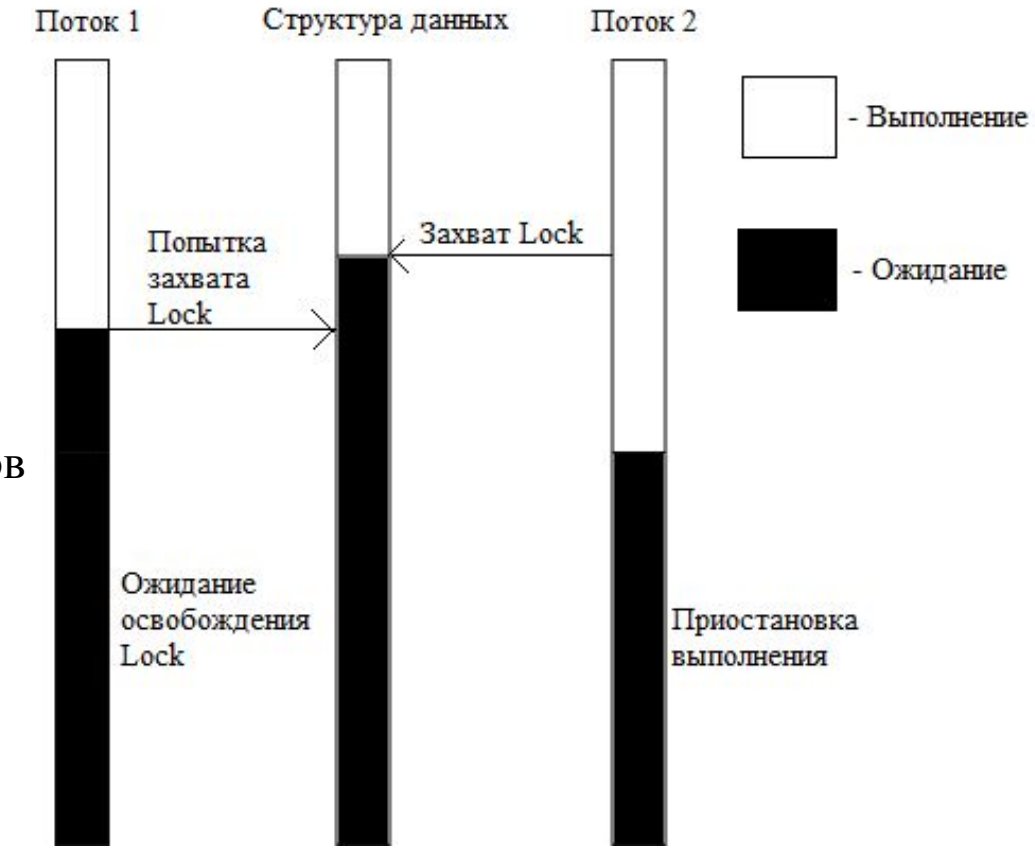
- Мьютексы;
- Семафоры;
- Спинлоки.

Достоинства:

- Простота реализации алг-мов

Недостатки:

- Низкая масштабируемость;
- Низкая отказоустойчивость.



Неблокирующая синхронизация

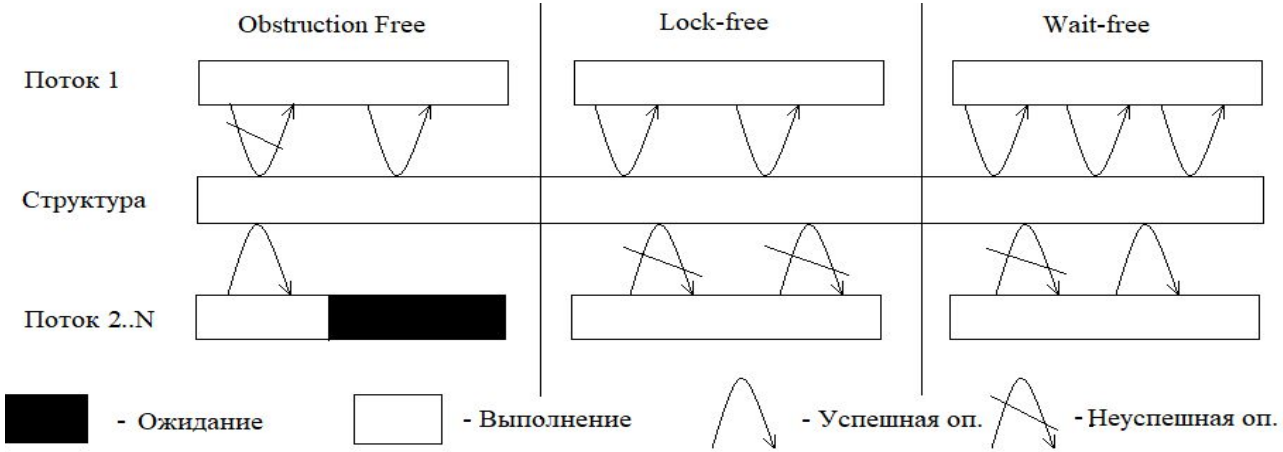
Атомарные операции (особенно CAS)

Достоинства:

- Высокая отказоустойчивость;
- Высокая масштабируемость;

Недостатки:

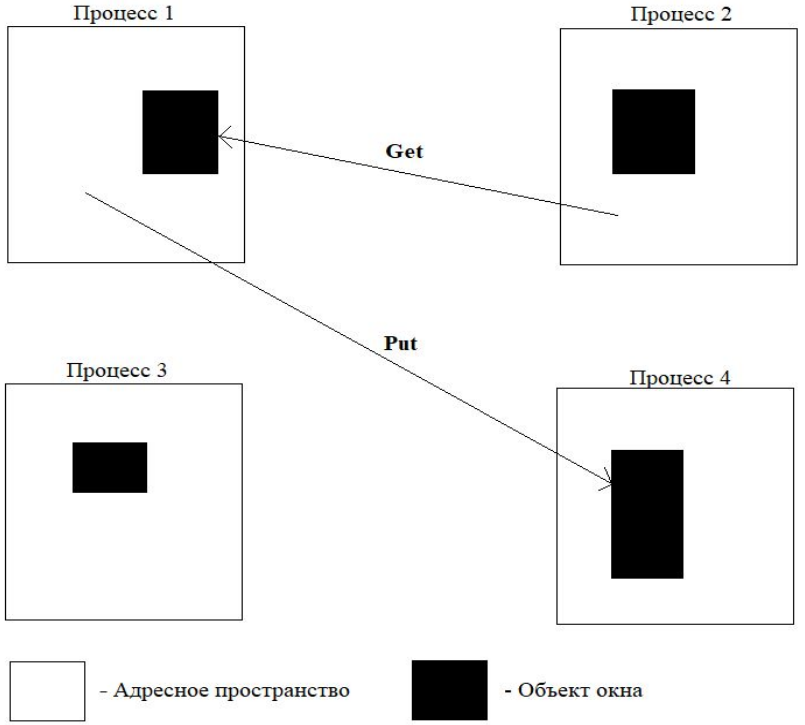
- Сложность построения алгоритмов;
- Большое количество атомарных оп.



Модель MPI RMA

MPI RMA - модель программирования, реализованная в библиотеки MPI, которая предоставляет возможность прямого доступа к памяти удаленных процессов без необходимости явного обмена сообщениями.

- Окна - сегменты памяти для межпроцессного взаимодействия.
- Эпохи - части программы, внутри которых происходит синхронизация
- RMA-операции - используются внутри эпох для межпроцессного обмена данными



Пассивная синхронизация

MPI RMA предоставляет два варианта синхронизации - активная и пассивная. При активной синхронизации оба процесса оказываются вовлечены в процесс синхронизации. При пассивной же только один (origin process). В работе используется пассивный метод синхронизации, так как он имеет меньше накладных расходов.

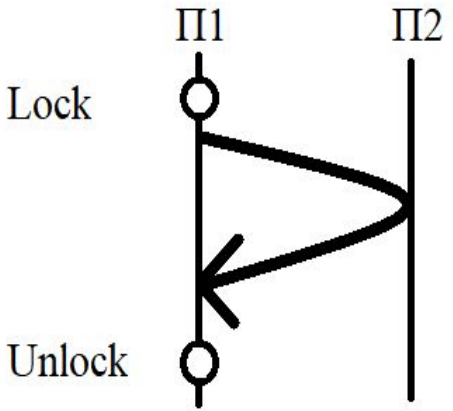
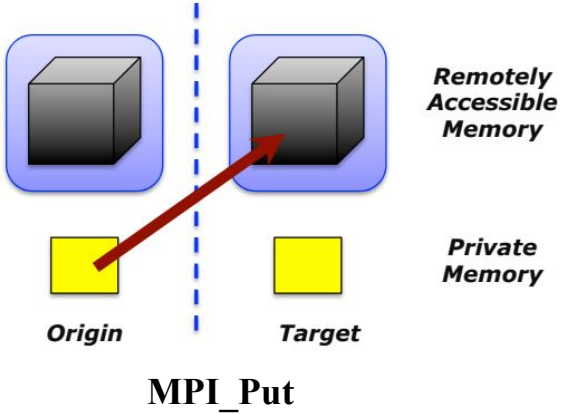


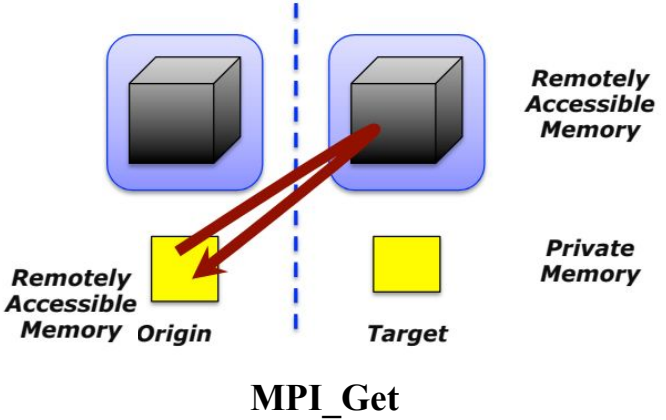
Схема пассивной синхронизации

RMA-операции

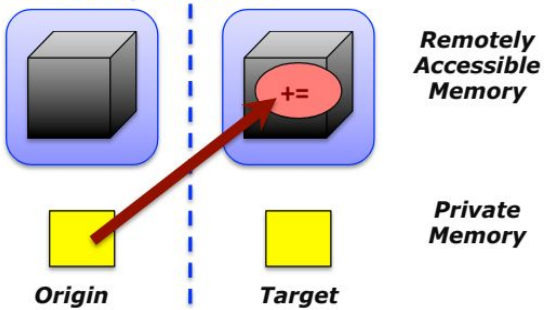
- MPI_Put (неблокирующая)
- MPI_Get (неблокирующая)
- MPI_Compare_and_swap (атомарная)
- MPI_Fetch_and_op (атомарная)
- MPI_Accumulate (атомарная)
- MPI_Get_accumulate (атомарная)



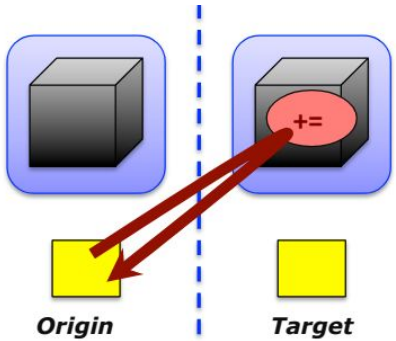
MPI_Put



MPI_Get



MPI_Accumulate



MPI_Get_Accumulate

Неблокирующая очередь Майкла и Скотта

```
1: function ENQUE(val, rank, q, win)
2:     tmpTail ← nullPtr
3:     tailNext ← nullPtr
4:     newNode ← allocElem(val, rank, win)
5:     MPI_Win_lock_all(0, win)
6:     while True do
7:         tmpTail ← getTail(q, win)
8:         MPI_Compare_and_swap(newNode, nullPtr, result, tmpTail.rank, tmpTail.offset + offsetof(node, next))
9:         MPI_Win_flush(tmpTail.rank, win)
10:        if result == nullPtr then
11:            MPI_Compare_and_swap(newNode, tmpTail, result, 0, q.tail.offset + offsetof(node, next))
12:            MPI_Win_flush(0, win)
13:            return
14:        else
15:            tailNext ← getTail(q, win)
16:            MPI_Compare_and_swap(tailNext, tmpTail, result, 0, q.tail.offset + offsetof(node, next))
17:            MPI_Win_flush(0, win)
18:        end if
19:    end while
20:    MPI_Win_unlock_all(win)
21: end function
```

```
typedef struct{
      nodePtr dummy;
      nodePtr head;
      nodePtr tail;
} Queue

typedef struct{
       int val;
       nodePtr next;
} node


typedef struct{
        uint64_t rank : 11;
        uint64_t offset : 53;
} nodePtr
```

Неблокирующая очередь Майкла и Скотта

```
1: function DEQUEUE(q, win)
2:     tail ← nullPtr
3:     head ← nullPtr
4:     afterHead ← nullPtr
5:     result ← nullPtr
6:     MPI_Win_lock_all(0, win)
7:     while True do
8:         head ← getHead(q, win)
9:         tail ← getTail(q, win)
10:        afterHead ← getNextHead(head, win)
11:        if tail == head then
12:            if afterHead == nullPtr then
13:                return
14:            else
15:                MPI_Compare_and_Swap(afterHead, tail, result, 0, q.tail.offset + offsetof(node, next))
16:                MPI_Win_flush(0, win)
17:            end if
18:        else
19:            MPI_Compare_and_swap(afterHead, head, result, 0, q.head.offset + offsetof(node, next))
20:            MPI_Win_Flush(0, win)
21:            if result == head then
22:                readVal(result, win)
23:                return
24:            end if
25:        end if
26:    end while
27:    MPI_Win_unlock_all(win)
28: end function
```

Неблокирующий стек Трайбера

```
1: function PUSH(val, rank, s, win)
2:     curHead ← nullPtr
3:     newHead ← nullPtr
4:     result ← nullPtr
5:     newHead ← allocElem(val, win)
6:     MPI_Win_lock_all(0, win)
7:     while True do
8:         curHead ← getHead(s, win)
9:         changeNext(curHead, newHead, win)
10:        MPI_Compare_and_swap(newHead, curHead, result, s.dummy.rank, s.dummy.offset + offsetof(node, next)
11:        MPI_Win_flush(s.dummy.rank, win)
12:        if result == curHead then
13:            return
14:        end if
15:    end while
16:    MPI_Win_unlock_all(win)
17: end function
```

```
typedef struct{
        nodePtr dummy;
        nodePtr head;
} Stack
```

```
typedef struct{
        int val;
        nodePtr next;
} node
```

```
typedef struct{
        uint64_t rank : 11;
        uint64_t offset : 53;
} nodePtr
```

Неблокирующий стек Трайбера

```
1: function POP(s, win)
2:     curHead ← nullPtr
3:     nextHead ← nullPtr
4:     result ← nullPtr
5:     MPI_Win_lock_all(0, win)
6:     while True do
7:         curHead ← getHead(s, win)
8:         if curHead == nullPtr then
9:             return
10:        end if
11:        nextHead ← getNextHead(curHead, win)
12:        MPI_Compare_and_swap(nextHead, curHead, result, s.dummy.rank, s.dummy.offset + offsetof(node, next
13:        MPI_Win_Flush(s.dummy.rank, win)
14:        if result == curHead then
15:            return
16:        end if
17:    end while
18:    MPI_Win_unlock_all(win)
19: end function
```

Тестирование на кластере

Экспериментальное исследование проводилось на вычислительном кластере с 4 вычислительными узлами. При этом на каждом из узлов находилось по 1 4-ядерному процессору линейки Intel Xeon с базовой частотой 2 ГГц и максимальной частотой 3.2 ГГц. В качестве MPI-библиотеки использовалась OpenMPI 4.1.2.

```
processor       : 7
vendor_id       : GenuineIntel
cpu family      : 6
model           : 106
model name      : Intel Xeon Processor (Icelake)
stepping        : 0
microcode       : 0x1
cpu MHz         : 1995.312
cache size      : 16384 KB
physical id     : 0
siblings        : 8
core id         : 3
cpu cores       : 4
apicid          : 7
initial apicid  : 7
fpu             : yes
fpu_exception   : yes
cpuid level     : 13
wp              : yes
```

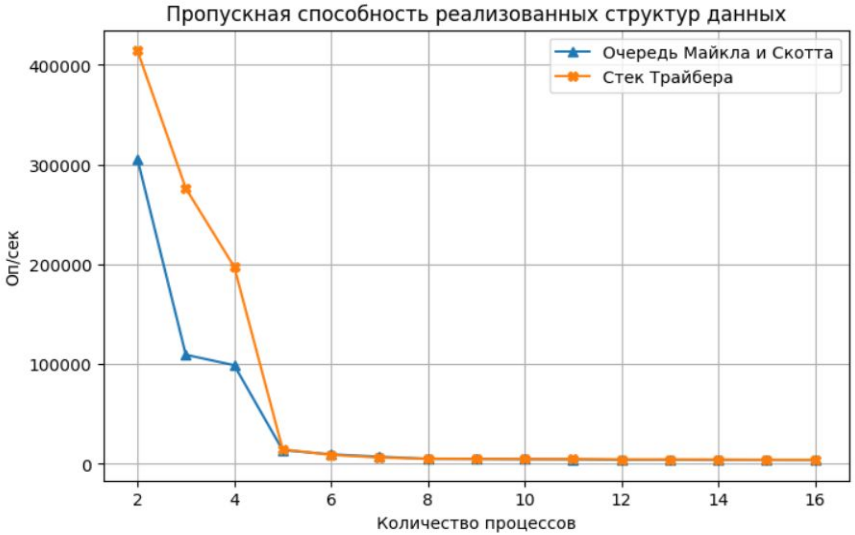
Конфигурация отдельного узла

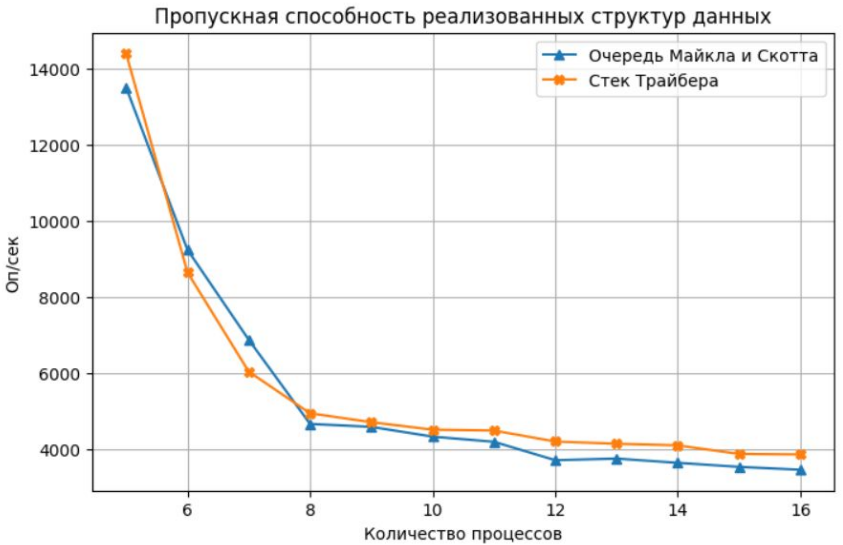
Тестирование на кластере

```
------val 9974 was inserted by rank 14 at displacement 559bac4c8dd0 next rank 14 next displacement 559bac4c8df0------
------val 9976 was inserted by rank 14 at displacement 559bac4c8df0 next rank 14 next displacement 559bac4c8e10------
------val 9977 was inserted by rank 14 at displacement 559bac4c8e10 next rank 14 next displacement 559bac4c92f0------
------val 9980 was inserted by rank 14 at displacement 559bac4c92f0 next rank 14 next displacement 559bac4c9310------
------val 9981 was inserted by rank 14 at displacement 559bac4c9310 next rank 14 next displacement 559bac4c9330------
------val 9986 was inserted by rank 14 at displacement 559bac4c9330 next rank 14 next displacement 559bac4c9350------
------val 9987 was inserted by rank 14 at displacement 559bac4c9350 next rank 14 next displacement 559bac4c9830------
------val 9989 was inserted by rank 14 at displacement 559bac4c9830 next rank 14 next displacement 559bac4c9850------
------val 9990 was inserted by rank 14 at displacement 559bac4c9850 next rank 14 next displacement 559bac4c9870------
------val 9992 was inserted by rank 14 at displacement 559bac4c9870 next rank 14 next displacement 559bac4c9890------
------val 9993 was inserted by rank 14 at displacement 559bac4c9890 next rank 14 next displacement 559bac4c9d70------
------val 9994 was inserted by rank 14 at displacement 559bac4c9d70 next rank 14 next displacement 559bac4c9d90------
------val 9996 was inserted by rank 14 at displacement 559bac4c9d90 next rank 14 next displacement 559bac4c9db0------
------val 9998 was inserted by rank 14 at displacement 559bac4c9db0 next rank 2047 next displacement 0------
Total element count = 457
Expected element count = 457
Test result: total elapsed time = 46.377310 ops/sec = 3449.962939
Queue Integrity: True
rank 2 of all 16 ranks was working on node master-node
rank 3 of all 16 ranks was working on node master-node
rank 1 of all 16 ranks was working on node master-node
rank 0 of all 16 ranks was working on node master-node
rank 7 of all 16 ranks was working on node cl1vpq12ej5uakm60r3r-ynih
rank 5 of all 16 ranks was working on node cl1vpq12ej5uakm60r3r-ynih
rank 4 of all 16 ranks was working on node cl1vpq12ej5uakm60r3r-ynih
rank 6 of all 16 ranks was working on node cl1vpq12ej5uakm60r3r-ynih
rank 11 of all 16 ranks was working on node cl1vpq12ej5uakm60r3r-awef
rank 15 of all 16 ranks was working on node cl1vpq12ej5uakm60r3r-ebab
rank 8 of all 16 ranks was working on node cl1vpq12ej5uakm60r3r-awef
rank 13 of all 16 ranks was working on node cl1vpq12ej5uakm60r3r-ebab
rank 9 of all 16 ranks was working on node cl1vpq12ej5uakm60r3r-awef
rank 14 of all 16 ranks was working on node cl1vpq12ej5uakm60r3r-ebab
rank 10 of all 16 ranks was working on node cl1vpq12ej5uakm60r3r-awef
rank 12 of all 16 ranks was working on node cl1vpq12ej5uakm60r3r-ebab
```

Пример теста очереди на кластере

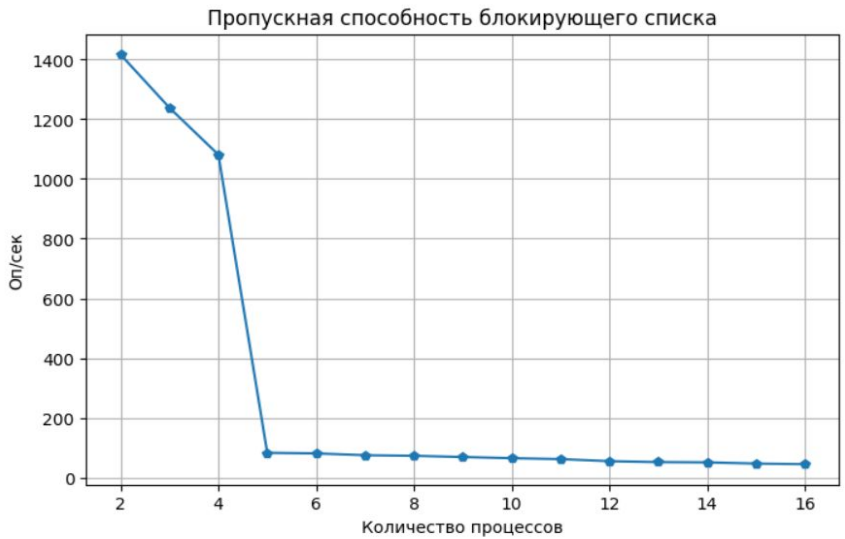
Тестирование на кластере

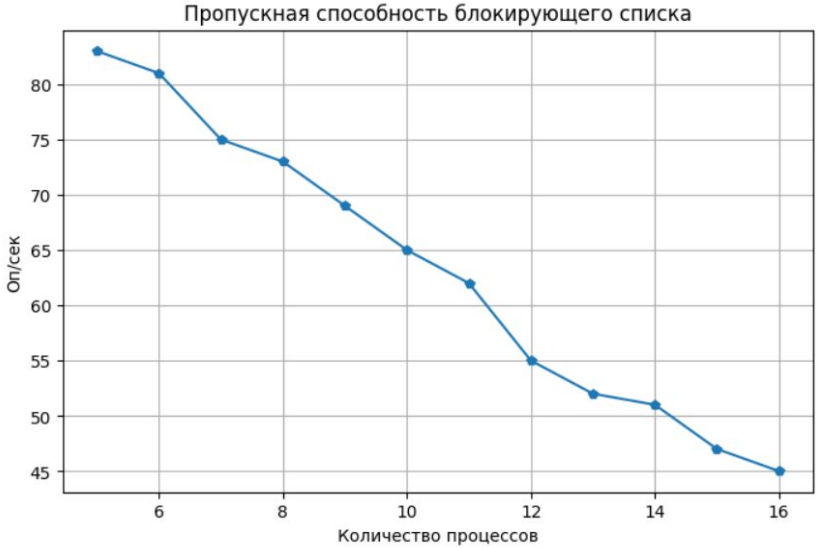




Каждый процесс производил по 10000 операций вставки/удаления (тип операции выбирался равновероятно)

Тестирование на кластере





Изначально корневой процесс генерировал список длиной 512 элементов, затем каждый из процессов производил по 512 операций вставки/удаления (тип операции выбирался равновероятно)

Заключение

Результаты: разработаны 3 структуры данных в модели MPI RMA. Для каждой из структур данных были проведены тесты на вычислительном кластере, которые показали, что очередь и стек достаточно хорошо масштабируются как минимум до 16 процессов. Реализованные структуры могут найти применение на вычислительных системах с распределенной памятью, например, для реализации более сложных структур данных (очередь с приоритетом, развернутый связный список и т.д.) или для хранения и упорядочивания данных, полученных в ходе вычислений.

Дальнейшие перспективы развития темы:

- В неблокирующих структурах отойти от идеи централизации головы и хвоста структур в памяти корневого процесса;
- Реализовать неблокирующий связный список, например, список Харриса.