Оптимизация утилизации при выделении ресурсов для высокопроизводительных вычислительных систем с сетью Ангара

А. В. Мукосей, А. С. Семенов
АО «НИЦЭВТ»

Работа посвящена оптимизации утилиции при выделении вычислительных узлов для заданий в суперкомпьютере с сетью Ангара, имеющей топологию «многомерный тор». В работе показано, что перестановка пользовательских заданий в очереди одновременно с использованием метода выделения ресурсов, сокращающего фрагментацию системы, в среднем дает прирост утилизации ресурсов на 7% и на 36,6% сокращает значение время ожидания задания в очереди по сравнению с базовым методом выбора узлов.

Ключевые слова: коммуникационная сеть Ангара, многомерный тор, планирование ресурсов, фрагментация, выбор узлов

1. Введение

В АО «НИЦЭВТ» разработана высокоскоростная коммуникационная сеть Ангара [1,2] с топологией «многомерный тор». В маршрутизаторе сети реализована бездедлоковая, адаптивная маршрутизация, основанная на правилах «пузырька» (Bubble flow control, [3]) и «порядка направлений» (Direction ordered routing, DOR, [4,5]) с использованием битов направлений [5]. Благодаря алгоритму First Step/Last Step «нестандартного первого и последнего шага» [5] аппаратно поддерживается обход отказавших узлов или линков. Эффективность этого метода по поддержанию связности в сети с отказами была показана в статье [6].

В настоящий момент для сети Ангара разрабатываются алгоритмы по выделению ресурсов при условии отсутствия пересечения сетевого трафика различных заданий, при этом особое значение имеет проблема фрагментации, возникающая в результате последовательного выделения вычислительных узлов.

Для сетей с тороидальной топологией существует несколько стратегий выделения ресурсов [7]. Возможно разделение вычислительной системы на партиции, по которым размещаются задачи пользователей. Такая стратегия может снижать эффективность использования кластера из-за выделения большего числа узлов, чем требовалось, или невозможности выделить доступный набор узлов из разных партиций. Данная стратегия использовалась в суперкомпьютерах IBM Blue Gene/P, Blue Gene/Q [8, 9], в которых ее недостатки компенсировались большим числом не очень мощных по производительности вычислительных узлов и адекватным выбором размера партиции.

Вторая стратегия используется в серии суперкомпьютеров Cray XT/XE, где расположение выделенных узлов [10] не зависит от топологии. Такой способ выделения ресурсов может привести к деградации производительности ввиду наличия конкурирующего трафика.

Помимо возникающей фрагментации, на эффективное использование ресурсов влияют методы определения порядка запуска пользовательских задач. Такая задача является NP-сложной. Существует множество алгоритмов планирования, направленных на оптимизацию использования вычислительных ресурсов по разным параметрам. FCFS [11] (First Come First Served) — политика обработки очереди заданий в порядке, в котором задания поступили. Такая политика обеспечивает справедливость в отношении порядка поступления заданий, но может снижать утилизацию из-за простаивания ресурсов.

В работе [12] проводится сравнение различных вариантов алгоритмов, таких как: First

Come First Served (FCFS, первым пришел — первым обслужен), Priority Queue (очередь с приоритетами), Shortest Job First (кратчайшая задача первая), Longest Job First (продолжительная задача первая) и других.

Одним из самых эффективных на данный момент алгоритмов многие авторы называют алгоритм Backfill [13] – алгоритм обратного заполнения, который является расширением политики FCFS. Для этого алгоритма необходимо наличие оценки о времени выполнения каждого задания. В алгоритме состояние системы по мере завершения работы запущенных заданий сопоставляется с очередью заданий. В статье рассматриваются консервативный вариант алгоритма, когда не допускается выполнения задания, если это повлияет на время запуска приоритетного задания, и агрессивный вариант, позволяющий запустить задание, если это не изменит времени запуска запланированного приоритетного задания.

При большом числе пользователей возникает задача справедливого распределения ресурсов, то есть избегания ситуации, когда один пользователь захватит все вычислительные ресурсы на длительное время. Различные варианты алгоритмов справедливого распределения ресурсов реализованны в программных системах управления ресурсами: PBS, Torque, MAUI, SGE, MBC-1000, SLURM.

Данная статья является продолжением работы [14], в данной работе к методу выделения узлов с оценкой фрагментации добавлена возможность перестановки пользовательских заданий в очереди. Примененный алгоритм перестановки заданий в очереди основан на политике First-Fit (выбора первого подходящего задания); в алгоритме возможность перестановки ограничивается некоторым рассматриваемым окном заданий.

Стоит отметить, что для решения задачи планирования иногда используются генетические алгоритмы, например, в [15], однако предварительные результаты авторов данной статьи показывают, что использование данного механизма ведет к слишком большому времени работы процедуры выбора узлов.

Также в данной работе по сравнению с предыдущей работой авторов модифицирован принцип формирования очереди пользовательских заданий, который стал более приближенным к реальности.

Статья организована следующим образом. В разделе 2 приводятся необходимые формальные определения и постановка задачи. В разделе 3 описаны разработанные алгоритмы. В разделе 4 проведено исследование построенных алгоритмов.

2. Определения и постановка задачи

В данном разделе приводятся формальные определения, которые в дальнейшем будут использоваться в статье.

Рассмотрим вычислительную систему, узлы которой объединены в тороидальную топологию. Размерности тора обозначим $(d_1, d_2, ..., d_n)$, а множество всех узлов вычислительной системы обозначим $N = \{u | u = (u_1, ..., u_n), \forall i\ u_i \in \mathbb{Z}_{d_i}\}$, а общее число узлов — $|N|$. Расстояние на множестве N определим следующим образом: $L(u, v) = \sum_{i=1}^{n} |u_i - v_i|,\ \forall u, v \in N$.

Состояние системы S можно описать множествами узлов, доступных и недоступных для выделения, обозначим эти множества N_{free} и N_{locked}, соответственно.

Будем называть *маршрутизируемым* множеством узлов в коммуникационной сети Ангара такое множество, что для каждого узла этого множества существует сетевой маршрут в любой другой узел множества, удовлетворяющий правилам маршрутизации сети Ангара, а также весь сетевой трафик узлов множества не выходит за его пределы.

Будем называть *заданием* W – число узлов W_{nodes}, запрашиваемое пользователем в момент времени W_{start} на время W_{time}, а *ресурсами для задания* – маршрутизируемое множество узлов, размер которого не меньше, чем W_{nodes}. *Потоком заданий* назовем множество различных заданий W.

Ранее в работе [16] авторами статьи решалась проблема поиска маршрутизируемого

множества заданного размера в коммуникационной сети Ангара с учетом топологии и маршрутизации. Обозначим алгоритм, решающий эту проблему как $Find\_Systems(W,\ S)$. На вход этому алгоритму подается состояние системы S и задание W с требуемым числом вычислительных узлов W_{nodes}. Результатом работы алгоритма является набор вариантов ресурсов для задания. Необходимо заметить, что особенностью алгоритма является то, что все ресурсы для задания представляют собой многомерные прямоугольники.

Под *утилизацией* ресурсов U вычислительного кластера будем понимать среднее значение утилизации по всем вычислительным узлам:

$$U = \frac{\sum\limits_{i=1}^{|N|} U_i}{|N|}, U_i = \frac{T_i}{T},$$

где U_i – утилизация i-го вычислительного узла, T – время работы вычислительного кластера, T_i – полезное время работы i-го вычислительного узла.

Обозначим *значение времени нахождения задания в очереди относительно запрошенного времени* как $T^i_{delay} = \frac{Q^i}{W^i_{time}}$, где W^i_{time} – запрошенное время для задания W^i, Q^i – время ожидания задания W^i в очереди. За *среднее значение времени нахождения задания в очереди относительно запрошенного времени задания* примем $T_{mean} = \frac{\sum_{i=1}^{k} T^i_{delay}}{k} = \frac{1}{k}\sum_{i=1}^{k} \frac{Q^i}{W^i_{time}}$, где k – число различных заданий в потоке.

За оценку качества решения для потока пользовательских заданий возьмем утилизацию ресурсов вычислительного кластера и среднее значение времени нахождения задания в очереди относительно запрошенного времени. Эти характеристики используются по аналогии с работой [17].

Во введенных обозначениях проблема, которую решает данная статья, будет формулироваться следующим образом. Для заданного вычислительного кластера и последовательности заданий $Q = W^1, ..., W^k$ требуется найти ресурсы для всех заданий из последовательности, которые будут максимизировать утилизацию вычислительного кластера и минимизировать среднее значение времени нахождения задания в очереди относительно запрошенного времени.

3. Алгоритм выбора узлов

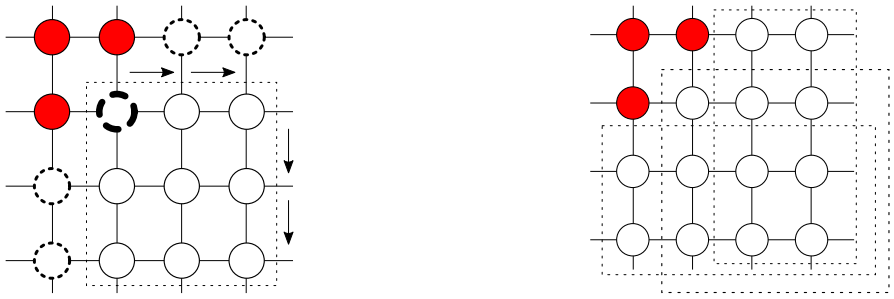
Задача упаковки контейнера является NP-полной задачей. В данной статье представлен алгоритм выбора узлов, основанный на методах, предложенных в работе [18], посвященной трехмерной упаковке контейнера. Идея алгоритма выбора узлов заключается в расположении задания таким образом, чтобы максимизировать оставшееся пространство в многомерном торе. Этот алгоритм предложен в работе авторов [14], однако для удобства восприятия приведен в данном тексте.

3.1. Алгоритм построения прямоугольников максимального размера

Назовем *прямоугольником максимально возможного размера* MSS ($MaxSpaceSize$) многомерный прямоугольник, состоящий только из узлов N_{free}, который нельзя расширить ни в одну из его сторон. Расширить прямоугольник может быть невозможно по двум причинам – либо по соответствующему измерению тора достигнуто максимальное количество узлов в кольце (расширять некуда), либо сторона прямоугольника граничит с узлом из множества N_{locked}. Множество различных прямоугольников MSS характеризуют меру фрагментированности системы.

Алгоритм поиска различных прямоугольников MSS ($Find\_MSSs(S)$) реализован следующим образом. Из множества N_{free} выбирается узел $u_1 \in N_{free}$. Выбранный узел последовательно расширяется во все стороны, пока это возможно. Полученное множество узлов

обозначим MSS_1. На следующем этапе выбирается узел $u_2 \in N_{free} \backslash MSS_1$ и аналогичным образом строится множество MSS_2. Алгоритм продолжается до тех пор, пока множество $N_{free} \backslash \bigcup_{iter=1}^{Iters} MSS_{iter}$ не пусто, где $Iters$ – число итераций алгоритма. Псевдокод алгоритма представлен на рисунке 2. Обозначим множество различных MSS_{iter}, как $MSSs$. Важно отметить, что каждый прямоугольник строится независимо от остальных прямоугольников, в предположении доступности всех изначально свободных узлов N_{free}.



(a) Построение прямоугольника максимально возможного размера

(b) Все прямоугольники максимально возможного размера, построенные алгоритмом

Рис. 1. Выделение прямоугольников максимального размера

```
Input:
    S -- массив, характеризующий состояние системы
    S[u], может принимать следующие значения: 0 - free, 1 - locked, 2 - discovered
Output:
    MSSs -- массив прямоугольников максимального размера
Find_MSS(S)
{
    Iter = 1
    dirs -- массив доступных направлений в торе, например, +x,-x,+y,-y...
    MSSs -- результирующий массив, изначально пуст
    for u in S {
        if S[u] == free {
            MSSs[Iter].push_back(u)
            for dir in dirs {
                MSSs[Iter].extend(dir)
            }
            for v in MSSs[Iter] {
                S[v] = discovered
            }
            Iter++
        }
    }
    return MSSs
}
```

Рис. 2. Псевдокод алгоритма Find_MSS поиска прямоугольников максимального размера

Иллюстрация работы алгоритма приведена на рисунках 1a и 1b, на которых в двумерной решетке узлы множества N_{locked} закрашены, а свободные узлы N_{free} – нет. Жирным

контуром на рисунке 1a выделен узел, из которого поочередным расширением построен двумерный прямоугольник, который обозначен пунктирной линией. Узлы, выделенные полужирным пунктиром, соответствуют множеству узлов N_{free}, не входящих в построенный прямоугольник. Из этих узлов будут строиться последующие прямоугольники. Все построенные прямоугольники MSS показаны на рисунке 1b.

3.2. Оценка состояния вычислительной системы на основе прямоугольников максимального размера

Для оценки состояния вычислительной системы предложена функция φ, которая тем больше, чем большее число прямоугольников максимального размера имеется в системе:

$$\varphi(S) = N * MSS_{max}^{nnodes} + |MSS_{max}|,$$

где MSS_{max} – множество прямоугольников максимального размера, имеющих наибольшее число узлов MSS_{max}^{nnodes}, $|MSS_{max}|$ – число таких прямоугольников, S – текущее состояние системы.

Эта метрика была добавлена в алгоритм $Find\_Systems(W, S)$ поиска маршрутизируемого множества заданного размера. Для каждого найденного маршрутизируемого множества оценивается значение функции $\varphi(S')$, где S' – состояние вычислительной системы S после выделения узлов. Для увеличения утилизации вычислительного кластера требуется выбирать решения с наибольшим значением функции φ. Модифицированный алгоритм $Find\_Systems(W, S)$, в котором возможные варианты систем отсортированы с учетом значения функции φ, в дальнейшем будем обозначать $Find\_Systems_{MSS}(W, S)$.

3.3. Первоначальный алгоритм выбора узлов для кластеров с сетью Ангара

Алгоритм, который изначально работал на кластерах с сетью Ангара, устроен следующим образом. Для всего кластера строится таблица маршрутизации [16]. Для требуемого числа узлов W_{nodes} и допустимого числа транзитных узлов $N_{transit}$ строятся всевозможные разложения чисел $W_{nodes}, W_{nodes} + 1, ..., W_{nodes} + N_{transit}$ на n множителей, таких что $1 \leqslant p_i \leqslant d_i, \forall i \in 1..n$, где p_i – множитель разложения. Все такие разложения обозначим D. Эти разложения описывают всевозможные размеры прямоугольников, подходящих под решение задачи W. Средним диаметром прямоугольника, соответствующего разложению $D_j \in D$, назовем среднее арифметическое всех расстояний между узлами прямоугольника: $\frac{\sum_{u,v \in D_j, u!=v} L(u,v)}{|D_j|}$.

Следующий этап выбора узлов – поиск множества узлов вычислительного кластера, которое можно покрыть одним из найденных прямоугольников таким образом, чтобы в покрытии присутствовали только узлы из множества N_{free}, то есть доступные для выделения. Поиск покрытия начинается с разложений с наименьшим средним диаметром. При первом найденном решении алгоритм заканчивает свою работу.

4. Экспериментальное исследование

4.1. Симулятор вычислительного кластера

Для оценки утилизации ресурсов вычислительного кластера разработан симулятор очереди задач (заданий) и модель состояния кластера. На вход симулятору подается поток пользовательских задач $Q = W^1, ..., W^k$. На выходе выдается полное время работы всего кластера T, время работы каждого узла T_i и время предоставления ресурсов для каждого задания. Используя эти данные, можно вычислить утилизацию ресурсов вычислительного кластера U и среднее значение времени нахождения задания в очереди T_{mean}.

Введем некоторые формальные определения, необходимые для описания работы симулятора. *Очередью* симулятора Q_{now} назовем набор заданий из потока, для которых не выделялись ресурсы и время их запуска T_{start} меньше текущего симулируемого времени t. *Окном заданий* Q_{window} *размера* w назовем некоторое множество заданий таких, что $\forall W^i \in Q_{window}, i - i_{min} < w$, где i_{min} – минимальный индекс задания из множества Q_{now}. *Временем занятости узла* u системы S, назовем время, на которое узел u был выделен для некоторого задания W. В начальный момент времени $t = 0$ время занятости всех узлов равно 0. Операцией *выделения набора узлов* на время T_{alloc} назовем увеличение времени занятости для этих узлов на время T_{alloc}. *Временем изменения системы* T_S назовем время, через которое освободится хотя бы один из выделенных узлов. *Временем изменения очереди* T_{queue} назовем время, через которое хотя бы одно задание перейдет из потока заданий в очередь симулятора. Тогда *временем ожидания симулятора* T_{sleep} назовем минимальное время до изменения состояния симулятора: $T_{sleep} = min(T_S, T_{queue})$.

Алгоритм работы симулятора устроен следующим образом. Если окно заданий не пусто, симулятор выполняет процедуру поиска маршрутизируемого множества для каждого задания из окна по очереди. Если удалось найти решение, то симулятор выделяет найденные ресурсы на необходимое время, а также удаляет это задание из очереди. Если решение не было найдено, то симулятор выполняет процедуру поиска для следующего задания из окна. Если ни одно решение ни для одного задания из окна не было найдено, время симулятора сдвигается на время ожидания T_{sleep}, а время занятости каждого занятого узла u системы S уменьшается на T_{sleep}. Если очередь заданий пуста, а все узлы перешли в состояние свободных, то симулятор завершает свою работу.

4.2. Результаты исследования

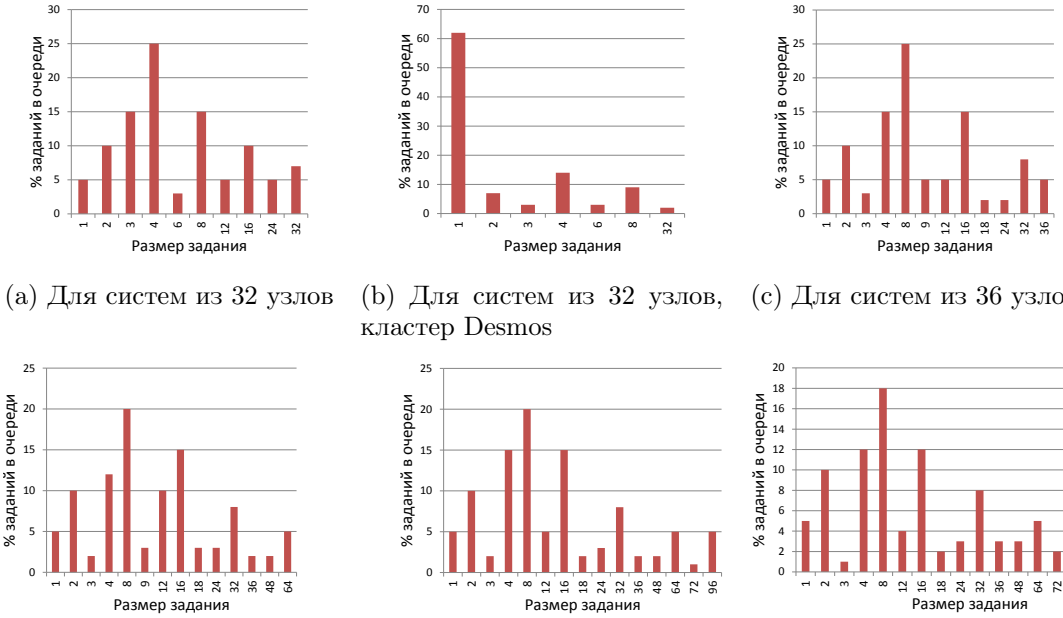
Исследование разработанного алгоритма проводилось на симуляторе для вычислительных систем, представленных в таблице 1.

Таблица 1. Моделируемые системы

Количество узлов вычислительной системы	Топология 3х-мерный тор	Топология 4х-мерный тор
32	4x4x2	4x2x2x2
36	4x3x3	3x3x2x2
64	4x4x4	4x4x2x2
96	6x4x4	4x4x3x2
144	8x6x3	4x4x3x3

Поток заданий для каждой из систем характеризуется вероятностью появления задания для каждого числа узлов от 1 до максимального. Распределение таких вероятностей представлено на рисунке 3. На рисунке 3b представлено реальное распределение пользовательских задач (заданий) по количеству узлов, полученное с гибридного суперкомпьютера Desmos на базе сети Ангара, имеющую топологию 4х-мерный тор 4x2x2x2 [22]. В дальнейшем системы с данной очередью будем помечать символом s – 4x2x2x2s и 4x4x2s. Остальные распределения долей заданий по количеству узлов – синтетические, основанные на предположении о том, что чаще всего встречаются заданий с требуемым числом узлов, равным степеням двойки. Вероятности для остальных чисел узлов равны 0.

Задания равномерно случайно размещаются на временной шкале в диапазоне $\left[0; 60^2 * 24 * 30\right]$ для распределений полученных на кластере Desmos и на диапазоне $\left[0; 4 * 60^2 * 24 * 30\right]$ для



(a) Для систем из 32 узлов (b) Для систем из 32 узлов, кластер Desmos (c) Для систем из 36 узлов

(d) Для систем из 64 узлов (e) Для систем из 96 узлов (f) Для систем из 144 узлов

Рис. 3. Распределение заданий по размеру для различных систем

остальных распределений вне зависимости от числа требуемых узлов.

Время продолжительности заданий соответствует распределению представленному на рисунке 4, полученному в результате анализа запускаемых заданий на кластере Desmos. По оси y представлено отношение требуемого времени для задания к максимальному времени задания. На кластере Desmos максимальное время задания ограничено одними сутками. По оси x представлено процентное отношение заданий. Распределение разбито на две составляющие: на интервале $[0; 90]$ представляет из себя логарифм от линейной функции, такой что в точке 0 оно принимает значение $0,01$, а в точке 90 – значение 99; на интервале $[90; 100]$ представляет линейную функцию и принимает значения от 99 до 100.

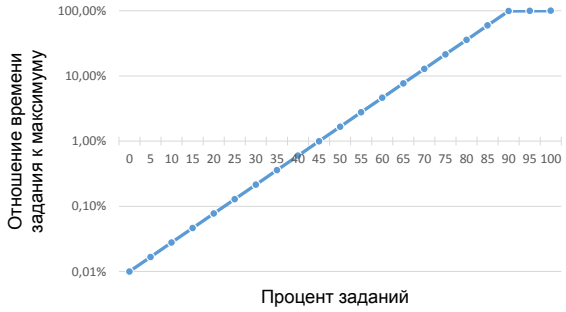
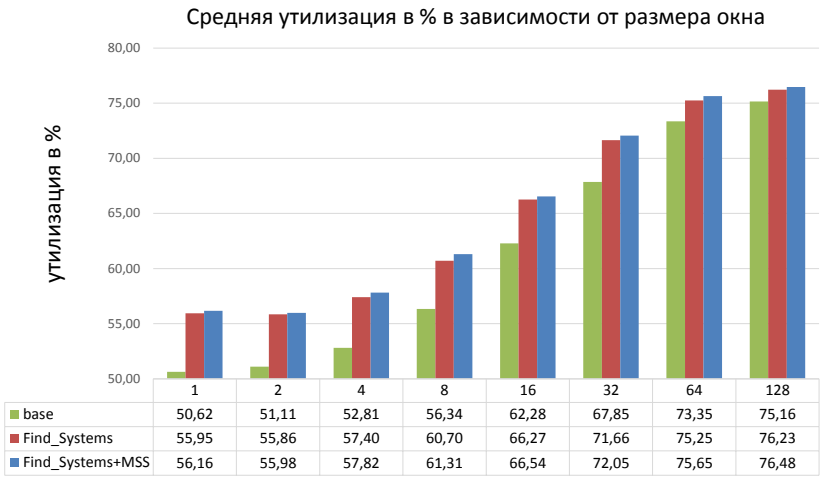


Рис. 4. Распределение времен заданий

Для исследования разработанного алгоритма поиск ресурсов для заданий производился тремя различными способами: методом $Find\_Systems$ без применения разработанной метрики ($Find\_Systems$); методом $Find\_Systems_{MSS}$ с применением разработанной метрики ($Find\_Systems + MSS$); методом, который изначально функционировал на кластерах с сетью Ангара ($base$);

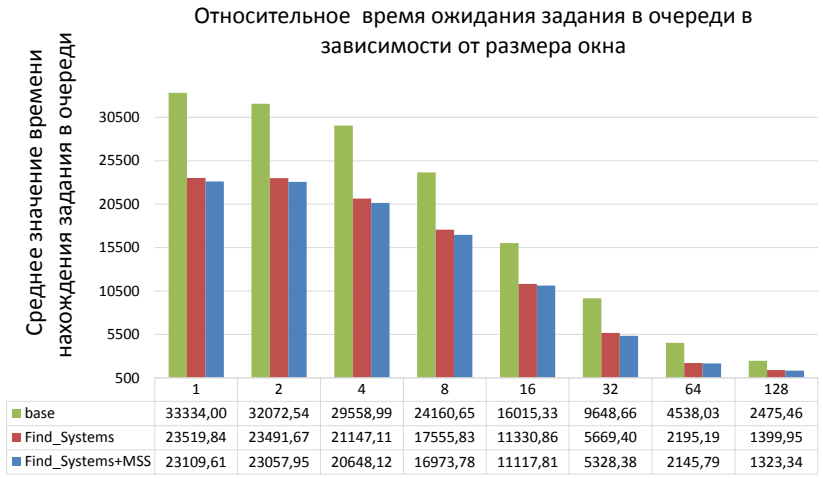
Исследования проводились на окнах заданий размера $1, 2, 4, 8, 16, 32, 64, 128$.

На рисунке 5 представлено среднее значение утилизации вычислительного кластера по всем системам в зависимости от размера окна. Разработанный алгоритм с применение



	1	2	4	8	16	32	64	128
base	50,62	51,11	52,81	56,34	62,28	67,85	73,35	75,16
Find_Systems	55,95	55,86	57,40	60,70	66,27	71,66	75,25	76,23
Find_Systems+MSS	56,16	55,98	57,82	61,31	66,54	72,05	75,65	76,48

Рис. 5. Сравнение утилизации вычислительного кластера для различных систем, методов поиска и размеров окон



	1	2	4	8	16	32	64	128
base	33334,00	32072,54	29558,99	24160,65	16015,33	9648,66	4538,03	2475,46
Find_Systems	23519,84	23491,67	21147,11	17555,83	11330,86	5669,40	2195,19	1399,95
Find_Systems+MSS	23109,61	23057,95	20648,12	16973,78	11117,81	5328,38	2145,79	1323,34

Рис. 6. Сравнение средних значений времени нахождения задания в очереди относительно запрошенного времени для различных систем, методов поиска и размеров окон

разработанной метрикой оценки фрагментированности в данных условиях в среднем дает увеличение на $0,5\%$ относительно алгоритма без учета фрагментированности системы и в среднем на 7% относительно базового. С ростом размера окна утилизация во всех экспериментах увеличивается.

На рисунке 6 представлено среднее значение времени нахождения задания в очереди по всем системам в зависимости от размера окна. Метод $Find\_Systems + MSS$ в среднем дал прирост на 2% относительное метода $Find\_Systems$ и на $36,6\%$ относительно $base$. С ростом размера окна значение времени ожидания задания в очереди в среднем значительно сокращается.

Заключение

В данной работе представлен метод оценки фрагментированности вычислительной системы и показанна возможность применения его в алгоритмах поиска ресурсов для пользовательских заданий. Разработан метод генерации пользовательских заданий. А также показана эффективность использования изменения порядка заданий для вычислительных систем с топологией многомерный тор.

Проведено экспериментальное исследование разработанного алгоритма для различных конфигураций вычислительных систем с топологией многомерный тор с общим числом узлов 32, 36, 64, 96 и 144, при этом рассмотрены 3х-мерные и 4х-мерные конфигурации топологий. Были рассмотрены различные варианты размера окна заданий.

Разработанный метод с учет фрагментированности системы в среднем дает прирост утилизации 7% и на 36,6% сокращает значение время ожидания задания в очереди по сравнению с базовым методом.

Добавление возможности изменения порядка заданий увеличивает утилизацию вычислительных ресурсов, время ожидания задания в очереди сокращается. Однако условия применения этого механизма требуют дальнейшего исследования.

Литература

1. Агарков А.А., Исмагилов Т.Ф., Макагон Д.В., Семенов А.С., Симонов А.С. Результаты оценочного тестирования отечественной высокоскоростной коммуникационной сети Ангара // Суперкомпьютерные дни в России: Труды международной конференции (26–27 сентября 2016 г., г. Москва). М.: Изд-во МГУ, 2016. С. 626–639.

2. Симонов А.С., Макагон Д.В., Жабин И.А., Щербак А.Н., Сыромятников Е.Л., Поляков Д.А. Первое поколение высокоскоростной коммуникационной сети «Ангара» // Наукоемкие технологии. 2014. Т. 15. №1. С. 21–28.

3. Puente V., Beivide R., Gregorio J.A., Prellezo J.M., Duato J., Izu C. Adaptive Bubble Router: a Design to Improve Performance in Torus Networks // Proceedings of the International Conference Parallel Processing (ICPP). 1999. P. 58–67. DOI: 10.1109/ICPP.1999.797388.

4. Adiga N.R., Blumrich M., Chen D. Blue Gene/L Torus Interconnection Network // IBM Journal of Research and Development. 2005. Vol. 49. No. 2. P. 265–276. DOI: 10.1147/rd.492.0265.

5. Scott S.L. The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus. 1996.

6. Пожилов И.А., Семенов А.С., Макагон Д.В. Алгоритм определения связности сети с топологией «многомерный тор» с отказами для детерминированной маршрутизации // Программная инженерия. 2015. № 3. С. 13–19.

7. Lan Z., Tang W., Wang J., Yang X., Zhou Z., Zheng X. Balancing Job Performance with System Performance via Locality-aware Scheduling on Torus-connected Systems // 2014 IEEE International Conference on Cluster Computing (CLUSTER). 2014. P. 140–148. DOI: 10.1109/CLUSTER.2014.6968751.

8. IBM Redbooks Publication: IBM System Blue Gene Solution: Blue Gene/Q System Administration. 2013. 282 p.

9. Tang W., Lan Z., Desai N., Buettner D., Yu Y. Reducing Fragmentation on Torus-Connected Supercomputers // In Proceedings of the 2011 IEEE International Parallel Distributed Processing Symposium (IPDPS'11). IEEE Computer Society, Washington, DC, USA. 2011. P. 828–839 DOI: 10.1109/IPDPS.2011.82.

10. Cray Document: Managing System Software for Cray XE and Cray XT Systems. 2010.

11. Schwiegelshohn U., Yahyapour R. Analysis of First-Come-First-Serve Parallel Job Scheduling //SODA. 1998. Vol. 98. P. 629–638.

12. Полежаев П.Н. Исследование алгоритмов планирования параллельных задач для кластерных вычислительных систем с помощью симулятора // Параллельные вычислительные технологии (ПАВТ'2010): Труды международной конференции. Челябинск: Изд. ЮУрГУ. 2010. С. 287-298.

13. Mu'alem A.W., Feitelson D.G. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling // IEEE Transactions on Parallel and Distributed Systems. 2001. Vol. 12. No. 6. P. 529–543.

14. А.В. Мукосей, А.С. Семенов Оптимизация фрагментации при выделении ресурсов для высокопроизводительных вычислительных систем с сетью Ангара (с. 310-318) // ПаВТ'2018, 2–6 апреля 2018 г., г. Ростов-на-Дону

15. Woo S.H. Task Scheduling in Distributed Computing Systems with a Genetic Algorithm // High Performance Computing on the Information Superhighway. 1997. HPC Asia'97. IEEE. 1997. P. 301–305.

16. Мукосей А.В., Семенов А.С., Приближенный алгоритм выбора оптимального подмножества узлов в коммуникационной сети Ангара с отказами // Вычислительные методы и программирование. 2017. Т. 18. С. 53–64.

17. Баранов А.В., Киселёв Е.А., Ляховец Д.С. Квазипланировщик для использования простаивающих вычислительных модулей многопроцессорной вычислительной системы под управлением СУППЗ // Вестник ЮУрГУ. Серия: Вычислительная математика и информатика. 2014. Т. 3. №4. С. 75–84. DOI: 10.14529/cmse140405.

18. Gonçalves J.F., Resende M.G.C. A Parallel Multi-population Biased Random-key Genetic Algorithm for a Container Loading Problem // Computers & Operations Research. February 2012. Vol. 39. No. 2. P. 179–190. DOI: 10.1016/j.cor.2011.03.009.

19. Аладышев О.С., Киселёв Е.А. Алгоритм эффективного размещения программ на ресурсах многопроцессорных вычислительных систем // Программные продукты и системы. 2012. №4. С. 18–25.

20. Полежаев П.Н. Симулятор вычислительного кластера и его управляющей системы, используемый для исследования алгоритмов планирования задач // Вестник ЮУрГУ. Серия: Математическое моделирование и программирование. 2010. № 6. С. 79–90.

21. Sharma D.D., Pradhan D.K. A Fast and Efficient Strategy for Submesh Allocation in Mesh-connected Parallel Computers // In Procs. of the 5th IEEE Symp. on Parallel and Distributed Processing. 1993. P. 682–689. DOI: 10.1109/SPDP.1993.395466.

22. Dlinnova E., Smirnov G., Stegailov V., Biryukov S., Kondratyuk N. Hybrid Supercomputer Desmos with Torus Angara Interconnect: Performance and Efficiency Optimisation // 12th International Conference, PCT 2018, Rostov-na-Donu, Russia, April 2–6, 2018.