Исследование алгоритмов размещения задач на кластере с учетом топологий задачи и системы

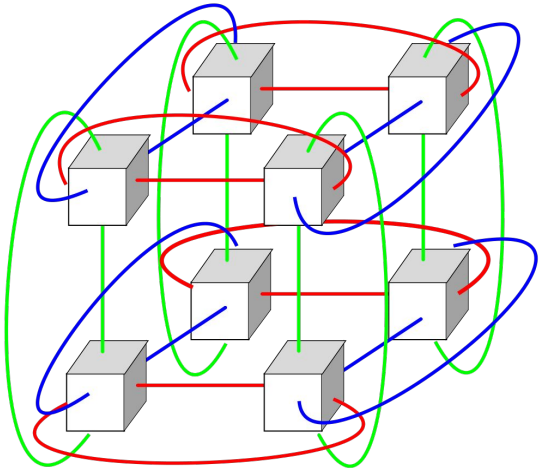
Авторы: Жданов Д. С, Корхов В. В.

Топологии кластера. Сетчатые топологии

В рамках данных топологий каждое ребро решетки параллельно одной из оси и связывает два смежных узла вдоль этой оси.

Примеры данных топологий:

- Плоская решетка
- N-мерный куб
- N-мерный тор



3-х мерный тор

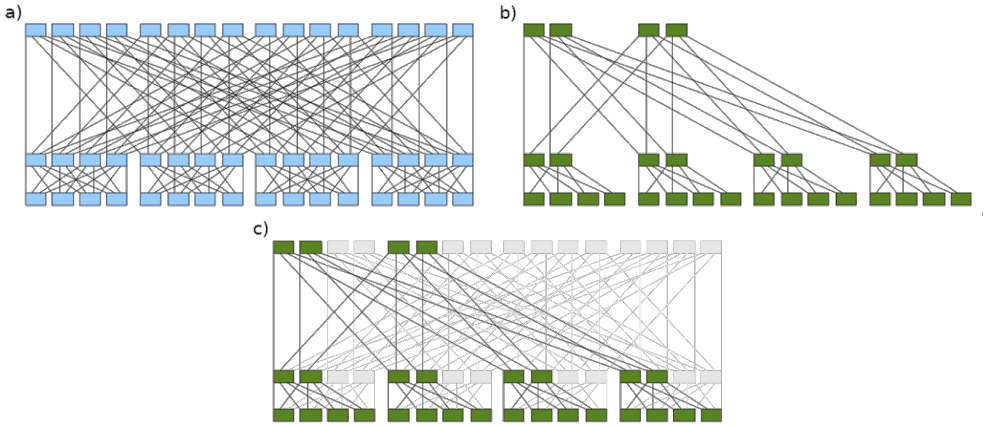
Топологии кластера. Иерархические топологии

В рамках данных топологий вычислительные узлы связываются с помощью дерева коммутаторов.

Важные параметры:

- n - количество уровней коммутаторов
- k - количество связей ведущих к коммутаторам на уровне ниже
- k' - количество связей ведущих к коммутаторам на уровне выше

Обычно деревья обозначаются как k:k'-арные n деревья.



a) 4,3-fat-tree b) 4:2,3-thin-tree c) fat-tree и thin-tree

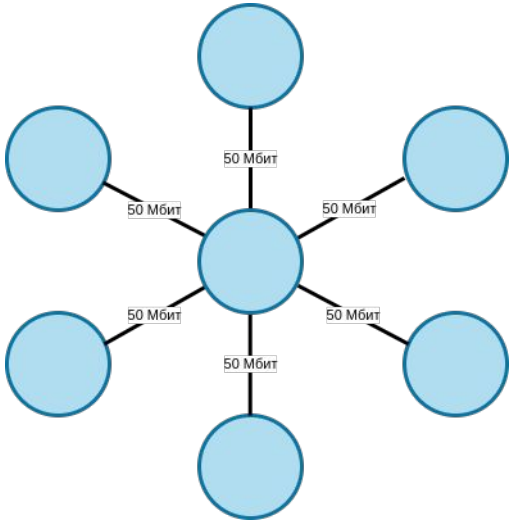
Топологии задач

Топология задачи - взвешенный граф, отображающий схему общения подзадач

Вершины отображают подзадачи

Ребра говорят о том, что соединенные вершины обмениваются информацией

Вес ребра отражает количество информации, которым обменивается данные подзадачи



Звезда

Моделируемые топологии

Топологии кластера:

1) Сетчатые топологии
   a) 2-х мерный тор (torus2d)
   b) 3-х мерный тор (torus3d)
2) Иерархические топологии
   a) толстое дерево (fatTree)
   b) тонкое дерево (thinTree)

Топологии задач:

1) Звезда (STAR)
2) Плоская решетка (GRID)
3) 3-х мерный куб (CUBE)
4) Бинарное дерево (TREE)

Стратегии размещения

- Простое размещение (Simple) - выбор узлов с учетом нумерации узлов
- Случайное размещение (Random) - узлы выбираются случайным образом
- Оптимальное размещение (Optimal) - выбор узлов с учетом топологии кластера
- Продвинутое размещение (Advanced) - выбор узлов с учетом топологии задачи

Оптимальное размещение

Иерархические топологии

Перебираем снизу все уровни в поисках коммутатора, имеющего запрашиваемое количество вычислительных узлов.

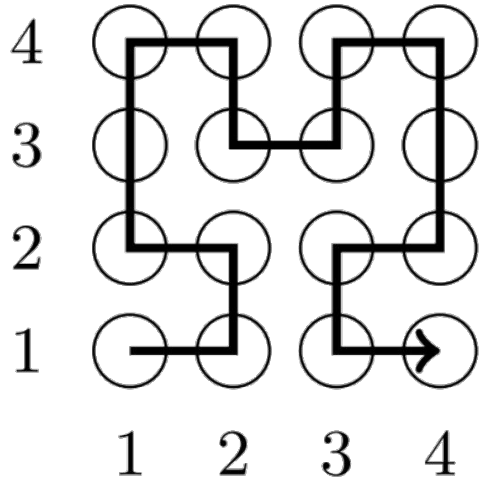
Для улучшения размещения мы на каждом шаге сортируем нижележащие коммутаторы. Это позволяет уменьшить количество используемых коммутаторов.

Оптимальное размещение

Сетчатые топологии

Для отображения на сетчатых топологий необходимо предварительно создать отображение n-мерной топологии кластера на одномерную кривую Гилберта. Данные прямые достаточно плотно заполняют пространство

На следующем этапе необходимо отобразить задачи на данную топологию. Для этого мы находим отрезок минимальной длины, который может вместить задачу

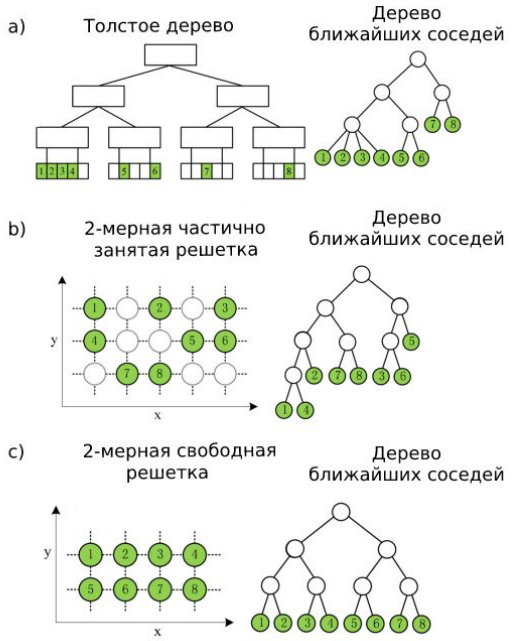


Кривая Гилберта

Продвинутое размещение

Размещение с учетом топологии задачи происходит в три этапа:

1. Выбор узлов - берутся результаты оптимального размещения
2. Построение из узлов дерева ближайших соседей, в котором вычислительные узлы находятся в листьях
3. Отображение топологии задачи на дерево ближайших соседей



Дерево ближайших соседей

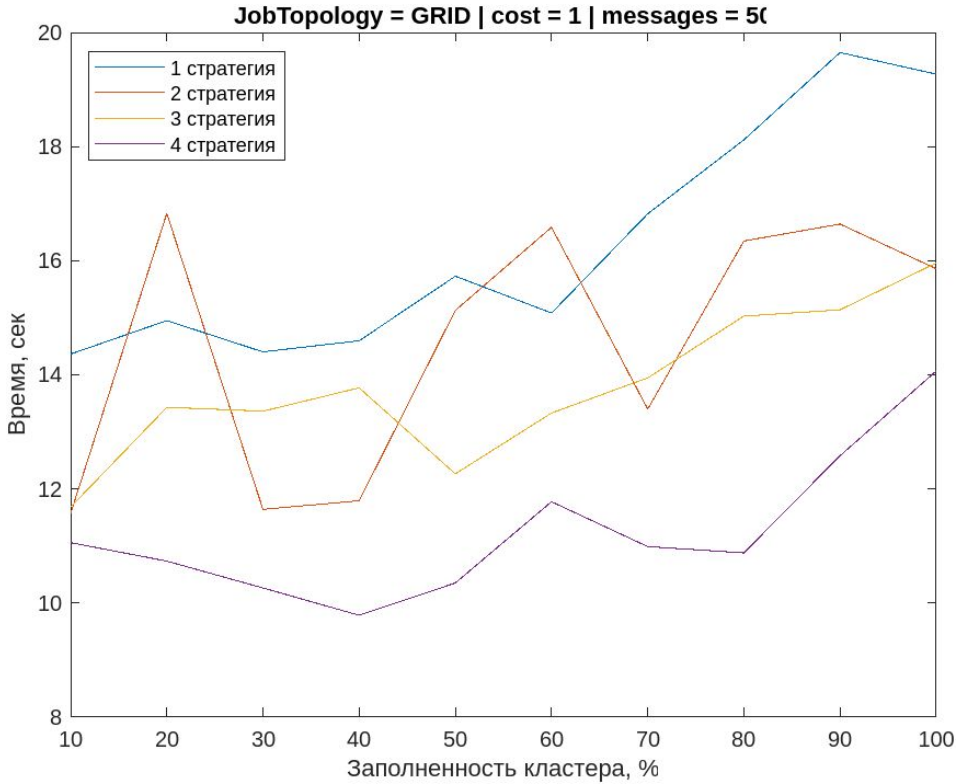
Построение дерева ближайших соседей

Для иерархических топологий дерево ближайших соседей строится на основе связывающих узлы коммутаторов

Для построения дерева ближайших соседей в сетчатых топологиях сравнили 4 стратегии:

1. Построение полного графа и его рекурсивное деление на 2 части
2. Модификация алгоритма Краскала
3. Разделение списка на 2 части, отсортированного по номерам узлов
4. Разделение списка на 2 части, отсортированного по номерам на кривой Гилберта

Лучший результат показала 4 стратегия



Отображение на дерево ближайших соседей

Отображение топологии задачи на дерево ближайших соседей происходит с помощью рекурсивного алгоритма

Мы разбиваем топологию задачи настолько подграфов, сколько поддеревьев у текущего корня дерева ближайших соседей. Разделение графа на подграфы происходит таким образом, чтобы число вершин в соответствующих поддеревьях и подграфах было одинаково и связь элементов из разных подграфов была минимальной

Мы продолжаем разбиение до тех пор, пока не дойдем до вычислительных узлов

Algorithm 1 Recursive Tree Mapping

Input: communication graph $\widehat{G} = (\widehat{V}, \widehat{E})$,
topology tree T.
Output: mapping $\phi : \widehat{V} \rightarrow$ leaves of T.

```
1  tree_mapping(Ĝ, T)
2  {
3    if (root(T) is a leaf) {
4      φ(i) = root(T), ∀ process i ∈ V̂;
5      return;
6    }
7    k = the number of children of root(T);
8    T_i be the subtree rooted at the ith child
     of root(T);
9    // Apply the multilevel k-way partitioning algorithm
10   // [Karypis and Kumar(1998)] to partition Ĝ into
11   // subgraphs Ĝ_i = (V̂_i, Ê_i), 1 ≤ i ≤ k, where
12   // |V̂_i| = the number of leaves of T_i.
13   (Ĝ_1, Ĝ_2, ··· , Ĝ_k) ← graph_partition(Ĝ);
14   for i = 1 to k {
15     tree_mapping(Ĝ_i, T_i);
16   }
17 }
```

Программная реализация

Исходный код приложения:
https://github.com/Zhdanov-Dmitrii/Study_Algorithms_For_Placement_Tasks/tree/master

Используемые технологии:

- C++17
- SimGrid
- METIS

Входные данные

Характеристики кластера:

- 4096 узлов
- Вычислительная мощность 1 Gf
- Пропускная способность 125 MBps
- Задержка 50 us

```
[
  {
    "topologyType": 0,
    "jobs": [
      {
        "jobType": 0,
        "placementMode": 0,
        "processes": 4096,
        "isValid": true,
        "countP2pMessage": 50,
        "cost": 1,
        "path": "output.txt"
      }
    ]
  }
]
```

Проводимые эксперименты

Виды экспериментов:

- Запуск задач на пустом кластере
- Запуск задач на частично заполненном кластере

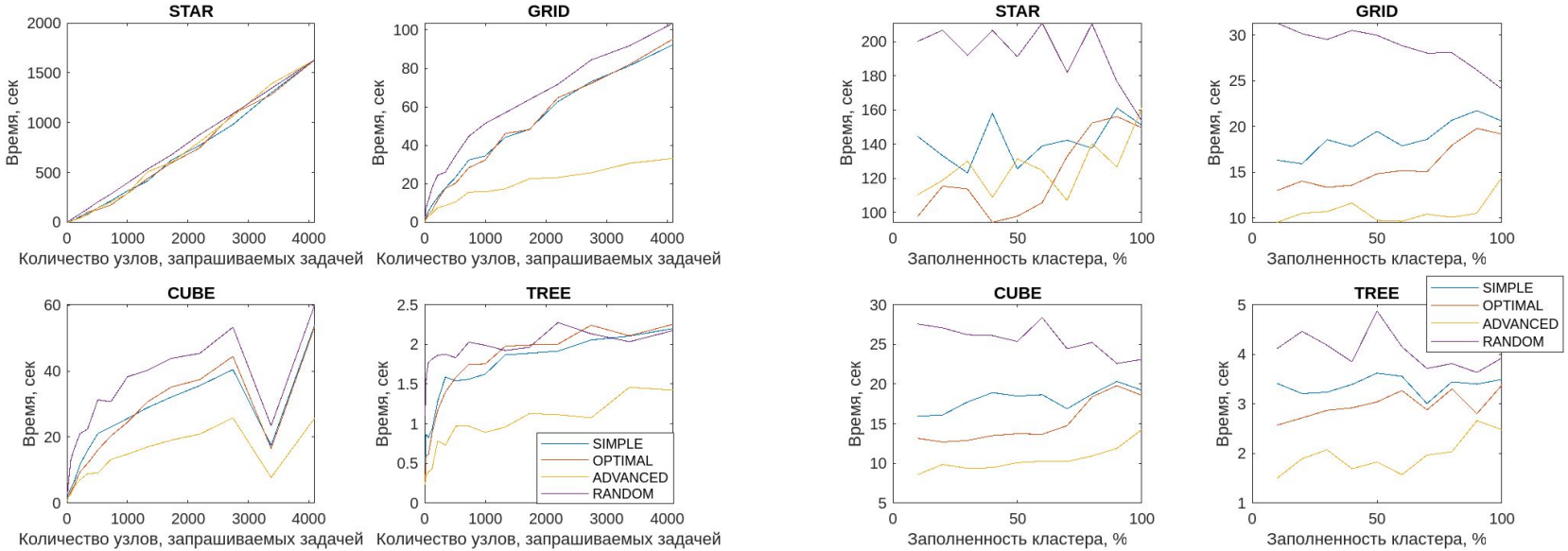
Для моделирования частично заполненного кластера мы перед запуском контрольной задачи загружаем на кластер задачи размером 32 узла с различными топологиями и стратегиями размещения

Параметры запуска:

- С объемом сообщений в 1 байт
- С объемом сообщений 1 мбайт

Сравнение стратегий размещения

Результаты на топологии трехмерного тора с объемом сообщений в 1 байт



Сравнение стратегий размещения

Средние отношения времени работы стратегий размещений к продвинутой стратегии в 1 эксперименте

		Объем сообщений в 1 байт			Объем сообщений в 1 мбайт		
		SIMPLE	OPTIMAL	RANDOM	SIMPLE	OPTIMAL	RANDOM
torus2d	STAR	1.423	1.004	3.302	1.109	0.984	1.59
	GRID	3.341	2.633	6.002	1.748	1.501	2.191
	CUBE	2.298	1.792	5.314	1.573	1.284	1.963
	TREE	2.567	1.891	4.731	2.681	1.79	1.654
torus3d	STAR	1.182	1.012	2.005	1.033	0.989	1.21
	GRID	2.148	1.986	3.334	1.268	1.206	1.391
	CUBE	1.764	1.546	2.871	1.211	1.141	1.328
	TREE	1.868	1.738	2.661	1.887	1.678	1.363
fatTree	STAR	0.974	0.974	1.472	0.99	0.99	1.131
	GRID	1.76	1.206	2.623	1.169	1.031	1.293
	CUBE	1.406	1.016	2.161	1.098	0.982	1.233
	TREE	1.26	1.206	1.927	1.532	1.393	1.325
thinTree	STAR	0.974	0.974	1.472	0.99	0.99	1.131
	GRID	1.756	1.212	2.641	1.242	1.011	1.359
	CUBE	1.426	1.020	2.169	1.252	0.974	1.359
	TREE	1.235	1.210	1.878	3.877	4.482	2.216

Сравнение стратегий размещения

Средние отношения времени работы стратегий размещений к продвинутой стратегии во 2 эксперименте

		Объем сообщений в 1 байт			Объем сообщений в 1 мбайт		
		SIMPLE	OPTIMAL	RANDOM	SIMPLE	OPTIMAL	RANDOM
torus2d	STAR	1.411	1.013	2.252	1.245	1.016	1.636
	GRID	2.422	1.484	4.375	1.656	1.378	2.119
	CUBE	2.161	1.31	3.819	1.512	1.122	2.119
	TREE	2.486	1.907	3.602	2.869	1.98	1.664
torus3d	STAR	1.142	0.968	1.562	1.069	1.032	1.206
	GRID	1.771	1.467	2.730	1.239	1.156	1.334
	CUBE	1.744	1.44	2.499	1.199	1.09	1.277
	TREE	1.768	1.55	2.14	1.923	1.755	1.402
fatTree	STAR	0.921	0.848	1.153	0.966	0.929	1.048
	GRID	1.54	1.088	2.081	1.162	1.017	1.246
	CUBE	1.365	1.025	1.921	1.110	0.991	1.197
	TREE	1.391	1.290	1.72	1.442	1.384	1.23
thinTree	STAR	0.912	0.848	1.153	0.969	0.929	1.048
	GRID	1.521	1.089	2.083	1.137	1.033	1.295
	CUBE	1.37	1.023	1.922	1.176	0.981	1.22
	TREE	1.42	1.306	1.714	2.291	2.862	1.47

Выводы

1. Не для всех топологий задач можно получить прирост производительности используя представленные алгоритмы

2. Использование алгоритмов, учитывающих топологию кластера, могут дать существенный прирост производительности на заполненном кластере

3. Алгоритмы, учитывающие топологию задачи показывают наибольшее ускорение, в том числе и на пустом кластере

4. Представленное ПО можно использоваать для изучения взаимодействий подзадач, топологий и алгоритмоа размещения.

Бонус. Сравнение топологий кластера

Результаты продвинутой стратегии: