АО «Научно-исследовательский центр электронной вычислительной техники»

На правах рукописи

Мукосей Анатолий Викторович

Алгоритмы выбора узлов и построения таблиц маршрутов для высокоскоростной сети с топологией «многомерный тор»

Специальность 2.3.5 —
«Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей»

Диссертация на соискание учёной степени
кандидата физико-математических наук

Научный руководитель:
кандидат технических наук
Семенов Александр Сергеевич

Москва — 2023

Оглавление

Стр.

Стр.

Стр.

Введение

Актуальность

В настоящее время суперкомпьютеры имеют важное значение как для научных и промышленных приложений, так и для обеспечения обороноспособности страны. Производительность суперкомпьютеров растет высокими темпами за счет увеличения количества ядер и применения ускорителей в вычислительных узлах, а для обмена данными и синхронизации между узлами необходима высокоскоростная коммуникационная сеть. Зачастую именно коммуникационная сеть определяет реальную производительность, в особенности на задачах с интенсивным обменом данными в условиях распределенной вычислительной системы.

При создании коммуникационных сетей одной из распространенных топологий являются топологии типа «многомерный тор». Данные топологии используются в суперкомпьютерах IBM Blue Gene/Q, K computer/PRIMEHPC FX10/Fugaku, Sugon, при этом в таких сериях могут создаваться как уникальные суперкомпьютеры из первой десятки списка самых мощных суперкомпьютеров в мире Top500, так и небольшие системы для нужд промышленных организаций.

Сеть Ангара – первая российская высокоскоростная коммуникационная сеть на основе СБИС маршрутизатора. СБИС маршрутизатора коммуникационной сети является разработкой АО «НИЦЭВТ» и выпущен по технологии 65 нм. Сеть поддерживает топологию «многомерный тор» (возможны варианты от 1D- до 4D-тор). В настоящее время существует более десяти высокопроизводительных вычислительных систем от 8 до 92 узлов, использующих сеть Ангара.

Суперкомпьютер – это высокопроизводительная вычислительная система коллективного пользования, поэтому при эксплуатации суперкомпьютеров в условиях наличия отказов (каналов связи или узлов) и узлов, занятых заданиями других пользователей, необходимо предоставлять возможность выделения или выбора для задания пользователя достижимого множества узлов, отвечающего запрашиваемым вычислительным мощностям, а также создавать в этих множествах таблицы маршрутизации.

При этом необходимо учитывать для сетей с топологией «многомерный тор» специфические требования маршрутизации и возможности отказов, требования равномерности распределения (балансировки) сетевого трафика,

минимизации фрагментации, минимизации числа возможных транзитных узлов. В силу противоречивости ряда требований важен выбор разумного компромисса. От качества решения задачи построения сбалансированных таблиц маршрутов зависит производительность при выполнении задания пользователя, а от расширения возможности выбора узлов зависит эффективность использования суперкомпьютера с точки зрения выполнения вычислительными узлами полезной работы.

Степень разработанности темы

Научно-технические решения по выбору узлов и построения таблиц маршрутов в коммуникационной сети с топологией «многомерный тор» в условиях наличия отказов и занятых узлов разработаны недостаточно. Основная причина – различные особенности маршрутизации в сетях с данной топологией создают специфические условия, в которых необходимо решать задачу построения таблиц маршрутов. Также при решении задачи выбора узлов для топологии «многомерный тор» часто используются подходы с избыточностью и необходимо оценивать утилизацию вычислительной системы во время запуска заданий пользователей. Для сети Ангара существовали базовые алгоритмы решения задач построения таблиц маршрутов и выбора узлов, однако они не пригодны к использованию в условиях возможных отказов, не используют все возможности маршрутизации и допускают потерю возможных решений задачи выбора узлов. Все перечисленные аргументы, а также возрастающие требования практики при использовании сети Ангара в научных и промышленных организациях определяют актуальность данной диссертационной работы.

Цель и задачи диссертационной работы

Целью работы является расширение возможности выбора множества узлов в сети Ангара в условиях наличия занятых и отказавших ресурсов.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Разработать алгоритм для анализа маршрутов в сетях с топологией «многомерный тор» с учетом отказавших узлов и каналов связи и ограничений маршрутизации на маршрут сетевого пакета в зависимости от истории его прохождения по сети.
2. Разработать алгоритм определения достижимости множества вычислительных узлов.

3. Разработать алгоритм построения таблицы маршрутов для решения задачи балансировки трафика.
4. Разработать алгоритм выбора множества вычислительных узлов требуемого размера.
5. Реализовать и провести исследование разработанных алгоритмов по сравнению с существовавшими ранее алгоритмами.

Научная новизна

1. Впервые предложен алгоритм построения маршрутного графа для сетей с топологией «многомерный тор» и произвольным количеством отказавших узлов и каналов связи, а также маршрутизацией, накладывающей ограничение на маршрут сетевого пакета в зависимости от истории его прохождения по сети. Данный маршрутный граф позволяет применять алгоритмы обработки графов для анализа маршрутов в коммуникационной сети.
2. Впервые разработаны алгоритмы определения достижимости множества вычислительных узлов, построения таблиц маршрутов и выбора множества узлов для сети Ангара с возможными отказами узлов и каналов связи.
3. В отличие от существовавших ранее алгоритмов разработанные алгоритмы охватывают несколько уровней системного программного обеспечения суперкомпьютера и позволяют на уровне управления ресурсами суперкомпьютера учитывать аппаратные возможности маршрутизации сети Ангара.

Практическая ценность работы

1. Разработанные алгоритмы реализованы в программном дополнении (плагине) ANSU для системы Slurm управления заданиями на вычислительном кластере, которая является частью системного программного обеспечения сети Ангара.
2. Дополнение ANSU в составе системного программного обеспечения используется в семи вычислительных системах, построенных в различных организациях с использованием сети Ангара.
3. Внедрение разработанного дополнения ANSU позволило на суперкомпьютере «Десмос» в ОИВТ РАН повысить утилизацию вычислительных ресурсов на 7,65%.

Методология и методы исследования

При получении основных результатов диссертационной работы использовалась теория графов, методы анализа таблиц маршрутов, имитационное моделирование работы вычислительной системы с использованием синтетических очередей пользовательских заданий. При разработке реализаций предложенных алгоритмов и плагина ANSU для Slurm использовались методы объектно–ориентированного анализа и проектирования на языке C++, для имитационного моделирования – средства прототипирования с использованием языка Python.

Основные положения, выносимые на защиту

1. Разработан алгоритм построения маршрутного графа для анализа маршрутов в высокоскоростных коммуникационных сетях с топологией «многомерный тор» с произвольным количеством отказавших узлов и каналов связи, а также маршрутизацией, накладывающей ограничение на маршрут сетевого пакета в зависимости от истории его прохождения по сети. Временная сложность алгоритма $O(N^2)$, где N – количество узлов в сети.
2. Разработан алгоритм определения достижимости множества вычислительных узлов сети размера N, временная сложность алгоритма $O(N^2)$. Алгоритм использует возможность программного контроля отсутствия дедлоков в сети Ангара, что позволяет сохранять достижимость сети при большем числе случайно отказавших каналов связи (от 5% до 34%) по сравнению с возложением контроля отсутствия дедлоков на аппаратные возможности сети Ангара.
3. Разработан алгоритм построения таблицы маршрутов для решения задачи балансировки трафика в достижимом множестве узлов размера N, временная сложность алгоритма $O(N^2)$.
4. Разработан алгоритм выбора узлов в сети размера N с учетом её фрагментации, временная сложность алгоритма $O(N^4)$. Алгоритм позволил по сравнению с существовавшими ранее алгоритмами от 2 до 12 раз расширить возможности при выборе множества узлов в сети Ангара в зависимости от потока пользовательских заданий и исследуемой системы.

5. Разработанные алгоритмы реализованы и используются в составе системного программного обеспечения десяти вычислительных систем, построенных с использованием сети Ангара.

Степень достоверности и апробация результатов

Основные результаты работы докладывались на конференциях и семинарах:

1. Национальный Суперкомпьютерный Форум, 30 ноября – 3 декабря 2021.
2. Международная конференция «Суперкомпьютерные дни в России», 27–28 сентября 2021.
3. Международная конференция «Суперкомпьютерные дни в России», 23–24 сентября 2019.
4. Международная конференция «Суперкомпьютерные дни в России», 24–25 сентября, 2018.
5. XII Международная научная конференция «Параллельные вычислительные технологии» (ПаВТ'2018), 2–6 апреля 2018.
6. Школа-семинар «Поиск эффективных суперкомпьютерных архитектур в пост-Муровскую эру», МИЭМ НИУ ВШЭ, 11 декабря 2017.
7. Международная конференция «Суперкомпьютерные дни в России», 26–27 сентября 2016.
8. X Международная научная конференция «Параллельные вычислительные технологии» (ПаВТ'2016), 28 марта – 1 апреля 2016.
9. Научный семинар НИВЦ МГУ под руководством В.В. Воеводина.
10. Научный семинар МСЦ РАН под руководством Б.М. Шабанова.
11. Научный семинар ИПМ РАН под руководством М.В. Якобовского.
12. Научный семинар ЮУрГУ под руководством Л.Б. Соколинского.
13. Научный семинар кафедры ИИТ факультета ВМК МГУ под руководством И.В. Машечкина.
14. Научный семинар кафедры АСВК факультета ВМК МГУ под руководством Р.Л. Смелянского.

Личный вклад

Все представленные результаты в диссертационной работе получены лично автором. Подготовка части материалов к публикациям проводилась совместно с соавторами, причем вклад диссертанта был определяющим. В работах [1–9] А.С. Семенову, А.С. Симонову и Д.В Макагону принадлежит постановка задач и консультирование. В работе [1] А.А. Третьякову принадлежит выполнение оценочного тестирования производительности на суперкомпьютере, данные результаты не вошли в диссертационную работу. В работе [10] автору принадлежит реализация алгоритмов выбора узлов в программном дополнении Slurm.

Публикации

Основные положения и выводы диссертационного исследования в полной мере изложены в 11 научных работах [1–11]. 7 работ [1–6; 10] опубликованы в рецензируемых научных изданиях, определенных п. 2.3 Положения о присуждении ученых степеней в Московском государственном университете имени М.В. Ломоносова, 4 из которых [1–3; 10] изданы в журналах, индексируемых Scopus/Web of Science; а 3 другие работы [4–6] изданы в журналах, рекомендованных ВАК при Минобрнауки России.

Объем и структура работы

Диссертация состоит из введения, пяти глав и заключения. Полный объём диссертации составляет 147 страниц, включая 46 рисунков и 13 таблиц. Список литературы содержит 112 наименований.

Глава 1. Анализ проблемы выбора узлов и построения таблиц маршрутов в высокоскоростных сетях с топологией «многомерный тор»

1.1 Высокоскоростные сети с топологией «многомерный тор»

При использовании суперкомпьютеров для решения прикладных задач важнейшим фактором, влияющим на производительность, является коммуникационная сеть, объединяющая вычислительные узлы. Коммуникационная сеть необходима для обмена данными между узлами и для синхронизации параллельных процессов, при помощи работы которых происходит решение прикладной задачи.

Коммуникационная сеть состоит из узлов, в каждом из которых находится сетевой адаптер, соединённый с одним или несколькими маршрутизаторами, которые, в свою очередь, соединяются между собой высокоскоростными каналами связи (линками).

Структура сети, определяющая как именно связаны между собой узлы системы, задаётся топологией сети (обычно используются решётка, тор, жирное дерево или Dragonfly [12]) и набором параметров, в числе которых количество измерений тора, количество уровней дерева, размеры сторон тора или число коммутаторов на уровнях дерева, число узлов сети, портов у коммутаторов и так далее [13].

Топология типа «многомерный тор» представляет из себя узлы, соединенные в многомерную решетку, крайние узлы которой соединены. Примером одномерной топологии тор является кольцо.

К преимуществам топологии тор можно отнести масштабируемость; хорошую производительность на задачах физического 3D-моделирования, которые естественным образом накладываются на топологию тор; избыточность соединений, что повышает отказоустойчивость и увеличивает бисекционную пропускную способность.

Высокоскоростные сети с топологией «многомерный тор» получили широкое распространение и используются в различных суперкомпьютерах. Примерами систем с такими сетями на протяжении последних двадцати пяти лет являются: Cray T3E [14], Cray Seastar [15], Cray Seastar2 [16] и Cray Seastar2+ [17] с топологией 3D-тор; Cray Gemini [18] с топологией 3D-тор; IBM Blue Gene/L [19] и IBM Blue Gene/P [20] с топологией 3D-тор, Tofu [21] и Tofu2 [22] с

топологией 6D-тор. Среди последних разработок сетей можно выделить американскую сеть IBM Blue Gene/Q [23] с топологией 5D-тор, японскую Tofu D [24] с топологией 6D-тор, европейскую EXTOLL с топологией 3D-тор [25], китайскую Sugon с топологией 3D-тор [26].

Рассмотрим подробнее некоторые суперкомпьютеры и их высокоскоростные сети.

IBM Blue Gene. Классическими представителями систем, использующих тороидальную топологию объединения вычислительных узлов, являются системы серии IBM Blue Gene. В первых двух поколениях этих систем Blue Gene/L (2004 год) и Blue Gene/P (2007 год) использовалась топология 3D-тор.

Сеть 3D-тор в Blue Gene/P имеет относительно слабые линки с односторонней пропускной способностью 0.425 ГБ/с, что на порядок меньше, пропускной способности линка сети Infiniband QDR. Однако аппаратная поддержка барьерной синхронизации и коллективных операций в виде отдельных сетей позволяют достигать эффективной масштабируемости на реальных приложениях. Кроме того, все интерфейсы и блоки маршрутизации встроены в микропроцессор (или «систему-на-кристалле») BPC (Blue Gene/P compute chip), что значительно снижает задержку при передаче сообщений.

Коммуникационная сеть Blue Gene/Q (2011) имеет топологию 5D-тор [27]. В отличие от Blue Gene/P, в Blue Gene/Q нет отдельных сетей для барьерной синхронизации и коллективных операций (все они осуществляются посредством сети 5D-тор). Пропускная способность линка увеличена до 2 ГБ/с, но все равно остаётся небольшой по сравнению с Cray Gemini или EXTOLL. Поскольку каждый узел имеет 10 линков, агрегатная пропускная способность (дуплексная) составляет 40 ГБ/с. Таким образом, низкая пропускная способность нивелируется большой размерностью тора и, как следствие, малым диаметром сети (значительно меньшим, чем у 3D-тор).

Cray Gemini. Другим последователем подхода к построению коммуникационных сетей с тороидальной топологией являлась компания Cray. В отличие от IBM, которая избрала идеологию «большой размерности и слабых линков» для своих новых систем, Cray продолжала использовать топологию 3D-тор, при этом наращивая пропускную способность и количество линков, соединяющих соседние узлы.

Основным отличием Gemini от предшественника SeaStar2+ является то, что один маршрутизатор Gemini соответствует двум маршрутизаторам

SeaStar2+, то есть фактически, двум узлам сети. Поэтому в Gemini вместо 6 линков для соединения с соседними узлами используется 10 (4 линка для направлений X, Z и 2 линка для Y). Сети SeaStar2+ и Gemini использовались в суперкомпьютерах серий Cray XT/XE [28].

Fujitsu Tofu. Японский суперкомпьютер Fugaku, изготовленный компанией, Fujitsu является продолжением серии K Computer, производительность на HPL – 442 Пфлопс, что составляет 82.3% от пиковой (537.3 Пфлопс). Узлы состоят из процессоров Fujitsu A64FX на базе Arm v8-A SVE (собственной разработки Fujitsu), не содержат графических ускорителей и имеют водяное охлаждение. Помимо того, что суперкомпьютер в настоящее время занимает первое место в Top500 (редакция июня 2021 года), он является весьма энергоэффективным (особенно с учётом его масштабов).

В Fugaku используется коммуникационная сеть собственной разработки, названная Tofu D (от TOrus FUsion). Основная идея — большая гибкость при выборе подмножества узлов при сохранении топологии трёхмерный тор. Официально заявляется, что сеть имеет топологию 6D-тор, но на самом деле это не совсем так. Сеть имеет два уровня иерархии. На верхнем уровне — масштабируемый 3D-тор (измерения X, Y, Z). В его узлах — группы (node groups, Tofu unit) по 12 узлов. Узлы каждой группы соединены между собой 3D-тором со сторонами $2 \times 3 \times 2$ без дублирующих линков (измерения A, B, C), что эквивалентно 2D-тору со сторонами 3×4. Таким образом, узел сети Tofu D имеет 10 линков с пропускной способностью в 6.8 ГБ/с. Аппаратно поддерживаются коллективная операция barrier, технология RDMA.

EXTOLL. Сеть EXTOLL – разработка Гейдельбергского университета (точнее, группы компьютерных архитектур по руководством проф. У. Брюнинга). Сеть EXTOLL имеет топологию 3D-тор. EXTOLL имеет ряд отличительных особенностей: во-первых, это поддержка сразу двух интерфейсов с процессором – HyperTransport 3.0 и PCI Express gen3, что позволит тесно интегрировать EXTOLL в рамках платформ как на базе процессоров Intel, так и AMD31; во-вторых, это использование оптических линков с пропускной способность 10 Гбит/с на линию и шириной 12x, что в сумме дает одностороннюю пропускную способность 15 ГБ/с на один линк. Также в EXTOLL поддержаны несколько способов организации односторонних записей, собственный MMU, поддержка атомарный операций.

Ангара. В России также разрабатываются сети с топологией «многомерный тор». Сеть Ангара [29; 30] – первая российская высокоскоростная коммуникационная сеть на основе СБИС маршрутизатора. СБИС маршрутизатор коммуникационной сети является российской разработкой и выпущен в 2013 году по технологии 65 нм. Сеть поддерживает топологию «многомерный тор», возможны варианты от 1D- до 4D-тор.

СМПО, Паутина. В РФЯЦ-ВНИИЭФ разработана система межпроцессорного обмена СМПО-10G [31], основу которой должна составлять заказная СБИС. Информации об этой разработке в открытой печати очень мало. В ИПС им. А. К. Айламазяна РАН разработаны основанные на ПЛИС коммуникационные сети СКИФ-Аврора [32] и Паутина [33], они имеют топологию 3D-тор.

Следующий раздел посвящен подробному описанию маршрутизации в сетях с топологией «многомерный тор».

1.2 Принципы маршрутизации в сетях с топологией «многомерный тор»

Данный раздел посвящен принципам маршрутизации в сетях с топологией «многомерный тор».

Формально маршрутизацию (или алгоритм маршрутизации) можно представить как функцию R и функцию выбора ρ. Функция R возвращает множество возможных путей (или каналов связи), а функция ρ позволяет выбрать один из них и использовать его для пакета. В зависимости от алгоритма маршрутизации функцию R можно представить как [34]:

$$R : N \times N \to \mathcal{P}(P)$$
$$R : N \times N \to \mathcal{P}(C)$$
$$R : C \times N \to \mathcal{P}(C),$$

где N – множество узлов сети, C – множество каналов сети, P – множество всех маршрутов в сети, $\mathcal{P}(X)$ – множество всех подмножеств множества X. Данная нотация позволяет отобразить то, что функция R может возвращать множество маршрутов или каналов сети, один из которых будет выбран функцией ρ.

В первом случае маршрутизация называется all-at-once: когда сетевой пакет инжектируется в сеть в узле-источнике x и отправляется к узлу-получателю y, вычисляется функция $R(x, y)$, которая вместе с функцией ρ позволяет записать маршрут в пакет, который затем проходит по сети заданным маршрутом.

Второй способ представления – так называемая инкрементальная маршрутизация. Вместо определения сразу всего пути в начале движения пакета функция $R(w, y)$ применяется на каждом шаге маршрута пакета в некотором узле w и задает набор возможных каналов на следующего шага маршрута в зависимости от узла-получателя y. Третий способ похож на второй, отличие в том, что на вход функции R подается канал, из которого пришел пакет.

Классифицировать алгоритмы маршрутизации принято следующим образом. Алгоритмы маршрутизации могут быть статическими или адаптивными. Статический алгоритм маршрутизации (англ. oblivious routing) – определение маршрута $R(x, y)$ для каждой пары узлов «отправитель» x и «получатель» y без учета сетевого трафика в сети. Детерминированная маршрутизация – подкласс статической маршрутизации, детерминированный маршрут между двумя заданными узлами всегда один и тот же. Примером маршрутизации, отличающий ее от детерминированной, является случайный выбор маршрута среди набора маршрутов между двумя заданными узлами.

Адаптивные алгоритмы, наоборот, учитывают наличие сетевого трафика. При адаптивной маршрутизации маршрутизатор в зависимости от состояния сети может менять маршрут пакета.

Может показаться, что адаптивная маршрутизация всегда лучше. Однако детерминированная маршрутизация имеет свои преимущества:

- Легко обеспечивается фиксированный порядок доставки пакетов: узел получатель принимает пакеты в том же порядке, в каком они вышли из узла-отправителя. Это главное свойство детерминированной маршрутизации, из-за которого она обязательно присутствует в любой сети. При отправке пакета указывается, требуется ли передавать его только по детерминированному маршруту или допустимо использование адаптивной маршрутизации.
- Алгоритмы детерминированной маршрутизации требуют не только меньше места на кристаллах при их реализации, но и, что особенно важно, расходуют меньше времени на принятие решений, тем самым уменьшая задержку в каждом узле.

Данная работа посвящена исключительно детерминированной маршрутизации в связи с ее первоочередной необходимостью для высокоскоростной сети. В дальнейшем будет рассматриваться только детерминированная маршрутизация.

Одна из основных проблем, с которой приходится иметь дело при разработке сетей и алгоритмов маршрутизации – это возможность тупиковых ситуаций типа «дедлок» и «ливлок». Дедлок (англ. deadlock) – это такое состояние сети, когда группа пакетов не может продолжить движение, поскольку каждому из них для передачи требуется ресурс, занятый другим пакетом. Ресурс – это место в буфере следующего узла, в который дальше должен перемещаться пакет. Данная ситуация происходит из-за циклической зависимости при движения сетевого трафика.

Ливлок (англ. livelock) – состояние сети, в котором трафик не останавливается, но по различным причинам пакет никогда не будет доставлен. Проблема ливлоков обычно решается тем, что применяется минимальная маршрутизация. Маршрутизация называется минимальной, если все пакеты перемещаются по маршрутам минимальной длины, измеренной в количестве хопов (англ. hop – передача пакета между соседними узлами). Тогда на каждом шаге расстояние до узла-получателя уменьшается на единицу, и циклы становятся невозможными. Тем не менее, неминимальная маршрутизация также представляет интерес для повышения отказоустойчивости и адаптивности сети.

Для топологии «многомерный тор» для борьбы с дедлоковыми ситуациями применяется подход, когда решение проблемы разделяется для одномерной маршрутизации и многомерной маршрутизации.

Одномерная маршрутизация является базой для построения многомерной маршрутизации. В случае тора каналы, соединяющие узлы, лежащие на одной оси, образуют кольцо, а в кольце возможны дедлоки. Методы, применяемые для бездедлоковой маршрутизации в кольце: виртуальные каналы и пузырьковая маршрутизация.

Пузырьковая маршрутизация (англ. Bubble flow control, [35]) состоит в том, что после прихода нового пакета в сеть должно оставаться место на еще один пакет. В таком случае движение в кольце всегда возможно.

Другим способом решения проблемы блокировок в кольце сети является использование дополнительных виртуальных каналов. Виртуальный канал – это пара отдельных входных и выходных буферов для каждого из физических

линков, в который маршрутизатор осуществляет прием (отправку) сетевых пакетов. Передача пакетов по нескольким виртуальным каналам реализуется на одном физическом канале так, что пакеты передаются вперемешку. Благодаря этому удается добиться высокой степени утилизации физических каналов ценой некоторого усложнения маршрутизаторов. Виртуальные каналы позволяют разделить высокоскоростную сеть на несколько виртуальных независимых подсетей.

Отсутствие дедлоков при помощи дополнительного виртуального канала можно гарантировать, если при пересечении пакетом определенной границы в одномерном кольце этот пакет перемещать во второй виртуальный канал. Используя по два виртуальных канала для каждого физического канала между соседними маршрутизаторами, можно организовать передачу пакетов так, чтобы каналы, по которым перемещаются пакеты, не образовывали циклов. Идея состоит в том, что кольцо превращается в спираль. Узел, в котором пакеты меняют виртуальный канал, называется dateline («линия перемены дат» — по аналогии с 180-ым меридианом, где время меняется на 1 сутки).

Многомерная маршрутизация надстраивается над одномерной. Для этого часто применяется правило порядка направлений (англ. Direction Order Routing, DOR) [14;36], которое состоит в следующем. Заметим, что в каждом измерении тора по два направления – положительное и отрицательное. Вводится порядок направлений в торе и накладывается ограничение на порядок перемещения пакетов по направлениям. Например, для сети с топологией 3D-тор порядок может быть таким: $+X, +Y, +Z, -X, -Y, -Z$; один пакет может перемещаться по направлениям, скажем, $+X, +Z, -Y$, другой по $+Z, -X, -Y$ и т.п. Каждый пакет перемещается по следующему направлению до тех пор, пока все координаты с целевым узлом не совпадут.

Существует другой метод многомерной маршрутизации, который называется моделью поворотов (turn model) [37]. Чтобы избежать циклов, приводящих к дедлокам, пакетам разрешается делать только часть возможных поворотов. Множество допустимых поворотов для данного пакета вычисляется по адресам отправителя и получателя. Данный метод допускает неминимальную маршрутизацию, которая более отказоустойчива. Существуют различные методы с использованием модели поворотов, когда для предотвращения дедлоков запрещается часть поворотов [38].

Детерминированная маршрутизация может позволять не единственный маршрут, а множество маршрутов между двумя узлами. Часто для выбора конкретного маршрута используется таблица маршрутов, которая может храниться на каждом вычислительном узле. При инжекции пакета в сеть происходит выбор маршрута, по которому детерминированно движется пакет.

В следующих подразделах представлена информация о реализации маршрутизации в различных сетях с топологией «многомерный тор», акцент сделан на детерминированную маршрутизацию.

1.2.1 Правила маршрутизации в сетях IBM Blue Gene/L/P/Q

В IBM Blue Gene/Q нет отдельной сети коллективных операций, как это было в IBM Blue Gene/L/P, поэтому в каждом направлении тора существует виртуальные каналы для операций точка-точка (детерминированный, адаптивный, высокого приоритета, системный) и для коллективных операций (пользовательский и системный). В детерминированной маршрутизации действует правило порядка направлений, которое, впрочем, можно задавать. В остальном детерминированная маршрутизация в IBM Blue Gene/Q устроена также, как и в IBM Blue Gene/L.

В IBM Blue Gene/L [36] в каждом направлении тора имеется 4 виртуальных канала: 2 адаптивных и 2 детерминированных, работающих по правилу пузырька, один обычный, другой – приоритетный. Используется порядок измерений, причем движение пакета и в положительном, и в отрицательном направлении по измерению одновременно запрещено.

Детерминированная маршрутизация позволяет описать несколько маршрутов между каждой парой вычислительных узлов. При совместном использовании вычислительного кластера несколькими заданиями маршруты для детерминированной маршрутизации выбираются таким образом, чтобы не пересекаться между заданиями [39].

При отказе вычислительного узла или канала связи из работы вычислительного кластера исключается midplane, содержащий отказ [40]. Под midplane понимают множество вычислительных узлов размера 512 и имеющих топологию 8x8x8.

1.2.2 Правила маршрутизации в сетях Cray Gemini

Коммуникационная сеть Cray Gemini [18] поддерживает топологию 3D-тор. Каждое направление тора содержит несколько каналов связи. В каждом канале связи имеется по 2 виртуальных канала: один виртуальный канал для запросов и один для ответов. Маршрутизация в сети Gemini основана на правиле порядка измерений [41], пакет маршрутизируется сначала по измерению с высоким приоритетом и затем по измерениям с низким. При построении маршрутов используются только кратчайшие [42]. Из данного утверждения следует, что возможно, что в аппаратно реализованной маршрутизации нет ограничения битов направлений. Адаптивная маршрутизация строится над детерминированной и позволяет выбрать наименее загруженный канал связи в измерении.

В случае наличия отказа в сети необходима перестройка детерминированных маршрутов для обхода отказавшего ресурса сети. Трафик различных заданий может пересекаться.

1.2.3 Правила маршрутизации в сетях Fujitsu Tofu, Tofu 2/D

В суперкомпьютере Fugaku используется коммуникационная сеть собственной разработки, названная Tofu D (от TOrus FUsion) с топологией «многомерный тор» 6D, которая эквивалентна 5-мерному тору, так как у каждого маршрутизатора имеется 10 линков для соединения с соседями.

Во многих аспектах, в том числе и по маршрутизации, сеть TofuD сходна с сетями Tofu и Tofu 2. Маршрутизация осуществляется по порядку измерений: вначале по измерениям A, B, C, затем по X, Y, Z, затем снова по A, B, C. Ограничения битов направлений нет. Это позволяет закладывать неминимальные маршруты для балансировки нагрузки и обхода отказавших узлов. Вероятно, что в каждом направлении имеется 4 виртуальных канала. При совместном использовании вычислительного кластера несколькими заданиями маршруты для детерминированной маршрутизации выбираются таким образом, чтобы не пересекаться между заданиями [22]. Более детальную информацию о маршрутизации найти не удалось.

1.2.4 Правила маршрутизации в сети Ангара

Маршрутизатор высокоскоростной сети Ангара поддерживает топологию «многомерный тор», до 4D-тор. В маршрутизаторе сети реализована бездедлоковая, детерминированная и адаптивная маршрутизации, основанные на правилах «пузырька» (Bubble flow control, [35]) и «порядка направлений» (Direction ordered routing, DOR) с использованием битов направлений (dirbit-маршрутизация). Пузырьковая маршрутизация обеспечивает отсутствие дедлоков в кольце. Правило порядка направлений применяется для бездедлоковой маршрутизации многомерной топологии. Однако dirbit-маршрутизация запрещает движение пакета в разных направлениях одного и того же измерения, например, в $+X$ и $-X$ пакету двигаться запрещено, точно такое же ограничение имеется в сети IBM Blue Gene L/P/Q. Применяемая маршрутизация позволяет избежать взаимных блокировок из-за циклических зависимостей пакетов в кольцах и между кольцами нескольких измерений, а также гарантирует сохранение порядка передачи пакетов между любыми двумя адресатами.

Для расширения возможностей маршрутизации в сети Ангара реализован алгоритм First Step/Last Step (FS/LS) «нестандартного первого и последнего шага» [14], который позволяет сделать первый и последний шаг без ограничений DOR и dirbit-маршрутизации. Например, благодаря FS/LS допустим следующий порядок шагов: $+X(FS), +Y, +Z, -X, -Z(LS)$, где $+X(FS)$ – первый нестандартный шаг, $+Y, +Z, -X$ – шаги согласно dirbit-маршрутизации и последний нестандартный шаг $-Z(LS)$. Эффективность этого метода по поддержанию достижимости узлов в сети с отказами была показана в статье [43].

В сети Ангара адаптивная маршрутизация является надстройкой над детерминированной. В случае, если пакет не может продвинуться по адаптивной подсети, он навсегда переходит в детерминированную подсеть. Однако в диссертационной работе рассматривается только детерминированная маршрутизация.

Таблицы маршрутов

Правила детерминированной маршрутизации в сети Ангара реализованы аппаратно и позволяют множество маршрутов между двумя узлами. Задание конкретного детерминированного маршрута происходит программно путем заполнения таблиц маршрутов, хранящихся в каждом маршрутизаторе сети. Таблица маршрутов состоит из строк, каждая из которых описывает маршрут до определенного узла сети. Таблицу маршрутов можно обновлять при старте вычислительной задачи.

Пакеты в сети передаются по каналам по флитам – 1 флит за (логический) такт. Таким образом, флит – единица передачи данных. При попадании каждого сетевого пакета в маршрутизатор, то есть при инжекции пакета в сеть, формируется головной флит пакета, в который по идентификатору узла назначения из таблицы маршрутов добавляется информация об маршруте пакета. В сети Ангара флит – 128-битное слово.

1.3 Проблема выделения узлов и построения таблиц маршрутов

При эксплуатации суперкомпьютеров на основе сети Ангара в условиях наличия отказов (каналов связи или узлов) и занятых другими задачами вычислительных узлов необходимо предоставлять возможность выделения для задачи пользователя множества узлов, отвечающего запрашиваемым вычислительным мощностям и требованиям маршрутизации (то есть достижимого множества), если это возможно.

Также в настоящий момент в системном программном обеспечении сети Ангара поставлено требование отсутствия пересечения сетевого трафика для разных задач, запускаемых на вычислительной системе. Аналогичное требование ставится в таких зарубежных сетях как IBM Blue Gene/L [39] и Tofu 1/2/D [22].

Поставленные требования поиска только достижимых множеств узлов и отсутствия пересечения сетевого трафика для разных множеств могут негативно влиять на общую эффективность использования суперкомпьютера с точки зрения утилизации вычислительных ресурсов, в первую очередь из-за явления фрагментации в распределении задач пользователей в топологии сети.

Описанные правила маршрутизации реализованы аппаратно в маршрутизаторе сети Ангара и не могут меняться. Однако данная маршрутизация позволяет задавать множество маршрутов между двумя вычислительными узлами. Для детерминированной маршрутизации необходимо выбрать только один из множества маршрутов. Это возможно делать при помощи таблицы маршрутов, которая формируется программно для каждого узла.

Таблицы маршрутов для множества узлов задачи необходимо строить так, чтобы распределение сетевого трафика по каналам связи внутри этого множества было как можно более равномерным.

При этом важно, чтобы задачи поиска узлов и построения таблиц маршрутов решались за относительно небольшое время, чтобы их можно было использовать при эксплуатации суперкомпьютера: задача поиска узлов решается каждый раз при поиске возможного множества для пользовательского задания в очереди; задача построения таблиц маршрутов, решается при запуске задания на выполнение.

В последующих разделах проводится обзор способов анализа маршрутов в высокоскоростных сетях, алгоритмов построения таблиц маршрутов, в том числе сбалансированных, а также методов уменьшения фрагментации при распределении пользовательских заданий.

1.4 Алгоритмы анализа достижимости узлов в высокоскоростных сетях

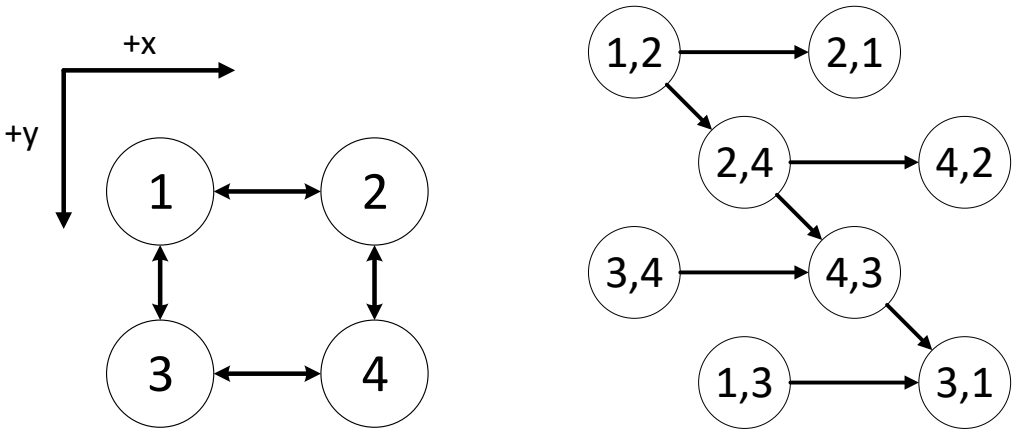
Самым естественным средством анализа топологии сети и наличия маршрутов в ней является *граф сети*, в котором вершинами являются узлы сети, а ребрами – каналы связи. Такой граф топологии сети используется, например, в работах [44–46]. Однако такой граф не позволяет учитывать маршрутизацию правила порядка направлений.

Фундаментальным для всех теорий бездедлоковой маршрутизации в сетях любой топологии является граф зависимостей каналов (англ. Channel Dependency Graph, CDG).

Граф зависимостей каналов для заданной сети и функции маршрутизации R – это ориентированный граф $G(C, E)$, в котором вершинами являются

каналы связи сети, а ребра индуцированы функцией маршрутизации, то есть $(c_p, c_q) \in E$ тогда и только тогда, когда $\exists$ узел сети $n_y : R(c_p, n_y) = c_q$.

Пример графа зависимостей каналов для двухмерной решетки приведен на рисунке 1.1. Одним числом обозначены номера вершин в графе сети, парой чисел обозначены вершины в CDG, соединенные ребром в графе сети в направлении, соответствующему указанному в вершине порядку. Например, вершина 3,4 в CDG графе соответствует ребру графа сети от узла 3 до узла 4.



а) Двухмерная решетка 2x2

б) CDG для решетки 2x2

Рисунок 1.1 — Пример графа зависимостей каналов для двухмерной решетки 2x2 с DOR маршрутизацией. Одним числом обозначены номера вершин в графе сети, парой чисел обозначены вершины в CDG, которые соответствуют ребру в графе сети.

Выбранные правила маршрутизации определяют возможность передачи пакетов, то есть наличие ребер в графе зависимостей каналов. В статье [47] доказана теорема о том, что маршрутизация в сети является бездедлоковой тогда и только тогда, когда в соответствующем графе зависимостей каналов отсутствуют циклы. Впоследствии в работе [48] приведен контрпример, показывающий, что необходимое условие теоремы неверно: в определенных условиях граф зависимостей каналов имеет цикл, но в сети с данной функцией маршрутизации, которая индуцировала цикл, дедлока не будет. Тем не менее, достаточное условие верно и является мощным инструментом.

Избежать циклов в графе зависимостей каналов, соответствующему некоторой сети, можно путем наложения ограничения на маршрутизацию. Так, например, в маршрутизации типа Dimension Order Routing вводится ограничение в виде порядка измерений, в котором допускается движение пакета в

сети. Граф зависимостей каналов используется в большинстве работ по анализу достижимости системы и построения маршрутов. Например, используется в работе [49].

Граф зависимостей каналов применим для анализа наличия маршрута в сети между двумя узлами сети. При этом возможно применение алгоритмов обхода графа. Однако маршрутизация в сети Ангара кроме правила порядка направлений (DOR) ограничена еще требованием dirbit-маршрутизации, поэтому применение графа зависимостей каналов в исходном виде невозможно, так как, например, возможность перехода из $+Z$ в $-X$ в сети Ангара определяется наличием в предыдущих шагах пакета направления $+X$.

Идея хранить в вершинах графа предысторию маршрута в сети удалось найти в статье [50]. В данной работе авторы решают проблему минимизации времени доставки сообщения в гетерогенных сетях с неустойчивыми связями. Примером таких сетей являются сети, построенные с использованием спутников, мобильных сетей и так далее. Алгоритмы маршрутизации, применимые для других сетей, в которых вероятность потери связи невелика, не подходят для таких сетей, поэтому алгоритмы маршрутизации для сетей с неутойчивыми связями являются областью интенсивных исследований. Для таких сетей разрабатывают подходы, которые называются сети устойчивые к разрывам (англ. Delay-tolerant networking, DTN) [51].

Авторы статьи [50] предлагают новый подход в построении маршрутизации с кратчайшим маршрутом в сети (Conditional Shortest Path Routing). Они предлагают новую метрику оценки времени передачи сообщения (conditional intermeeting time), основанную на времени, потраченном на предыдущем шаге, и времени, необходимом для передачи следующему узлу. Для решения задачи авторы используют граф, в котором для каждого из N узлов рассматриваемой сети вводятся дополнительные N вершин, которые показывают, из какого предыдущего узла сети маршрут привел в текущий узел. Таким образом, в графе хранится предыстория маршрута, при этом количество вершин графа составляет N^2.

Приведенная идея с необходимыми расширениями используется в данной диссертационной работе для построения графа с целью анализа маршрутов с учетом всех ограничений маршрутизации сети Ангара.

1.5 Алгоритмы построения таблиц маршрутов в высокоскоростных сетях с топологией «многомерный тор»

Правила маршрутизации чаще всего реализованы аппаратно в высокоскоростных сетях и не могут меняться во время работы системы. При этом правила маршрутизации часто позволяют задавать множество маршрутов между двумя вычислительными узлами. Поэтому в случае детерминированной маршрутизации возникает задача для каждого узла вычислительной системы построить маршрут до всех остальных узлов.

Разработано большое количество алгоритмов построения маршрутов в высокоскоростных сетях, отличающихся целями, условиями, методами и подходами. В задачи алгоритмов построения таблиц маршрутов для детерминированной маршрутизации может входить:

- построение минимальных путей,
- предотвращение дедлоков,
- оптимизация загрузки сети для того или иного паттерна, причем загрузка может определяться по-разному,
- оптимизация скорости построения таблиц маршрутов.

Алгоритмы могут опираться на различное чисто доступных виртуальных каналов, возможность или невозможность перехода между виртуальными каналами, различные топологии [52].

Для решения задачи построения маршрутов популярным методом является применение алгоритмов обработки графов. Например, построение Эйлерова пути по графу топологии сети, алгоритмы поиска вширь, вглубь, Дейкстры поиска кратчайших путей по графу зависимостей каналов. Например, в работе [44] для построения таблиц маршрутизации для сетей с нерегулярной топологией с использованием коммутаторов используется построение Эйлерова пути по графу топологии сети. Аналогичный подход используется в [53].

В алгоритме up/down (UD) [54] для построения маршрутов для произвольных топологий используется поиск вширь кратчайших путей в графе (Breadth-First Search, BFS). После выбора корневого узла алгоритмом поиска вширь строится набор кратчайших путей до остальных узлов. В дальнейшем для построения пути от одного узла к другому позволяется идти по построенному дереву вверх, а затем вниз. В алгоритме up/down все каналы связи помечаются как up или down, и движение разрешается, соответственно, только

вверх или вниз, что приводит к неэффективному использованию каналов связи и проблемам балансировки трафика [55]. Данный алгоритм не гарантирует минимальной длины построенных маршрутов.

Алгоритм Layered Shortest Path (LASH) имеет несколько вариантов [56;57] и использует виртуальные каналы для обеспечения отсутствия дедлоков при построении минимальных маршрутов. Физическая сеть разделена на множество виртуальных слоев сети при помощи отдельных виртуальных каналов. Каждый такой слой не содержит дедлоков и содержит подмножество требуемых минимальных путей, которое строится по графу пути при помощи алгоритма Дейкстры.

Алгоритмы UD, LASH применимы для произвольных топологий. В работе [58] авторы предложили метод маршрутизации с использованием двух виртуальных каналов для топологии 2D-тор. В каждой виртуальной подсети они использовали разный порядок направлений. Авторы статьи [59] расширили этот метод на топологию 3D-тор.

Более детальному обзору подходов к построению алгоритмов маршрутизации, обеспечивающих оптимизацию загрузки, то есть балансировку сетевого трафика, посвящен следующий раздел.

1.5.1 Алгоритмы построения сбалансированных таблиц маршрутов

Алгоритмам построения сбалансированных таблиц маршрутов посвящено большое количество работ, и в каждой из них встает вопрос о критериях качества построенной таблицы маршрутов.

Критерии сбалансированности таблиц маршрутов

Загруженностью G_c канала связи c принято называть количество маршрутов, которым в данной таблице маршрутов принадлежит данный канал связи. Одним из самых популярных критериев сбалансированности таблицы маршрутов RT является *максимальная загруженность канала связи* $\pi_{max}(RT) = \max\limits_{c \in C} G_c$, которую необходимо минимизировать [45; 60] (C – множество всех каналов связи сети).

Другим способом оценки качества таблицы маршрутов может быть оценка наихудшего паттерна обмена сообщениями в сети [61; 62]. Однако данный способ часто слишком затратен по времени, и может не приводить к корректной оценке в общем случае. Например, один из важнейших критериев при оценке сети – бисекционная пропускная способность, реально достижимая пропускная способность зависит как от топологии, так и от маршрутизации, однако измерение достижимой бисекционной пропускной способности может быть не очень просто, требовать симуляции работы сети [63].

Для оценки сбалансированности таблицы маршрутов в литературе принято использовать следующие критерии:

1. Минимизация диаметра построенной таблицы маршрутов;
2. Минимизация максимальной загруженности канала связи π_{max} [64];

Первый критерий возникает из-за того, что в сети с минимальным диаметром задержка на передачу данных будет наименьшей. Второй критерий следует из стремления получить равномерно загруженную систему. Данные критерии приняты к использованию в данной диссертационной работе.

Алгоритмы построения таблиц маршрутов

Оптимальное решение задачи нахождения таблицы маршрутов для минимизации максимальной загруженности канала связи π_{max} – NP-трудная задача для произвольных топологий [65].

Существует несколько подходов для решения задачи оптимизации таблицы маршрутов для произвольных топологий и для сетей с топологией тор.

При использовании подхода построения таблицы маршрутов сразу для всех узлов возникает понятие «глобальной маршрутизации», при таком построении у алгоритма есть информация о загруженности каждого линка сети, и возможно ставить задачу оптимизации качества таблицы маршрутов. Для решения данной задачи возможно использовать подход целочисленного или смешанного целочисленного программирования (англ. Mixed Integer Linear Programming, MILP), данный подход используется, например, в работах [66; 67] для произвольных топологий. Однако подход целочисленного программирования является NP-трудным. При использовании линейного программирования (англ. Linear Programming) возможно получить решение задачи оптимального

распределения потоков трафика за полиномиальное время, однако данный подход не позволяет учитывать широкий набор ограничений маршрутизации.

В статье [68] ученым A. Abdel-Gawad предложен подход, основанный на разделении этапов нахождения наилучших значений загрузки для каждого линка в системе и построения маршрутов для всех пар узлов отправитель-получатель, для которых достигаются эти наилучшие значения. Нахождение наилучших значений загрузки линков достигается при помощи сведения задачи глобальной маршрутизации к задаче линейного программирования, которая может быть решена за полиномиальное время в худшем случае, но допускает разветвление в узлах сети потоков трафика для фиксированной пары отправитель-получатель. Поэтому на втором этапе происходит, собственно, построение маршрутов, но наилучшая загрузка линков достигается при помощи возможности для фиксированной пары узлов отправитель-получатель использовать несколько дополнительных маршрутов. Второй этап реализован как жадный алгоритм, использующий поиск вглубь в графе сети.

Авторы работы [45; 46] предложили алгоритм балансировки маршрутов DFSSSP для произвольной топологии сети на основе алгоритма Дейкстры [69] поиска кратчайших путей графе с весами. Сначала по графу сети строятся маршруты для каждого узла, при этом выбираются сбалансированные варианты при помощи графового алгоритма SSSP (Single Source Shortest Paths). При этом за вес ребра берется сумма числа путей, проходящих через это ребро. Возможны варианты алгоритма: запускать классический алгоритм SSSP от каждой вершины графа, то есть узла сети, до всех остальных сразу, и после этого модифицировать веса в графе. Качество балансировки будет хуже, а время работы – лучше. Второй вариант называется P^2-SSSP, в нем каждой пары узлов отправить-получатель строится маршрут, затем происходит обновление весов графа. Качество балансировки получается выше, но временные затраты также выше. На второй фазе алгоритма при помощи графа зависимостей каналов возможные дедлоки разбиваются с использованием нескольких виртуальных каналов. Все наборы маршрутов размещаются по разным виртуальным подсетям для исключения циклических зависимостей в каждой из подсетей. Идея алгоритма используется и развивается в дальнейших работах [70], авторы статьи [71] модернизировали данный алгоритм для работы с заданным числом виртуальных каналов.

Для топологии тор самым простым является подход к маршрутизации, основанной на относительном положении пары узлов отправитель-получатель [58]. Данный подход прост и может быть реализован в аппаратуре, однако он не предусматривает возможность несимметричных топологий и отказов.

Для суперкомпьютера IBM Blue Gene/Q с топологией 5D-тор предложен метод балансировки oblivious-маршрутизации при помощи использования нескольких путей между каждой парой отправитель-получатель [72]. Для каждой такой пары строится K кратчайших путей при помощи алгоритма Йена [73] по графу сети, затем при отправке данных происходит разбиение данных на порции, отправка каждой порции происходит по одному из построенных заранее маршрутов. Для поиска путей используется алгоритм Йена. При этом используются возможности динамической (адаптивной) маршрутизации Blue Gene/Q, которая позволяет при разных размерах сообщения использовать различные маршруты для каждой фиксированной пары вычислительных узлов.

В некоторых работах применяется метод построения таблицы маршрутов, оптимизированной под конкретную пользовательскую задачу, например, в [66;70], однако такие методы требуют информации о сетевом паттерне в задаче, в том числе об уровне инжекции пакетов в сеть. Также данные методы не учитывают динамику изменения сетевого паттерна.

Для построения сбалансированных маршрутов кроме алгоритма Дейкстры применяются различные эвристические алгоритмы. В статье [55] предложен один из методов балансировки заключается в удалении лишних маршрутов. Для этого на первом этапе строится всеобъемлющий набор маршрутов между каждой парой вычислительных узлов. После этого происходит поиск перегруженного канала и удаляется маршрут наибольшей длины, проходящий через этот канал связи, также данный маршрут удаляется из рассмотрения. Алгоритм продолжает свою работу пока не останется дублирующих маршрутов.

Вполне естественной является идея применения генетического алгоритма для построения сбалансированной таблицы маршрутов [74].

1.6 Алгоритмы выбора узлов в суперкомпьютерах

Для сетей с тороидальной топологией существует несколько стратегий выделения ресурсов [75]. В топологии «многомерный тор» из-за вопросов производительности предпочтительнее выделять ресурсы компактно. Поэтому большинство стратегий предполагает избыточность, то есть для задания пользователя выделяется не менее требуемого числа узлов, что подразумевает возможность выделения узлов, которые не используются задачей.

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 [76; 77], в которых ее недостатки компенсировались большим числом не очень мощных по производительности вычислительных узлов и адекватным выбором размера партиции. Для сети Blue Gene/L разрабатывалась стратегия объединения выделения ресурсов партициями и более мелкозернистый подход к выделению ресурсов [78]. Для сети Blue Gene/Q разрабатывались улучшенные методы планирования ресурсов, основанные на знаниях о требовании приложений к сети [79].

Вторая стратегия используется в серии суперкомпьютеров Cray XT/XE, в которых используется сеть Cray Gemini. В Cray XT/XE расположение выделенных узлов [28] не зависит от топологии. Такой способ выделения ресурсов может привести к деградации производительности ввиду наличия конкурирующего трафика.

В сети Tofu [21] используется упаковка заданий в топологию тор, причем эффективность достигается за счет вариативности упаковки в большом количестве измерений (5D). Сетевой трафик заданий пользователей не должен пересекаться [22].

Алгоритмы компактного выделения узлов

Существует большое количество алгоритмов для компактного выделения узлов в топологии «многомерный тор». При этом к созданию алгоритмов можно выделить несколько подходов: перебора многомерных прямоугольников, при помощи построения кривой, проходящей через необходимое число узлов, построения многомерных прямоугольников путем расширения из разных узлов.

Перебор многомерных прямоугольников. Данный подход описан для суперкомпьютера IBM Blue Gene/L [39]. Для пользовательского задания размера m производится поиск всевозможных многомерных прямоугольников с числом узлов m, которые можно вписать в топологию. Каждый такой прямоугольник не должен пересекаться с ранее запущенными заданиями. Из множества полученных прямоугольников выбирается такой, что после выделения узлов под задание пользователя в системе останется наибольшее свободное место, вписываемое в многомерный прямоугольник. В системе может не существовать прямоугольник размера m из-за того, что m невозможно разложить на множители размера меньше, чем размер измерения тора, тогда размер искомых прямоугольников увеличивается до ближайшего подходящего.

Построение кривой. Основная идея подхода (в работе [80] алгоритм называется best-fit) заключается в линейной перенумерации узлов по порядку прохождения некоторой кривой, соединяющей все узлы. Для топологии «многомерный тор» в качестве такой кривой может использоваться кривая Гильберта, в данном случае соседние в линейной нумерации узлы расположены максимально близко в топологии. Затем доступные узлы группируются в непрерывные участки в созданной линейной нумерации и выбирается группа с наименьшим числом узлов, но при этом удовлетворяющая требованиям задания пользователя. Если такой группы не существует, то происходит поиск узлов с наименьшим максимальным расстоянием в линейной нумерации. Данный алгоритм хорош тем, что имеет небольшую вычислительную сложность выбора узлов, однако не учитывает особенности маршрутизации в коммуникационной сети и допускает пересечение сетевого трафика различных заданий пользователя.

Расширение многомерных прямоугольников. В работе [80] описывается алгоритм Mc1x1 поиска компактного множества узлов. Для каждого свободного узла $n \in N$ строится гиперкуб с центром в узле n. Начальный размер гиперкуба равен единице, то есть содержит только узел n. В дальнейшем

гиперкуб расширяется до тех пор, пока не покроет необходимое число свободных вычислительных узлов. Каждый гиперкуб J оценивается относительно узла n с помощью функции $NS_{n,J} = \sum_{i \in J} dist_{n,i}$, где $dist_{n,i}$ – расстояние между узлом n и свободным узлом i в гиперкубе J. Из полученных гиперкубов выбирается гиперкуб с наименьшим значением функции $NS_{n,J}$. Алгоритм применим только к топологии «многомерный тор». Данный алгоритм не учитывает фрагментацию вычислительной системы. Также данный метод не учитывает особенности маршрутизации и допускает пересечение сетевого трафика различных заданий пользователя.

Примером одновременного решения задачи выделения узлов и отображения узлов задания пользователя на топологию сети может стать алгоритм PaCMap (англ. Partitioning and Center Mapping), предложенный в работе [80]. На этапе отображения на топологию сети граф коммуникаций пользовательского задания с помощью библиотеки METIS [81] разбивается на J наиболее сильно связанных групп процессов, каждая из которых будет соответствовать одному вычислительному узлу. Таким образом, задача сводится к выделению необходимого количества узлов из доступных. При этом из полученных групп выделяют центральную, характеризующуюся наименьшей суммой длин кратчайших путей до остальных групп, которая будет отображена на вычислительный узел, обозначенный как n. Остальные группы отображаются на другие узлы итеративно таким образом, чтобы минимизировать использование сети.

Этап выделения узлов производится следующим образом. Для требуемого числа узлов J и каждого возможного узла n вычислительной системы выбираются узлы в окрестности R. Из этой окрестности выбираются первые $J - 1$ узлов, отсортированных в порядке удаления от узла n. Для каждого такого множества узлов оценивается характеристика $NS_{n,J} = \sum\limits_{i=1}^{J} f(dist_{n,i}) - \sum\limits_{i=J+1} f(dist_{n,i})$. Для топологии 3D-тор авторы используют $f(dist_{n,i}) = \frac{1}{4dist_{n,i}^2+2}$, где $dist_{n,i}$ – расстояние между узлами n и i. В итоге выбирается тот узел n, для которого $NS_{n,J}$ будет наибольшим.

На эффективность использования вычислительных ресурсов влияет понятие фрагментации системы. Авторы статьи [82] вводят понятие подрешетки (submesh) – это такой многомерный прямоугольник, вписанный в топологию тор, который можно описать двумя координатами противоположных углов. Для описания ресурсов доступных для выделения авторы вводят расширение поня-

тия подрешетки. Максимально свободная подрешетка (MFS) – это подрешетка, которая состоит из не занятых узлов и не может быть расширена в любом из направлений. Таких подрешеток может быть несколько.

При этом алгоритм поиска узлов для задания происходит следующим образом. После попадания задания в очередь система выбора заданий ожидает доступного свободного множества узлов и выбирает наиболее подходящий MFS из доступных. Первыми рассматриваются MFS размера, равного размеру задания. Если таких нет, то будет выбран первый подходящий или случайный. Подобный подход используется и в других работах, например, в работе [83].

Планирование ресурсов

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 [84] (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 [85] 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.

В работе [86] рассматривается алгоритм планирования, в котором более поздней задаче разрешается обойти задачи, находящиеся дольше в очереди и ожидающие выделения ресурсов. Эта возможность ограничена окном последовательных заданий фиксированного размера, начиная с самого первого задания, ожидающего выделения ресурсов. В работе показывается, что предложенный алгоритм показывает улучшение времени ожидания задания и увеличивает утилизацию вычислительной системы при больших нагрузках.

Одним из самых эффективных на данный момент алгоритмов многие авторы называют алгоритм Backfill [87] – алгоритм обратного заполнения, который является расширением политики FCFS. Для работы этого алгоритма необхо-

димо 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 реализованы в программных системах управления ресурсами: PBS [88], Torque [89], MAUI [90], SGE [91], MBC-1000 [92], Slurm [93].

В системе управления ресурсами Slurm выбор следующего задания для запуска реализован двумя способами FCFS – первыми будут обслужены задания с большим приоритетом и Backfill – допускающий запуск задания с меньшим приоритетом, если это не повлияет на задачу с большим приоритетом. Для решения задачи справедливого распределения ресурсов Slurm может выставлять приоритет задачам согласно очереди, в таком случае наивысший приоритет будет у той задачи, которая стоит первой в очереди, вторая задача будет иметь меньший приоритет, третья – еще меньший и так далее; такая политика реализована в плагине, который называется priority/basic и используется по умолчанию. Однако такая политика может плохо работать в системе с большим числом пользователей, позволяя определенным пользователям захватить все ресурсы вычислительного кластера. Для решения этой проблемы в Slurm реализован плагин priority/multifactor, который по различным параметрам вычисляет приоритет задания, среди параметров: время ожидания задания в очереди; требуемое число узлов; приоритет задания, выставленного пользователем; приоритет партиции; качество обслуживания (QoS); требуемые ресурсы на узлах; используемые ранее ресурсы конкретным пользователем.

1.7 Выводы

В данной главе рассмотрены основные принципы маршрутизации в сетях с топологией «многомерный тор».

Рассмотрена проблема необходимости в условиях наличия отказов и занятых другими заданиями узлов предоставлять возможность выделения для задания пользователя связного множества узлов, отвечающего запрашиваемым вычислительным мощностям и требованиям маршрутизации. При этом поставлено требование отсутствия пересечения сетевого трафика для разных заданий, запускаемых на вычислительной системе.

Вторая задача, возникающая при решении сформулированной проблемы – необходимость построения таблицы маршрутов, то есть для выбора маршрута из множества возможных для любого фиксированного узла ко всем остальным в предоставленном множестве. При этом распределение сетевого трафика по каналам связи внутри этого множества должно быть как можно более равномерным.

В последующих разделах проводится обзор способов анализа маршрутов в высокоскоростных сетях, алгоритмов построения таблиц маршрутов, в том числе сбалансированных, а также методов уменьшения фрагментации при распределении пользовательских заданий.

Выяснено, что основной инструмент анализа высокоскоростных сетей – граф зависимостей каналов не может быть применен для маршрутизации с правилом порядка направлений и ограничениями dirbit-маршрутизации. Для анализа маршрутов предполагается использовать идею хранить в вершинах графа информацию о предыстории маршрута.

В таблице 1 представлено сравнение сети Ангара с сетями с топологией «многомерный тор». Из таблицы видно, что близкими для сети Ангара по правилам маршрутизации являются сети IBM Blue Gene L/P/Q. Ограничение на запрет пересечения трафика различных заданий введен в трех из четырех из сравниваемых сетей. Таким образом, решение поставленных задач диссертационной работы может быть применено и к другим сетям с топологией «многомерный тор», схожими правилами маршрутизации и ограничениями, в первую очередь к сетям IBM Blue Gene L/P/Q.

Таблица 1 — Сравнение сетей с топологией многомерный тор и применяемого программного обеспечения.

		Cray Gemini	IBM Blue Gene L/P/Q	Tofu 1/2/D	Ангара
	Топология	3D-тор	3D/3D/5D-тор	6D-тор	4D-тор
Аппаратные возможности	Маршрутизация согласно правилу порядка направлений DOR	−	+	+	+
	Запрет движения в разных направлениях по измерению (dirbit)	−	+	−	+
	Дополнительные шаги в маршруте пакета	−	−	+	+
Программное обеспечение	Защита от дедлоков в сети возложена на программное обеспечение	−	−	н/д	+
	Трафик различных заданий не пересекается	−	+	+	+
	Компактное выделение узлов	−	+	+	+
	Мелкозернистая поддержка отказов (на уровне отдельных узлов и линков)	+	−	+	+

Глава 2. Маршрутный граф для анализа маршрутов в заданной сети

Глава посвящена описанию предложенного в работе алгоритма построения маршрутного графа для анализа маршрутов в сети с топологией «многомерный тор», на основе которого решается задача определения достижимости исследуемого множества узлов сети. В начале главы вводятся определения, необходимые для анализа маршрутов, и формулируется постановка задачи.

2.1 Общие определения

В диссертационной работе рассматривается *коммуникационная сеть* $I(N,C)$ с топологией «многомерный тор», где N – множество узлов сети, C – множество каналов связи. Размерности тора обозначим $(d_1, d_2, ..., d_n)$, где n – число измерений тора. Каждый узел сети u имеет координаты $(u_1, u_2, ..., u_n)$, где $0 \leqslant u_i < d_i$.

Направлением D_j будем называть набор $D_j = (0,..., \underbrace{\pm 1}_{j \bmod n} ,...,0)$ длины n, где на позиции $j \bmod n$ будет стоять $+1$, если $1 \leqslant j \leqslant n$; -1, если $n+1 \leqslant j \leqslant 2n$. На остальных позициях находятся ноли.

Направления с номерами $1, ..., n$ будем называть *положительными*, а с номерами $n+1, ..., 2n$ – *отрицательными*. Соседними в рамках тороидальной топологии в направлении D_j будем называть узлы $u = (u_1, u_2, ..., u_n)$ и $v = u + D_j = (u_1, ...,(u_{j \bmod n} \pm 1) \bmod d_{j \bmod n}, ..., u_n)$ для любого индекса $1 \leqslant j \leqslant 2n$. В дальнейшем будем употреблять выражение: «узел v находится в направлении D_j от узла u».

Множество направлений обозначим $\mathcal{D}$:

$$\mathcal{D} = \{D_j\}_{j=\overline{1,2n}}.$$

На множестве направлений $\mathcal{D}$ введем порядок в соответствие с указанной нумерацией: $D_i < D_j$, если $i < j$.

Определение 2.1. *Каналом связи (линком) будем называть пару (u, D), где $u \in N$, $D \in \mathcal{D}$. Множество всех каналов связи обозначим $C = N \times \mathcal{D} \setminus F$, где F – множество отказавших каналов связи.*

Определение 2.2. *Маршрут P_{u^0,u^l} в сети $I(N,C)$, соединяющий два узла сети u^0 и u^l, – это последовательность вида $u^0, D_1, u^1, D_2, ...,D_l,u^l$, где l –*

длина маршрута, $u^i \in N$, $\forall i = \overline{0,l}$, *шаг (переход) между узлами* u^{j-1} *и* u^j *производится по направлению* D_j, $\forall j = \overline{1,l}$, *а* $(u^j, D_{j+1}) \in C$. *При этом* $T(P_{u^0,u^l}) = u^1, ..., u^{l-1}$ *– транзитные узлы маршрута.*

Так как транзитные узлы маршрута могут быть получены из соответствующих шагов, то их можно опустить, тогда подобный маршрут будет записываться в виде: $u^0, D_1, D_2, ..., D_l$.

В реальной вычислительной системе некоторые каналы связи могут быть неисправны или заняты. Так как физический канал связи между двумя узлами v и u представляет собой канал связи от узла v к узлу u и наоборот, то разумно предположить, что при неисправности одного из каналов связи второй так же неисправен. Таким образом, множество F будет включать в себя отказавшие каналы связи попарно. Отказавший узел можно интерпретировать как узел, у которого каналы связи сломаны во всех направлениях.

Далее будут введены определений, описывающие правила маршрутизации в сети Ангара.

Правило порядка направлений с использованием битов направлений

В сети Ангара применяется правило порядка направлений (англ. Direction Order Routing, DOR), которое состоит в следующем.

Определение 2.3. *Маршрут* $P_{u,v} = u, D_1, D_2, ..., D_l$ *из узла* u *в узел* v *сети* I *удовлетворяет правилу порядка направлений, если* $D_{i-1} \leqslant D_i, i = \overline{2,l}$, *где* l *– длина маршрута;* $D_i \in \mathcal{D}$, $\forall i = \overline{1,l}$; $u \in N$ *– стартовый узел маршрута;* $v \in N$ *– конечный узел маршрута.*

Заметим, что маршрут $P_{u,v} = u^0, D_1, D_2, ..., D_l$ из узла u в узел v, удовлетворяющий правилу порядка направлений, можно записать при помощи группировки шагов по каждому из направлений сети: $u^0, S_1, ..., S_{2n}$, где S_j – набор шагов (возможно пустой) в направлении $D_j \in \mathcal{D}$. Длина маршрута может быть получена следующим образом: $l = \sum_{i=1}^{2n} |S_i|$, где $|S_i|$ – число шагов в наборе S_i.

В качестве дополнительного ограничения к правилу порядка направлений в сети Ангара реализована маршрутизация с использованием битов направлений (dirbit-маршрутизация), которая запрещает движение пакета в разных

направлениях одного и того же измерения, например, в $+X$ и $-X$ пакету двигаться запрещено.

Определение 2.4. *Маршрут* $P_{u,v}^{dirbit} = u^0,S_1,S_2,...,S_{2n}$ *из узла* u *в узел* v *сети* I *удовлетворяет правилу битов направлений, если маршрут* $P_{u,v}^{dirbit}$ *удовлетворяет правилу порядка направлений и* $\forall i = \overline{1,n}$ *наборы шагов* S_i *удовлетворяют следующему условию: либо* $|S_i| > 0$, *либо* $|S_{i+n}| > 0$, *либо* $|S_{i+n}| = |S_i| = 0$.

В маршруте $P_{u,v}^{dirbit} == u^0,S_1,S_2,...,S_{2n} = u,D_1,..,D_1,D_2,...,D_2,...,D_l,...,D_l$ набор направлений $D_1,D_2,...,D_l$ обозначим как D_{dirbit}. Множество таких маршрутов позволяет определить маршрутизацию R_{dirbit}.

Определение 2.5. *Маршрутизация на основе правила битов направлений в сети* $I(N,C)$ *определяется функцией* $R_{dirbit} : N \times N \to \mathcal{P}(P_{u,v}^{dirbit})$, *где* $\mathcal{P}(P)$ *– множество всех подмножеств* P, *маршрут* $P_{u,v}^{dirbit}$ *удовлетворяет правилу битов направлений.*

First Step/Last Step

Маршрутизация R_{dirbit} при помощи заданного порядка обработки направлений тора обеспечивает отсутствие дедлоков при движении сетевых пакетов между измерениями тора.

Для расширения возможностей маршрутизации в сети Ангара реализован алгоритм First Step/Last Step (FS/LS) [14] «нестандартного первого и последнего шага», который позволяет сделать первый и последний шаг без ограничений dirbit-маршрутизации и без ограничений правила порядка направлений. Например, благодаря FS/LS допустимы следующие порядки шагов: $+X(FS),+Y,+Z,-X,-Z(LS)$; $+Y(FS),+X,-Y,-X(LS)$.

Определение 2.6. *Маршрут* $P_{u,v}^{A}$ *в сети* I *с использованием первого и последнего нестандартного шага FS/LS – это маршрут, который может быть представлен как* $u, D_{FS}, D_{dirbit}, D_{LS}$, *где* u *– стартовый узел сети* I, D_{FS} *– положительное направление,* D_{LS} *– отрицательное направление, а* D_{dirbit} *– набор направлений, удовлетворяющих правилу битов направлений. При этом* D_{FS} *и* D_{LS} *могут отсутствовать.*

Определение 2.7. *Маршрутизация на основе правила порядка направлений с использованием битов направлений, а также FS/LS в сети* $I(N, C)$ *определяется функцией* $R_A : N \times N \to \mathcal{P}(P^A_{u,v})$, *где* $\mathcal{P}(P)$ – *множество всех подмножеств* P. *Данная функция для узла отправителя* $u \in N$ *и узла получателя* $v \in N$ *возвращает набор всевозможных маршрутов* $P^A_{u,v}$ *с использованием FS/LS из узла* u *в узел* v.

В сети Ангара аппаратно реализована маршрутизация R_A на основе правила порядка направлений с использованием битов направлений, а также FS/LS. Заметим, что такая маршрутизация может приводить к дедлокам в сети, так как на FS/LS не накладывается требование порядка направлений.

В диссертационной работе поставлено требование отсутствия пересечения сетевого трафика для разных задач, запускаемых на вычислительной системе. Это ограничение порождает задачу определения достижимости множества вычислительных узлов в заданной сети.

Определение 2.8. *Множество узлов* $M_{a,t}$ *сети* $I(N, C)$ *достижимо, если* $\forall u, v \in M_a, u \neq v \; \exists P_{u,v} : T(P_{u,v}) \subset M_{a,t}$, *где* $M_{a,t} = M_a \cup M_t$, *где* $M_a \subseteq N$ – *множество активных узлов сети, то есть выполняющих инжекцию/эжекцию (отправку/приём) пакетов в сеть,* $M_t \subset N$ – *множество транзитных узлов сети, которые не выполняют инжекцию/эжекцию пакетов, такие узлы могут быть необходимы для поддержания достижимости системы.*

Заметим, что вычислительный узел может быть недоступным, например в результате отказа. В таком случае он не входит ни в множество M_a, ни в множество M_t.

Маршрутный граф

Стандартные подходы к анализу маршрутов в сети Ангара не подходят из-за применения маршрутизации на основе правила битов направлений, так как решение о следующем допустимом шаге в маршруте зависит от предыдущих шагов. В данном разделе вводится понятие маршрутного графа, при помощи которого будет решаться задача разработки алгоритма анализа маршрутов в сети Ангара.

Определение 2.9. *Пусть даны сеть $I(N,C)$, а также функция маршрутизации $R : N \times N \to \mathcal{P}(P_{u,v})$, где $\mathcal{P}(P)$ – множество всех подмножеств P, $P^R_{u,v}$ – маршрут, соединяющий два узла сети u и v согласно функции R. Маршрутным графом сети I называется граф $RG(V,E)$, в котором маршрут $P^R_{u,v}$ в сети I между узлами u, v существует тогда и только тогда, когда в маршрутном графе существует путь между вершинами, соответствующими узлам u и v.*

Рассмотрим маршрутизацию в сети Ангара с ограничением правила порядка направлений на шаги FS/LS.

Определение 2.10. *Маршрут $P^{A^-}_{u,v}$ – это маршрут в сети I, который может быть представлен как $u, D_{FS}, D_{dirbit}, D_{LS}$, где u – стартовый узел сети I, D_{FS} – первое положительное нестандартное направление, D_{LS} – последнее отрицательное нестандартное направление, а D_{dirbit} – набор направлений, удовлетворяющих правилу битов направлений, причем набор D_{FS},D_{dirbit},D_{LS} также удовлетворяет правилу порядка направлений. При этом D_{FS} и D_{LS} могут отсутствовать.*

Определение 2.11. *Маршрутизация на основе правила порядка направлений с использованием битов направлений, а также FS/LS, удовлетворяющих правилу порядка направлений в сети $I(N,C)$, определяется функцией $R_{A^-} : N \times N \to \mathcal{P}(P^{A^-}_{u,v})$, где $\mathcal{P}(P)$ – множество всех подмножеств P. Данная функция для узла отправителя $u \in N$ и узла получателя $v \in N$ возвращает набор всевозможных маршрутов $P^{A^-}_{u,v}$ из узла u в узел v.*

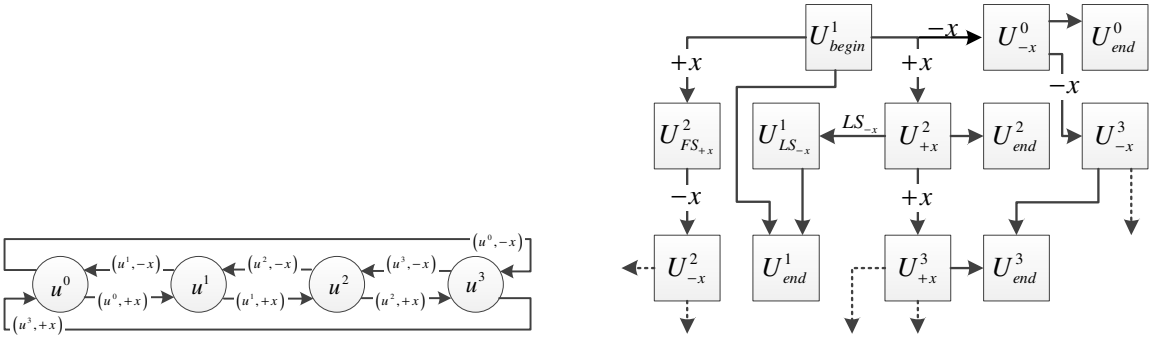
Задача разработки алгоритма анализа маршрутов в сети Ангара формулируется следующим образом. Так как маршрутизация R_A допускает дедлоки, то необходимо построить бездедлоковую функцию маршрутизации $R_{A^*} \supseteq R_{A^-}$. Для сети I требуется построить маршрутный граф для функции маршрутизации R_{A^*}.

2.2 Маршрутный граф без нарушения правила порядка направлений

В данном разделе будет описан алгоритм построения маршрутного граф для маршрутизации R_{A^-} в сети $I(N,C)$. Рассмотрим ориентированный граф $RG(V,E)$.

Вершины графа будем обозначать U^i_X. Каждому вычислительному узлу сети I соответствует несколько вершин графа. Верхний индекс i определяет, какому узлу сети соответствует данная вершина. Нижний индекс X, определяет информацию о предыстории маршрута, которая в соответствии с правилами маршрутизации вносит ограничения на принятие решения о следующем шаге.

Например, рассмотрим граф $RG(V,E)$ (рисунок 2.1б), построенный для одномерной топологии с длиной кольца 4 (рисунок 2.1а). На рисунке для наглядности опущена большая часть ребер.



а) Кольцо длиной 4

б) Часть графа для кольца длиной 4

Рисунок 2.1 — Пример построения части графа для кольца длиной 4. Порядок направлений $+X-X$.

Из вершины U^1_{begin}, соответствующей инжекции в сеть из узла u^1, возможны шаги: в направлении $+X$ в вершины U^2_{+x} и $U^2_{FS_{+x}}$, в направлении $-X$ в вершину U^0_{-x}, на эжекцию в вершину U^1_{end}. Вершина U^2_{+x}, соответствует узлу сети u^2. Нижний индекс $+x$ означает, что в эту вершину совершено движение по направлению $+X$, и дальнейшее движение с учетом всех правил маршрутизации возможно только: в вершину U^3_{+x} в направлении $+X$; в вершину $U^1_{LS_{-x}}$ в направлении $-X$, совершив последний нестандартный шаг; на эжекцию в вершину U^2_{end}. Вершина U^1_{+x-x} отсутствует, так как такой маршрут $+X-X$ не соответствует правилам маршрутизации. Остальные ребра и вершины строятся аналогично.

Рассмотрим движение по некоторому маршруту в торе в направлениях: $\{S_1\}, ..., \{S_i\}$. Этот маршрут можно продолжить в том же направлении D_i или в таком направлении D_{i+1}, что набор направлений $\{S_1\}, ..., \{S_i\}, D_{i+1}$ удовлетворяет правилам маршрутизации, где $i \leqslant 2n-1$. Возможность выбора D_{i+1} зависит только от набора $\{S_1\}, ..., \{S_i\}$.

Поэтому для описания каждого узла сети $u^i \in N$ в графе $RG(V, E)$ построим множество U^i вершин, которые будут характеризовать предысторию маршрутов, которые проходят через узел u^i. Множество U^i состоит из следующих вершин:

1. U^i_{begin} – вершина, из которой начинается движение (инжекция пакета в сеть);
2. $U^i_{FS_j}, j = 1, ..,n$ – вершины, в которые можно попасть, совершив первый нестандартный положительный шаг из соседнего узла в узел i в направлении D_j;
3. $U^i_{Dirbit_j}, j = \overline{1,3^n-1}$, где индекс $Dirbit_j$ – соответствует некоторому набору направлений, удовлетворяющему правилу битов направлений, двигаясь по которым можно попасть в данную вершину i;
4. $U^i_{LS_j}, j = n+1, ..,2n$ – вершины, в которые можно попасть, совершив последний нестандартный отрицательный шаг из соседнего узла в направлении D_j;
5. U^i_{end} – вершина, в которой заканчивается движение (эжекция пакета из сети).

Посчитать количество вершин U^i_{dirbit} можно следующим образом. Закодируем набор n-мерным вектором, где на j-ом месте стоит -1 в случае движения в отрицательном направлении, $+1$ – в случае движения в положительном направлении и 0 – в случае отсутствия движения в j-том измерении тора. Всего таких наборов $3^n - 1$, так как нулевому набору соответствует вершина U_{begin}.

Таким образом, $U^i = \{(\cup_{j=1}^{n} U^i_{FS_j}) \cup (\cup_{j=n+1}^{2n} U^i_{LS_j}) \cup (\cup_{j=1}^{3^n-1} U^i_{dirbit_j}) \cup U^i_{begin} \cup \cup U^i_{end}\}$,

$$C_V = |U^i| = n + n + 3^n - 1 + 2 = 3^n + 2n + 1. \tag{2.1}$$

Так как $V = \cup_{i=1}^{|N|} U^i$, то общее число вершин в графе

$$|V| = |N| * |U^i| = |N| * (3^n + 2n + 1) = C_V * |N|, \forall u^i \in N.$$

Вершины в графе соединяются таким образом, чтобы переход от одной вершины графа к другой соответствовал маршрутам, проходящим через соответствующие узлы сети и удовлетворяющим маршрутизации R_{A^-}. Опишем всевозможные ребра в графе $RG(V, E)$:

1. Рассмотрим вершину U^i_{begin}, соответствующую узлу u^i, из которой начинается движение. Первым шагом в маршруте из узла u^i может быть:
   - движение в направлении первого нестандартного положительного шага D_k в вершину $U^j_{FS_k}$, соответствующую узлу $u^j = u^i + D_k$,
   - движение в любом направлении D_k в вершину $U^j_{dirbit_l}$, где $dirbit_l$ в данном случае соответствует D_k. Вершина $U^j_{dirbit_l}$ соответствует узлу $u^j = u^i + D_k$,
   - движение в вершину U^i_{end} для завершения движения.
2. Рассмотрим вершины $U^i_{FS_l}$, соответствующие узлу u^i. Из этих вершин возможно движение в вершины $U^j_{dirbit_t}$, соответствующие узлам $u^j = u^i + D_k$ в направлениях D_k таких, что $D_l < D_k$. Аналогично предыдущему случаю – $dirbit_t$ в данном случае соответствует D_k.
3. Рассмотрим вершины $U^i_{dirbit_l}$, соответствующие узлу u^i, через который проходят маршруты с набором направлений $dirbit_l$. Из этих вершин возможно движение:
   - в некоторую вершину $U^j_{dirbit_t}$ в таком направлении D_k, что последнее направление в $dirbit_l \leqslant D_k$ и $dirbit_t = \{dirbit_l, D_k\}$. При этом $u^j = u^i + D_k$.
   - в некоторую вершину $U^j_{LS_k}$ в таком направление D_k, что последнее направление в $dirbit_l < D_k$. При этом $u^j = u^i + D_k$.
   - в вершину U^i_{end} для завершения движения.
4. Рассмотрим вершины $U^i_{LS_k}$, соответствующие узлу u^i, через который проходят маршруты с последними нестандартными отрицательными направлениями D_k. Из этих вершин возможно движение только в вершину U^i_{end} для завершения движения.

Посчитаем, сколько ребер графа приходится на один набор U^i вершин. В первом случае вершины U^i_{begin} имеют $n + 2n + 1$ ребер. Во втором случае для каждого первого нестандартного шага D_j существует $2n - j$ вариантов, итого получим $\sum\limits_{j=1}^{n}(2n - j) = 2n * n - \frac{(1+n)n}{2} = \frac{3n^2-n}{2}$ ребер. В третьем случае для

каждой вершины $U^i_{dirbit_l}$ из $3^n - 1$ вершин будет не больше, чем $2n$ соседей. В четвертом случае – n ребер.

Просуммировав все, получим C_E – оценку числа ребер на каждый узел сети u^i:

$$C_E = |E^i| = n + 2n + 1 + \frac{3n^2 - n}{2} + (3^n - 1)2n + n = 2n3^n + 1.5n^2 + 1.5n + 1. \tag{2.2}$$

Таким образом, общее количество ребер в графе

$$|E| = O((2n3^n + 1.5n^2 + 1.5n + 1)|N|) = O(C_E * |N|).$$

Теорема 2.1. *Из узла a в узел b сети I(N,C) существует маршрут $P_{a,b}$, удовлетворяющий маршрутизации R_{A^-}, тогда и только тогда, когда в графе $RG(V, E)$ существует путь из вершины U^a_{begin} в вершину U^b_{end}.*

Доказательство. Для доказательства приведем взаимно однозначное соответствие между множеством маршрутов в сети и множеством путей в графе RG. Рассмотрим маршрут:

$$P_{a,b} = a, D_{FS}, D_{dirbit}, D_{LS} = a, D_{FS}, D_1,...,D_{l-1}, D_{LS} =$$
$$= a, D_{FS}, \{S_1\}, ...,\{S_{2n}\}, D_{LS}$$

Построим соответствующий путь в графе RG. Для простоты номера узлов, соответствующих данным переходам, будем обозначать некоторой функцией $\delta(j)$, где j – номер шага. Первый переход в первом нестандартном положительном направлении D_{FS} соответствует переходу из вершины U^a_{begin} в вершину $U^{\delta(1)}_{D_{FS}}$.

Далее следует серия переходов в направлениях $\{S_k\}$, где k – минимальный индекс, при котором $|S_k| \neq 0$. При первом переходе из вершины $U^{\delta(1)}_{D_{FS}}$ в вершину $U^{\delta(2)}_{D_k}$ в направлении D_k помимо смены верхнего индекса (номера узла) произойдет смена нижнего индекса с D_{FS} на D_k, тем самым зафиксировав предысторию маршрута. Последующие шаги в направление D_k будут менять только верхний индекс. В результате $|S_k|$ переходов в направлении D_k будет произведен переход в вершину $U^{\delta(1+|S_k|)}_{D_k}$.

Аналогично строятся последующие переходы по множествам $\{S_i\}$, где $\forall i, k < i \leqslant 2n, |S_i| \neq 0$. Последним шагом совершается движение из вершины $U_{D_{dirbit}}^{\delta(1+\sum_{j=1}^{2n}|S_j|)}$ в направлении D_{LS} и затем эжекция. Полный путь в графе:

$$U_{begin}^{a}, U_{FS}^{\delta(1)}, \underbrace{U_{D_1}^{\delta(1+1)}, ..., U_{D_1}^{\delta(1+|S_1|)}}_{|S_1|}, ..., \underbrace{U_{D_{dirbit}}^{\delta(1+\sum_{j=1}^{2n-1}|S_j|+1)}, ..., U_{D_{dirbit}}^{\delta(1+\sum_{j=1}^{2n}|S_j|)}}_{|S_{2n}|},$$
$$U_{LS}^{\delta(1+\sum_{j=1}^{2n}|S_j|+1)}, U_{end}^{\delta(1+\sum_{j=1}^{2n}|S_j|+1)}.$$

Таким образом, маршруту $P_{u,v}$ в сети соответствует единственный путь в графе $RG(V,E)$. Аналогичным образом по данному пути в графе RG можно построить единственный маршрут в сети, удовлетворяющий правилам маршрутизации. Это соответствие показывает, что существование маршрута в сети из узла a в b равносильно существованию пути в графе RG из U_{begin}^{a} в U_{end}^{b}.■

По доказанной теореме согласно определению 2.9 построенный граф $RG(V,E)$ является маршрутным.

Таким образом, данный маршрутный граф описывает маршрутизацию R_{A^-} для исследуемой коммуникационной сети Ангара.

2.3 Маршрутный граф с нарушением правила порядка направлений

В маршрутизаторе сети Ангара отсутствуют аппаратные ограничения по выбору первого и последнего нестандартного шага, однако если их выбор производится с учетом правила порядка направлений, то гарантируется отсутствие дедлоков. Возможность построения маршрута с нарушением правила порядка направлений увеличивает общее число маршрутов между каждой парой вычислительных узлов, а следовательно, повышает отказоустойчивость вычислительной системы в целом. Данный раздел посвящен анализу возможности нарушения правила порядка направлений для первого и последнего нестандартного шага.

Для анализа возможности нарушения правила порядка направлений рассмотрим граф зависимости каналов (Channel Dependency Graph, CDG) для коммуникационной сети с топологией «многомерный тор» и маршрутизацией на основе правила порядка направлений и правила «пузырька».

Определение 2.12. *Маршрутизация с правилом порядка направлений в сети $I(N,C)$ определяется функцией маршрутизации $R_{DOR} : C \times N \to \mathcal{P}(C)$, $\mathcal{P}(C)$ – множество всех подмножеств C. Функция R_{DOR} для канала связи $c_{q-1} \in C$ на текущем шаге маршрута $P_{*,n}$ в узел-получатель n возвращает набор возможных каналов связи $\{c_q^1, ..., c_q^p\}$ следующего шага маршрута $P_{*,n}$, причем должно выполняться условие $D(c_{q-1}) \leqslant D(c_q^i), \forall i = \overline{1,p}$, где $D(c)$ – направление тора для канала связи c.*

Определение 2.13. *Графом зависимостей каналов для сети $I(N,C)$ и функции маршрутизации R назовем ориентированный граф $G(C,E)$, вершинами которого являются каналы связи C данной сети. Ребра графа G определяются функцией маршрутизации, то есть $(c_i, c_j) \in E$ тогда и только тогда, когда $\exists n_y \in N : c_j \in R(c_i, n_y)$.*

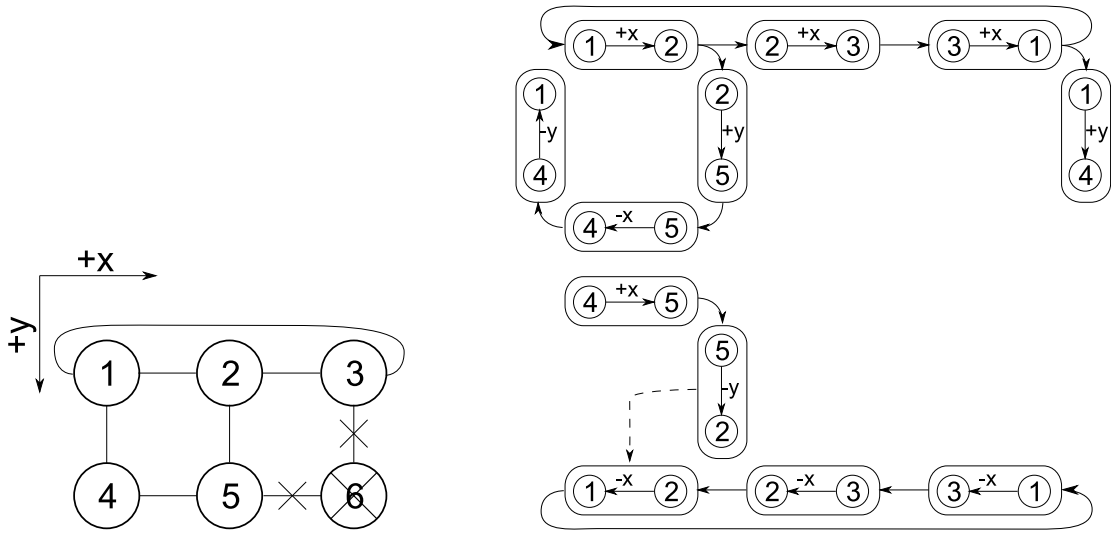
Пояснить существование ребра графа G можно, если представить ребро $(c_i, c_j) = ((u_i, D_i), (u_j, D_j))$ как допустимый функцией маршрутизации переход из узла u_i в узел u_j по направлению D_i: $u_j = u_i + D_i$.

В дальнейшем будем рассматривать граф зависимости каналов для маршрутизации R_{DOR} с правилом порядка направлений. В работах [47; 48] доказана следующая теорема:

Теорема 2.2. *Маршрутизация $R : C \times N \to \mathcal{P}(C)$ является бездедлоковой, если в соответствующем графе зависимостей каналов отсутствуют циклы.*

В качестве примера рассмотрим двухмерную топологию 3x2 со сломанным узлом 6, изображенной на рисунке 2.2а. Соответствующий граф зависимостей каналов, индуцированный маршрутизацией R_{DOR} представлен на рисунке 2.2б. На рисунке, для удобства, для каждой вершины графа $(u_i,D_i) \in C$ дополнительно указан соответствующий узел сети $u_j = u_i + D_i$. Сплошными стрелками показаны ребра графа, индуцированные маршрутизацией R_{DOR} с заданным порядком направлений: $+X + Y - X - Y$. Пунктирной линией нарисовано ребро графа, не удовлетворяющее функции маршрутизации R_{DOR}, так как нарушает правило порядка направлений.

На рисунке 2.2б можно заметить цикл в кольце по измерению X: $(1, +X) \longrightarrow (2, +X) \longrightarrow (3, +X) \longrightarrow (1, +X)$. Однако цикл в одном кольце в топологии тор может разрешаться при помощи правила «пузырька». В работе [35] для этого доказана следующая теорема:



а) Двухмерная топология 3x2 с 5 узлами

б) Соответствующий граф зависимости каналов индуцированный маршрутизацией R_{DOR}

Рисунок 2.2 — Пример графа зависимостей каналов для двухмерной топологи 3x2 с отсутствующим узлом 6.

Теорема 2.3. *Маршрутизация, построенная на правиле порядка направлений и правиле «пузырька» является бездедлоковой.*

Рассмотрим возможность нарушения правила порядка направлений в некоторых местах коммуникационной сети, не приводящих к дедлокам. На рисунке 2.2а можно увидеть, что в сети отсутствуют маршруты из узла 5 в узел 3 и из узла 4 в узел 3, так как нет ни одного маршрута, не нарушающего правило порядка направлений. Эта ситуация может быть исправлена путём добавления дополнительного ребра в графе зависимостей каналов $(5, +Y) \longrightarrow (2, -X)$, нарушающего правило порядка направлений, но не создающего дополнительных циклов.

Перейдем к обобщению приведенного примера по добавлению ребер графа зависимостей каналов, не создающих новых циклов. Обозначим $D(P_{u,v})$ – набор направлений тора, участвующих в маршруте $P_{u,v}$, где $u, v \in N$.

Определение 2.14. *Используемым набором направлений $B(u_i, D_i)$ в сети $I(N,C)$ для заданных узла сети u_i и направления D_i называется множество, состоящее из объединения наборов направлений тора $D(P_{u,v})$ всех маршрутов*

$P_{u,v}$, проходящих через узел u_i и совершающих шаг из этого узла в направлении D_i. То есть $B(u_i, D_i) = \bigcup D(P_{u,v})$, $\forall u,v \in N$: маршрут $P_{u,v}$ содержит шаг из узла u_i в направлении D_i.

Так для CDG графа на рисунке 2.2б для вершины $(2, +X)$ используемый набор направлений $B(2, +X) = \{+X, +Y, -X, -Y\}$.

Рассмотрим и докажем следующую теорему.

Теорема 2.4. *Пусть граф зависимостей каналов $G(C, E)$ построен для сети $I(N, C)$ и маршрутизации R_{DOR}. Дополнительное ребро $e = ((u_i,D_i),(u_j,D_j))$, где $e \notin E, u_i + D_i = u_j$, не создаст дополнительных циклов в G, если $D_i \notin B(u_j, D_j)$.*

Доказательство. Предположим, что дополнительное ребро $((u_i,D_i), (u_j,D_j))$ создало цикл в графе G и $D_i \notin B(u_j, D_j)$, но это бы означало что существует путь из вершины (u_i,D_i) в вершину (u_j,D_j), что в свою очередь означает, что $D_i \in B(u_j, D_j)$ – получаем противоречие. ■

Например, дополнительное ребро $((5, -Y),(2, -X))$ в графе CDG на рисунке 2.2б может быть добавлено, потому что направление $-Y \notin B(2, -X)$.

Теорема 2.5. *Функция маршрутизации R_{DOR^*}, индуцированная модифицированным графом зависимостей каналов, является бездедлокой.*

Доказательство. В случае, если в маршрутизации R_{DOR^*} возникнут дедлоки, то это значит, что дедлоки появились после добавления нового ребра. Однако так как добавленное новое ребро удовлетворяет теореме 2.4, то в соответствующем графе CDG не образуется новых циклов, и по теоремам 2.2 и 2.3 функция маршрутизации R_{DOR^*} останется бездедлоковой. ■

Псевдокод алгоритма построения дополнительных ребер в графе зависимостей каналов CDG для исследуемой сети с маршрутизацией R_{DOR} представлен на рисунке 2.3. Алгоритм состоит из двух этапов.

На первом этапе с помощью алгоритма поиска вширь для каждой вершины графа CDG вычисляется используемый набор направлений (процедура build_used_direction_set на рисунке 2.3). Для этого из каждой вершины графа CDG запускается алгоритм поиска вширь в этом графе и запоминаются все пройденные направления.

На втором этапе происходит попытка поиска всевозможных дополнительных ребер в графе CDG, удовлетворяющих теореме 2.4. Для этого для каждой

```
Input:
CDG(C, E)  -- граф зависимости каналов для заданной сети с маршрутизацией,
              основанной на правиле порядка направлений
~CDG(C, E) -- граф CDG(C,E) с инвертированными ребрами
min_dir    -- номер направления, с которого необходимо начинать поиск.
              Остальные направления, по порядку меньшие данного, которые не будут
              учитываться в алгоритме

Output:
CDG -- граф зависимостей каналов с дополнительными ребрами, не создающими циклов

B(u) -- список используемых направлений для каждой вершины u графа CDG (изначально
     пустой)

build_used_direction_set(CDG):
    foreach v in C:
        foreach u in BFS(CDG, v):
            B(v).push_back(u.get_direction())

rebuild_used_direction_set(CDG, v1, v2):
    foreach u in BFS(~CDG, v1):
        B(u) = B(u) + B(v2)

add_additional_edges(CDG, min_dir):
    foreach v1, v2 in C:
        if v1.get_node() + v1.get_direction() != v2.get_node():
            continue
        if v1.get_direction() < min_dir:
            continue
        if not v1.get_direction() in B(v2):
            v1.add_neighbor(v2)
            rebuild_used_direction_set(CDG, v1, v2)

build_used_direction_set(CDG)
add_additional_edges(CDG, min_dir)
```

Рисунок 2.3 — Псевдокод алгоритма построения дополнительных ребер в CDG. Комментарии: v.get_node() – узел сети, соответствующий вершине v графа CDG; v.get_direction() – направление сети для вершины v графа CDG.

вершины $(u_i,D_i) \in C$ графа CDG проводится поиск вершины $(u_j,D_j) \in C$, такой что $u_j = u_i + D_i$ и $D_i \notin B(u_j,D_j)$. Если такая вершина найдена, то

ребро $((u_i,D_i),(u_j,D_j))$ добавляется в граф (процедура add_additional_edges на рисунке 2.3).

После добавления ребра необходимо заново вычислить используемый набор направлений для каждой вершины. Для этого необходимо обновить используемый набор направлений для тех вершин, пути из которых проходят через (u_i,D_i). Это можно сделать с помощью алгоритма поиска вширь, запущенного из вершины (u_i,D_i) по графу CDG с инвертированными ребрами $\bar{E}$, при этом добавляя $B(u_j,D_j)$ к каждой посещенной вершине (процедура rebuild_used_direction_set на рисунке 2.3).

Во время исследований установлено, что для сети Ангара наилучший результат достигается, когда сначала строятся все дополнительные ребра для отрицательных направлений, то есть $min\_dir = -X$ в рамках обозначений согласно рисунку 2.3; а затем для всех остальных направлений, то есть $min\_dir = +X$.

Теорема 2.6. *Маршрутизация R_{DOR^*}, индуцированная модифицированным графом зависимостей каналов при помощи алгоритма построения дополнительных ребер, является бездедлоковой.*

Доказательство. Допустим, что после очередного добавленного дополнительного ребра в полученной маршрутизации R_{DOR^*} появились маршруты, образующие дедлок, тогда это бы означало, что в соответствующем графе зависимостей каналов появился новый цикл, что противоречит теореме 2.4. ■

Для возможности исследования маршрутов в сети Ангара с нарушением правила порядка направлений необходимо модифицировать маршрутный граф $RG(V,E)$, алгоритм построения которого разработан в разделе 2.2. Для его модификации нужно построить для исследуемой коммуникационной сети I соответствующий граф зависимости каналов CDG, в котором при помощи предложенного алгоритма на рисунке 2.3 построить дополнительные ребра, не создающие циклы. Далее для каждого построенного дополнительного ребра в графе CDG необходимо добавить в маршрутный граф дополнительные ребра для вершин, соответствующих первому или последнему нестандартному шагу и добавленному ребру.

Формально описать алгоритм `RoutingGraphAdd` добавления ребер в маршрутном графе можно следующим образом. Каждое новое построенное ребро в графе CDG $((u_i,D_i),(u_j,D_j))$ можно представить в виде (D_i,u_j,D_j), так как

$u_j = u_i + D_i$. Напомним, что данное ребро позволяет сделать новый поворот в маршруте при помощи нарушения правила порядка направлений, то есть $D_i > D_j$. Нарушить правило порядка направлений возможно только, если сделать первый шаг при помощи First Step, либо последний шаг при помощи Last Step. Таким образом, для варианта, когда D_i – первый положительный шаг, необходимо для соответствующей вершины маршрутного графа $U^j_{FS_i}$ построить ребро в вершину $U^k_{D_j}$ такую, что $u_k = u_j + D_j$. Для варианта, когда D_j – последний отрицательный шаг, необходимо для соответствующих вершин маршрутного графа U^j_{dirbit} таких, что D_i – последнее направление в наборе $dirbit$, построить ребро в вершину $U^k_{LS_j}$ такую, что $u_k = u_j + D_j$.

Определение 2.15. *Маршрутизация R_{A^*} с нарушением правила порядка направлений для FS/LS для сети I определяется функцией, которая индуцирована маршрутным графом с добавленными ребрами при помощи алгоритма RoutingGraphAdd.*

Теорема 2.7. *Маршрутизация R_{A^*} является бездедлоковой.*

Доказательство. Маршрутизация R_{A^*} является усеченным вариантом маршрутизации R_{DOR^*}, которая, в свою очередь, является бездедлоковой. Поэтому маршрутизация R_{A^*} является бездедлоковой.■

Теорема 2.8. *Маршрут $P_{u,v}$ из узла u в узел v сети I, удовлетворяющий маршрутизации R_{A^*}, существует тогда и только тогда, когда в графе $RG(V, E)$, соответствующем маршрутизации R_{A^*}, существует путь из вершины U^a_{begin} в вершину U^b_{end}.*

Доказательство теоремы 2.8 аналогично доказательству теоремы 2.1. По доказанной теореме построенный граф $RG(V,E)$ является маршрутным.

Временная сложность алгоритма построения маршрутного графа

Сложность алгоритма построения маршрутного графа для сети $I(N,C)$ складывается из сложности построения маршрутного графа без нарушения правила порядка направлений и сложности алгоритма `RoutingGraphAdd` построения дополнительных ребер в маршрутном графе для возможности нарушения правила порядка направлений.

Маршрутный граф $RG(V,E)$ без нарушения правила порядка направлений строится при помощи прохода по всем вершинам графа и построением ребер в соответствии с правилами маршрутизации R_{A^-}, поэтому сложность данного этапа $T_{RG} = O(|V|+|E|) = O(N*(C_V+C_E))$, где C_V и C_E – количество вершин и ребер для каждого вычислительного узла сети, см. 2.1 и 2.2. Здесь и далее под N будет пониматься количество узлов в сети.

Сложность второго этапа T_{CDG} – это сложность построения модифицированного графа зависимостей каналов, которая складывается, в свою очередь, из:

1. Сложность построения графа CDG составляет $O(|V_{CDG}| + |E_{CDG}|)$.
   а) Количество вершин $|V_{CDG}|$ графа CDG – это количество каналов связи в сети. Так как из каждого узла выходит $2n$ каналов связи, то $|V_{CDG}| = N * 2n$.
   б) Количество ребер $|E_{CDG}|$ графа CDG – это всевозможные по всем узлам сети переходы, удовлетворяющие правилу порядка направлений. Так как из каждого направления D_i можно перейти в любое направление D_j при условии $1 \leqslant i \leqslant j \leqslant 2n$, то для каждого узла сети u (или вершин (u, D_i) графа CDG) можно построить $\sum_{i=1}^{N}\sum_{j=i}^{N} = (1+2n)*n$ ребер с другими вершинами графа CDG. Таким образом, $|E_{CDG}| = N*(1+2n)*n$.

   Поэтому $O(|V_{CDG}| + |E_{CDG}|) = O(N * (3n + 2n^2)) = O(C_2(n) * N)$, где $C_2(n) = 3n + 2n^2$, N – количество узлов в сети.
2. Сложность построения используемого набора направлений для каждой вершины графа CDG – это сложность алгоритма поиска вширь по этому графу CDG из каждой его вершины, а именно: $O(|V_{CDG}| * (|V_{CDG}| + |E_{CDG}|)) = O(N * 2n * (N * (3n + 2n^2))) =$
   $= O(2n * C_2(n) * N^2)$.
3. С точки зрения временной сложности худший случай – это добавление всех ребер графа CDG, не удовлетворяющих правилу DOR, как

новых при помощи процедуры в алгоритме `RoutingGraphAdd`. Таких ребер для каждого узла сети $(2n)^2 - (1 + 2n) * n = 2n^2 - n$. Сложность добавления одного ребра состоит из обновления используемого набора направлений при помощи поиска вширь и составляет $O(C_2(n) * N)$. Поэтому временная сложность добавления новых ребер графа CDG составляет $O(N * (2n^2 - n) * N * C_2(n)) = O((2n^2 - n) * C_2(n) * N^2)$.

Итоговая временная сложность алгоритма построения маршрутного графа составляет

$$\begin{aligned} T_{RG} + T_{CDG} = O[&(C_V(n) + C_E(n)) * N + \\ &+ C_2(n) * N + \\ &+ 2n * C_2(n) * N^2 + \\ &+ (2n^2 - n) * C_2(n) * N^2] = O[(n + 2n^2) * (2n + 2n^2) * N^2], \end{aligned}$$

где N – количество узлов в сети, n – количество измерений сети.

2.4 Алгоритм определения достижимости

В данном разделе формулируется задача определения достижимости множества узлов и описывается предложенный в работе алгоритм её решения. Для сети I и заданной функции маршрутизации R_{A^*} требуется разработать алгоритм определения достижимости множества узлов $M_{a,t}$.

Сведем задачу определения достижимости множества $M_{a,t}$ в сети $I(N,C)$ к поиску пути в маршрутном графе $RG(V, E)$. При помощи маршрутного графа задача определения достижимости множества узлов $M_{a,t} \subseteq N$ сводится к определению достижимости множества вершин, соответствующих узлам множества $M_{a,t}$ в маршрутном графе RG. В маршрутном графе временно удаляются все ребра, ведущие из вершин для узлов $M_{a,t}$ к вершинам других узлов сети. Для каждой из вершин U^i_{begin} графа, соответствующих узлам сети из множества M_a, при помощи поиска вширь в графе определяется множество достижимых из нее конечных вершин вида U^j_{end}, данное множество должно содержать вершины для всех узлов множества M_a. При этом транзитные узлы будут принадлежать $M_{a,t}$.

Сложность временного удаления ребер, идущих от вершин узлов $M_{a,t}$ к вершинам других узлов сети выражается как

$$T_{init} = O(C_E|M_{a,t}|).$$

Алгоритм поиска вширь имеет сложность

$$T_{BFS} = O(V + E) = O(C_V|M_{a,t}| + C_E|M_{a,t}|) = O((C_V + C_E)|M_{a,t}|).$$

Так как алгоритм определения достижимости нужно выполнить для всех вершин M_a, то его сложность

$$T = O((C_V + C_E)|M_{a,t}||M_a|) = O(C(n)|M_{a,t}||M_a|),$$

где $C_V = (3^n + 2n + 1)$ (формула 2.1) и $C_E = 2n3^n + 1.5n^2 + 1.5n + 1$ (формула 2.2) – константы, введенные на этапе определения маршрутного графа,

$$C(n) = C_V + C_E = (2n + 1)3^n + 1.5n^2 + 3.5n + 2, \tag{2.3}$$

где n – размерность топологии тор.

2.5 Выводы

В данной главе формально определена маршрутизация R_A в сети Ангара, которая допускает возникновение дедлоков в сети. Предложен алгоритм построения графа, при помощи которого определена функция маршрутизации R_{A^*}. Для построенного графа доказана теорема о взаимооднозначном соответствии наличия пути в графе и в конкретной сети, которую граф представляет. Поэтому данный граф является маршрутным. Временная сложность алгоритма построения маршрутного графа составляет $O(N^2)$, где N – количество узлов в сети. Построенный маршрутный граф позволяет решать задачу анализа маршрутов в сети с маршрутизацией, в которой возможность совершения того или иного шага зависит от истории маршрута сетевого пакета, предложен маршрутный граф. Доказано, что построенная функция маршрутизации R_{A^*} является бездедлоковой.

Таким образом, задача определения достижимости множества узлов сети сведена к определению достижимости соответствующих вершин в графе, для решения этой задачи разработан полиномиальный алгоритм, основанный на поиске вширь в графе. Временная сложность предложенного алгоритма $O(|M|^2)$, где $|M|$ – количество вычислительных узлов проверяемой системы.

Необходимо отметить, что предложенный маршрутный граф, а также алгоритм определения достижимости множества узлов применимы не только для сети Ангара, но и для сетей серии IBM Blue Gene.

Глава 3. Алгоритмы построения таблиц маршрутов для решения задачи равномерного распределения трафика

Маршрутизация в сети Ангара позволяет задать различные маршруты между двумя фиксированными узлами вычислительной системы. Во время запуска задачи необходимо создать таблицу маршрутизации, в которой для каждого узла-получателя должен быть выбран единственный маршрут. При этом необходимо, чтобы маршруты были равномерно распределены по сети, то есть сбалансированы. Для формальной постановки задачи разработки алгоритма построения таблиц маршрутов для решения задачи равномерного распределения трафика необходимо определить ряд понятий.

Определение 3.1. *Таблицей маршрутов RT достижимого множества узлов* $M_{a,t} = M_a \cup M_t$ *сети I назовем множество маршрутов* $P_{u,v}$ *таких, что для любых двух активных узлов* $u, v \in M_a \; \exists \,!\; P_{u,v} \in RT$.

Так как маршрутов между каждой парой вычислительных узлов может быть несколько, необходимы критерии оценки предложенной таблицы маршрутов. Для этого в работе используются следующие характеристики таблицы маршрутов.

Определение 3.2. *Диаметром достижимого множества узлов* $M_{a,t}$ *сети с таблицей маршрутов RT назовем максимальную длину маршрута в RT.*

Определение 3.3. *Загруженностью* G_c *канала связи с для таблицы маршрутов RT в сети I будем называть количество маршрутов* $P_{i,j}$*, которым принадлежит данный канал связи:* $G_c = \mid \{P_{i,j} : c \in P_{i,j}, P_{i,j} \in RT\} \mid$.

Определение 3.4. *Множеством каналов связи* $C(M_{a,t})$ *множества узлов* $M_{a,t}$ *сети* $I(N,C)$ *назовем все такие* $(u,D) \in C, u \in M_{a,t}$*, что* $\exists v \in M_{a,t} : v = u + D$.

Определение 3.5. *Минимальной таблицей маршрутов* RT_{min} *назовем такую таблицу маршрутов, которая состоит из набора маршрутов минимальных длин.*

Определение 3.6. *Идеальной загруженностью канала связи* $\pi_{perfect}$ *множества узлов* $M_{a,t}$ *сети* I *назовем среднюю загруженность каналов связи сети для любой минимальной таблицы маршрутов* RT_{min} *для* $M_{a,t}$*:* $\pi_{perfect} = \frac{\sum_{c \in C(M_{a,t})} G_c}{|C(M_{a,t})|}$.

Заметим, что идеальная загруженность канала связи $\pi_{perfect}$ не зависит от вида таблицы маршрутов RT_{min}. Показать это можно от противного. Предположим, что $\pi_{perfect}(RT_{min1}) \leqslant \pi_{perfect}(RT_{min2})$, это бы означало, что $\exists P^1_{u,v} \in RT^1_{min}$ и $P^2_{u,v} \in RT^2_{min}$, что длина маршрута $P^1_{u,v}$ меньше длины $P^2_{u,v}$, что в свою очередь означает, что $P^2_{u,v}$ неминимальный по длине маршрут, а значит таблица маршрутов RT^2_{min} содержит неминимальные по длине маршруты.

Определение 3.7. *Максимальной загруженностью канала связи для таблицы маршрутов* RT *достижимого множества узлов* $M_{a,t}$ *сети* I *назовем* $\pi_{max}(RT) = \max\limits_{c \in C(M_{a,t})} G_c$.

Определение 3.8. *Отклонением* $\sigma(k,RT)$ *степени* k *таблицы маршрутов* RT *множества узлов* $M_{a,t}$ *сети* I *от идеальной загруженности назовем:*

$$\sigma(k,RT) = \sqrt[k]{\frac{1}{\mid C(M_{a,t}) \mid} \sum_{(u,D) \in C(M_{a,t})} |\pi_{perfect} - G_{(u,D)}|^k}.$$

Характеристика $\sigma(k,RT)$ положительна на всём интервале и достигает 0 в случае, когда загруженность каналов связи соответствует идеальной загруженности.

Допустим, что число маршрутов между двумя узлами ограничено некоторым числом N_{paths}, тогда существует $N_{paths}^{(|M_a|-1)|M_a|}$ различных таблиц маршрутов. Даже при небольшом числе узлов сети и вариантов маршрутов числе маршрутов между двумя узлами количество различных таблиц маршрутов очень велико, и требуется специальный алгоритм для выбора таблиц маршрутов. Дополнительную сложность добавляет несимметричность рассматриваемой маршрутизации сети Ангара и несимметричность системы ввиду наличия отказов.

Для оценки сбалансированности таблицы маршрутов возможно использовать следующие критерии:

1. Минимизация диаметра построенной таблицы маршрутов;
2. Минимизация максимальной загруженности канала связи $\pi_{max}(RT)$;

3. Минимизация отклонения $\sigma(k,RT)$ степени k построенной таблицы маршрутов от идеальной загруженности [94].

Первый критерий возникает из-за того, что в сети с минимальным диаметром задержка на передачу данных будет наименьшей. Второй и третий критерий следуют из стремления получить равномерно загруженную систему.

Ранее был разработан базовый алгоритм построения таблиц маршрутов, который подробно описан в следующем разделе 3.1. Этот алгоритм построен при помощи набора правил в зависимости от топологии сети, распределение сетевого трафика при этом зависит от координат начального узла. Алгоритм основан на идеях балансировки сетевого трафика, описанных в книге [34]. Данный алгоритм построения таблиц маршрутов не учитывает сломанные или недоступные ресурсы сети.

Задача построения таблиц маршрутов для приближенного решения задачи балансировки трафика формулируется следующим образом. Для достижимого множества узлов $M_{a,t}$ сети I с заданной функцией маршрутизации R_{A^*} требуется разработать алгоритм построения сбалансированных таблиц маршрутов RT, при этом необходимо оценить пригодность алгоритма с точки зрения применения на практике:

1. По времени выполнения алгоритма;
2. По критерию $\pi_{max}(RT)$ оценки сбалансированности таблицы маршрутов по сравнению с базовым алгоритмом для сетей без отказов.

Исследование пригодности алгоритмов проводится в разделе 5.2.

3.1 Базовый алгоритм построения таблицы маршрутов

В базовом алгоритме таблица маршрутов описывается набором правил в зависимости от топологии сети I. Данный метод построения таблиц маршрутов не учитывает сломанные или недоступные ресурсы сети. Распределение сетевого трафика основано на изменении направления в зависимости от координат начального узла по аналогии с [34]. Псевдокод алгоритма представлен на рисунке 3.1. Оператор $sum(src)$ означает сумму координат узла src.

Рассмотрим топологию одномерное кольцо с длиной измерения d_1 кратной двум. Между двумя узлами, находящимися друг от друга на расстоянии $d_1/2$ шагов, есть два варианта коротких маршрутов: в положительном направлении и

```
Input:
    I  -- конфигурация сети,
        I.n -- размерность сети
        I.d[] -- массив размеров измерений
    Ma, Mt -- множества активных и транзитных узлов

Output:
    RT -- таблица маршрутизации

construct_RT(I, Ma)
{
    RT = []
    for src in Ma:
        for dst in Ma:
            if src == dst:
                continue
            diff = [0] * I.n # массив размером I.n, заполненный 0
            for i in range(I.n):
                if I.d[i] > 2:
                    diff[i] = (dst[i] + I.d[i] - src[i]) % I.d[i]
                    if 2 * diff[i] > I.d[i]:
                        diff[i] = diff[i] - I.d[i]
                    if 2 * diff[i] == I.d[i] and sum(src) % 2 == 1:
                        diff[i] = diff[i] - I.d[i]
                else:
                    diff[i] = dst[i] - src[i]
            for i in range(I.n):
                if diff[i] > 0:
                    RT[src][dst] += "+" + "xyzk"[i]
            for i in range(I.n):
                if diff[i] < 0:
                    RT[src][dst] += "-" + "xyzk"[i]
    return RT
}
```

Рисунок 3.1 — Псевдокод базового алгоритма построения таблицы маршрутов.

в отрицательном. Для балансировки трафика базовый алгоритм меняет направление движения в зависимости от суммы координат начального узла. Данный способ балансировки вносит ограничение на выбор достижимого множества. Если в кольце выбрано подряд больше, чем $d_1/2$ узлов, то часть трафика пойдет через остальные узлы в кольце. Пример такой ситуации представлен на

рисунке 3.2. На рисунке стрелками показаны только самые длинные маршруты в рассматриваемой сети. Жирной линией показаны каналы связи. Крупным штрихом выделено множество активных узлов. Мелким штрихом показаны маршруты, которые пойдут вне выделенного множества узлов. Все рассуждения аналогично распространяются и на остальные размерности. Таким образом, достижимым множеством узлов с трафиком внутри данного множества будет являться любое множество узлов, которое можно вписать в прямоугольник со сторонами меньше или равными половины длины измерения, округленного до ближайшего большего целого, или равному длине измерения.

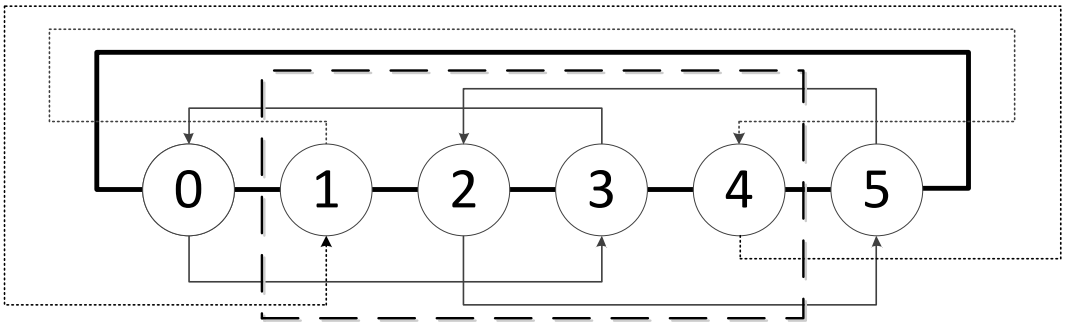


Рисунок 3.2 — Пример проблемы выделения достижимого множества узлов при построении таблицы маршрутов базовым алгоритмом в сети с топологией одномерный тор с длиной измерения 6.

К достоинствам базового алгоритма построения таблицы маршрутов можно отнести, во-первых, простоту реализации с точки зрения вычислительной сложности алгоритма, что немаловажно для реальных вычислительных систем, так как сокращается время на запуск задачи, во-вторых, что все маршруты минимальны по длине.

К недостаткам базового алгоритма построения таблицы маршрутов можно отнести:

1. В случае возникновения отказа (поломка канала связи или узла) пропадает достижимость множества вычислительных узлов, использующих для маршрутов отказавшие элементы, в то время как могут существовать другие маршруты в обход отказавших элементов.
2. Для множества узлов $M_{a,t}$ может быть не построена таблица маршрутов, в то время, как, в действительности, это множество является достижимым.

3.2 Алгоритм построения таблиц маршрутов на основе поиска вширь в графе

Для устранения недостатков базового алгоритма в данной работе разработан алгоритм построения сбалансированных таблиц маршрутов на основе поиска вширь в маршрутном графе. Псевдокод алгоритма представлен на рисунках 3.3 и 3.4. На вход алгоритму кроме сети I и множества $M_{a,t}$ подается состояние сети S, которое описывает недоступные узлы и каналы связи, особенно это важно для ресурсов «внутри» $M_{a,t}$. Для лучшего понимания необходимо отметить, что под узлом понимается вычислительный узел сети, а под вершиной – вершина в маршрутном графе. Также напомним, что каждому узлу сети соответствует набор вершин маршрутного графа.

По заданному количеству транзитных узлов их множество выбирается случайно внутри множества $M_{a,t}$ узлов сети $I(N,C)$. В соответствующем маршрутном графе RG сети I временно удаляются все ребра, ведущие из вершин для узлов $M_{a,t}$ к вершинам других узлов сети, а также все ребра, соответствующие недоступным каналам связи и ведущие к недоступным узлам из состояния сети S.

Изначально все каналы связи $C(M_{a,t})$ имеют нулевую загруженность. Для каждого узла u достижимого множества M_a в маршрутном графе $RG(V,E)$ запускается алгоритм поиска вширь. После окончания поиска из каждого узла множества $M_{a,t}$ необходимо подняться по построенному дереву обратно вверх к узлу u, увеличивая при этом загруженность $G_{u,D}$ проходимых каналов связи сети. Напомним, что каждое ребро графа G соответствует некоторому каналу связи в сети. Эвристически выяснено, что сбалансированная таблица маршрутов получается, если в качестве следующего узла для запуска поиска вширь выбирать максимально удаленным от узла u. Вторая эвристика, введенная для получения более равномерной загрузки каналов связи, заключается в сортировке вершин на каждом новом слое поиска вширь по возрастанию загруженности каналов связи, соответствующих вершинам. Заметим, поиск вширь в графе находит кратчайший путь, поэтому построенные маршруты в сети между каждой парой узлов будут минимальными.

```
Input:
    I      -- конфигурация сети
    S      -- состояние сети
    Ma, Mt -- множества активных и транзитных узлов

Output:
    RT -- таблица маршрутов

construct_RT_bfs(I, S, Ma, Mt)
{
    RT = []
    G = RoutingGraph(I, S, Ma + Mt)
    V = Ma[0]
    do {
        RT += path_by_BFS(G, V)
        mark_processed(V)
        V = get_max_remote(V)
    } while not all_processed(Ma):
    return RT
}

get_max_remote(V)
{
    maxl = V
    for n in not_processed(Ma) {
        if length(V, maxl) < length(V, n) {
            maxl = n
        }
    }
    return maxl
}
```

Рисунок 3.3 — Псевдокод алгоритма построения таблицы маршрутов на основе поиска вширь в графе.

Сортировку слоев в алгоритме можно оценить как

$$O(\sum_{i=1}^{l}(|V_i|log_2|V_i|)) = O(\sum_{i=1}^{l}(|V_i|log_2|V|)) = O(|V|log_2|V|),$$

где l – число слоев в алгоритме, V_i – множество вершин на каждом слое.

Сложность одного прохода разработанного алгоритма можно оценить как сумму трех слагаемых: $T_1 = O(C(n)|M_{a,t}|)$ для поиска вширь в гра-

```
Input:
G -- маршрутный граф
startV -- стартовая вершина

Output:
paths -- набор путей из вершины startV

path_by_BFS(Graph G, startV)
{
    layers[0] = [startV]
    iter = 0
    while length(layers[iter]) != 0 {
        for n in layers[iter]:
            layers[iter + 1] += G.get_neighbor(n)
        sort(layers[iter + 1])
        iter ++
    }
    iter --
    paths = []
    for n in layers[iter] {
        endV = n
        nextE = n
        path = [endV]
        while nextE != startV {
            next_edge_tmp = endV.get_previous()
            G.vertex(nextE, next_edge_tmp) ++
            nextE = next_edge_tmp
            path += nextE
        }
        path.revert()
        paths += path
    }
    return paths
}
```

Рисунок 3.4 — Псевдокод алгоритма построения таблицы маршрутов на основе поиска вширь в графе. Вспомогательная функция.

ф е $RG(V, E)$; $T_2 = O(|V|log_2|V|)$ – сортировка вершин на каждом шаге поиска; $T_3 = O(L_{max}|M_{a,t}|)$ – вычисление загруженности каналов связи.

$T_{BFS} = |M_a| * (T_1 + T_2 + T_3) = O(C(n)|M_{a,t}||M_a|)$, где $C(n)$ – константа 2.3, зависящая от размерности сети I.

3.3 Генетический алгоритм построения таблиц маршрутов

Генетический алгоритм – это эвристический алгоритм поиска, основанный на идеях эволюционных теорий. Переменные, характеризующие решение задачи, представлены в виде генов в хромосоме индивидуума. Генетический алгоритм оперирует конечным множеством решений (популяцией) и генерирует новые решения как различные комбинации генов индивидуумов, используя такие операции как отбор, рекомбинация (кроссинговер) и мутация.

Сведем задачу построения сбалансированной таблицы маршрутов в множестве узлов $M_{a,t}$ сети I с состоянием S к понятиям генетического алгоритма:

- *Ген* – функциональная единица индивидуума, в текущей задаче это маршрут P_{ij} между двумя узлами сети. Обозначим ген $\mathcal{G}_k^l$, где k – позиция гена в индивидууме, характеризующая пару узлов сети u_i и u_j (так как число узлов конечно, то всевозможные пары узлов можно однозначно отобразить на набор натуральных чисел), l – вариант маршрута между узлами u_i и u_j;
- *Индивидуум* – набор генов, которые определяют основные свойства организма. В данном случае это таблица маршрутов с набором маршрутов (генов). Каждый индивидуум можно представить как вектор генов:

$$\mathcal{I} = (\mathcal{G}_1^{\lambda(1)}, \mathcal{G}_2^{\lambda(2)}, ..., \mathcal{G}_{L-1}^{\lambda(L-1)}, \mathcal{G}_L^{\lambda(L)}),$$

  где длина вектора $L = (|M_a| - 1)|M_a|$ равна числу маршрутов, $\lambda(k)$ – функция вариантов маршрутов, выбранных для индивидуума $\mathcal{I}$;
- *Популяция* – набор различных индивидуумов: $(\mathcal{I}_1, ..., \mathcal{I}_S)$, где S – размер популяции;
- *Кроссинговер* – операция, при которой индивидуумы обмениваются частью генов, не меняя их позицию в индивидууме;
- *Мутация* – эвристическая операция изменения индивидуума путем случайного изменение гена или набора генов. Под изменением гена подразумевается замена его вида на другой возможный;

– *Пригодность* – функция качества индивидуума, экстремум которой необходимо найти. Для генетического алгоритма в качестве функции пригодности используется отклонение k-степени $\sigma(k,RT)$.

Также, как и в алгоритме на основе поиска вширь, в генетическом алгоритме по заданному количеству транзитных узлов их множество выбирается случайно внутри множества $M_{a,t}$.

Общая схема используемого генетического алгоритма представлена на рисунке 3.5. На рисунке 3.6 представлены некоторые элементы генетического алгоритма. Начальная популяции генерируется случайным образом.

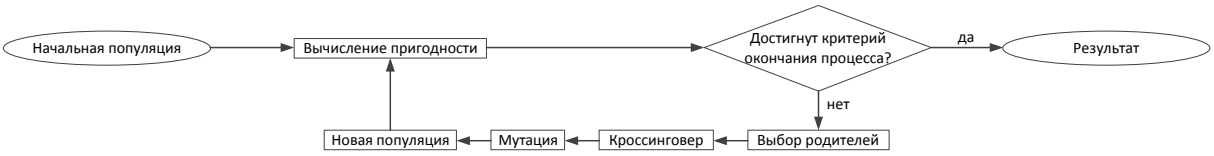
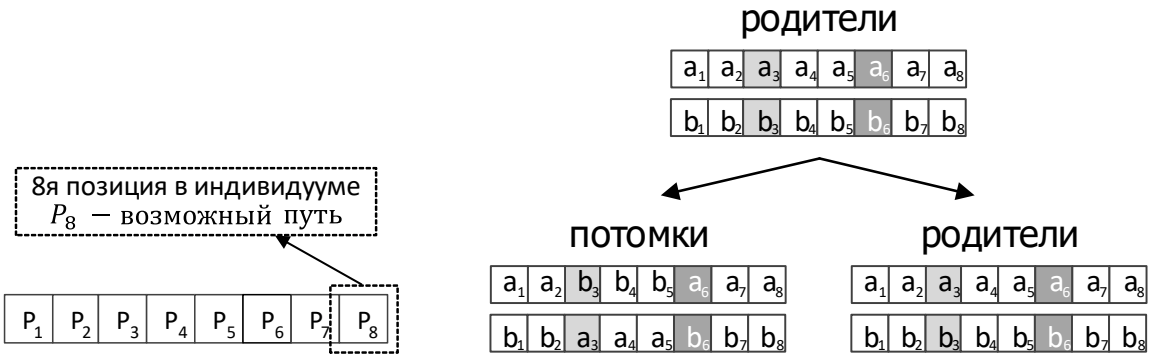


Рисунок 3.5 — Общая схема генетического алгоритма.



а) Ген

б) Операция двухточечного кроссинговера

Рисунок 3.6 — Элементы генетического алгоритма.

В разработанном генетическом алгоритме реализована операция выбора родителя – *панмиксия*. В соответствии с ним каждому члену популяции сопоставляется случайное целое число на отрезке $[1,generation\_size]$, где $generation\_size$ – размер популяции. Это число – номер партнера из популяции. При таком выборе некоторые члены популяции не будут участвовать в процессе размножения, так как образуют пару сами с собой. Какие-то члены примут участие в процессе размножения неоднократно.

После выбора партнеров происходит размножение путем двухточечного кроссинговера. Для этого в индивидууме выбираются случайно две точки и происходит обмен между особями наборами генов, ограниченных этими точками.

Схематично процедура представлена на рисунке 3.6б. Таким образом получается два новых потомка. Обозначим точки кроссинговера соответственно $\mathcal{K}_1$ и $\mathcal{K}_2$. Тогда из двух родителей $\mathcal{I}_{parent_1}$ и $\mathcal{I}_{parent_2}$:

$$\mathcal{I}_{parent_1} = (\mathcal{G}_1^{\lambda(1)}, \mathcal{G}_2^{\lambda(2)}, ..., \mathcal{G}_{L-1}^{\lambda(L-1)}, \mathcal{G}_L^{\lambda(L)})$$
$$\mathcal{I}_{parent_2} = (\mathcal{G}_1^{\tau(1)}, \mathcal{G}_2^{\tau(2)}, ..., \mathcal{G}_{L-1}^{\tau(L-1)}, \mathcal{G}_L^{\tau(L)})$$

получим двух потомков:

$$\mathcal{I}_{child_1} = (\mathcal{G}_1^{\lambda(1)}, ..., \mathcal{G}_{\mathcal{K}_1-1}^{\lambda(\mathcal{K}_1-1)}, \mathcal{G}_{\mathcal{K}_1}^{\tau(\mathcal{K}_1)}, ..., \mathcal{G}_{\mathcal{K}_2-1}^{\tau(\mathcal{K}_2-1)}, \mathcal{G}_{\mathcal{K}_2}^{\lambda(\mathcal{K}_2)}, ..., \mathcal{G}_L^{\lambda(L)})$$
$$\mathcal{I}_{child_2} = (\mathcal{G}_1^{\tau(1)}, ..., \mathcal{G}_{\mathcal{K}_1-1}^{\tau(\mathcal{K}_1-1)}, \mathcal{G}_{\mathcal{K}_1}^{\lambda(\mathcal{K}_1)}, ..., \mathcal{G}_{\mathcal{K}_2-1}^{\lambda(\mathcal{K}_2-1)}, \mathcal{G}_{\mathcal{K}_2}^{\tau(\mathcal{K}_2)}, ..., \mathcal{G}_L^{\tau(L)}).$$

После процедуры размножения к потомкам применяется операция мутации. Каждый ген с вероятностью q_m может мутировать. Партнеры и потомки образуют новую популяцию, к которой применяется операция селекции: для каждого индивидуума вычисляется функция качества $\sigma(k,\mathcal{I})$, в новой популяции останутся индивидуумы с наименьшим значением качества. Наилучшую сходимость алгоритм показал для отклонения $\sigma(4,\mathcal{I})$ степени 4.

Критерии окончания процесса выбраны следующим образом:

1. Максимальное число поколений (итераций алгоритма) – 30 поколений;
2. Скорость сходимости функции качества – 0,05, т.е. алгоритм завершается, если наилучшее значение качества отличается от наилучшего значения качества следующего поколения менее, чем на 0,05.

На рисунке 3.7 представлен псевдокод генетического алгоритма построения таблицы маршрутов. Оценка сложности алгоритма складывается из оценки каждого этапа. Обозначим размер исходной популяции $generation\_size$ более кратко как g.

Операцию панмиксия можно оценить как $T_{parent\_selection} = O(g)$ операций бросания кости. В результате работы алгоритма образуется максимум g пар.

Оценка сложности операции размножения $T_{selection} = O(2L)$ операций размещения генов в потомках. Эту операцию необходимо проделать для каждой пары, то есть не более g раз.

Оценка сложности операции мутации реализуется за $T_{mutation} = O(L)$ операций бросания кости. Эту операцию необходимо проделать для каждого нового индивидуума, которых можно оценить как $2 \cdot g$.

Для того, чтобы вычислить функцию пригодности, необходимо вычислить загруженность каждого линка в системе. Каждый маршрут проходит

```
Input:
    I       -- конфигурация сети
    S       -- состояние сети
    Ma, Mt -- множества активных и транзитных узлов
    generation_size -- размер популяции

Output:
    RT -- таблица маршрутов

construct_RT_Genetic(I, S, Ma, Mt) {
    path = GenerateAllPath(I, S, Ma + Mt)
    start_generation = []
    next_generation = []
    generation_num = 0
    for i in generation_size {
        for src,dst in Ma {
            start_generation [i] = random(paths[src][dst])
        }
    }
    min_fitens = calculate_fitness(start_generation)
    current_min_fitnes = min_fitens
    while min_fitens != 0 or generation_num < 30 or
          abs(min_fitens - current_min_fitnes) > 0.05 {
        for parent1 in Ma {
            parent2 = random(start_generation)
            if parent1 != parent2:
                next_generation += crossing_over(parent1, parent2)
        }
        for individual in next_generation {
            individual = mutation(individual)
        }
        sort_by_fitness(next_generation)
        start_generation = next_generation[0:generation_size]
        generation_num ++
    }
    return best_individual(start_generation)
}
```

Рисунок 3.7 — Псевдокод генетического алгоритма построения таблицы маршрутов.

через некоторый набор каналов связи, число которых можно оценить сверху как максимальная длина маршрута в сети: $L_{max} = \sum_{i=1}^{n} d_i$. Для вычисления загруженности всей системы необходимо $T_{loading} = O(L \cdot L_{max})$ операций. Для вычисления отклонения степени k необходимо еще $O(|\{(u, D)\}|) = O(2n|M_{a,t}|)$ операций. Итого $T_{fitness} = O(|M_a|^2 + 2n|M_{a,t}|)$. Необходимо вычислить функцию пригодности для каждого нового индивидуума.

На последнем этапе нам необходимо отсеять все плохие организмы и оставить $generation\_size$ (или g) наилучших. Размер текущей популяции (после этапа размножения) можно оценить как $3g$, где одна часть – это старые особи, и две части – это новое поколение. Алгоритм сортировки оценивается как $T_{evolution} = O(3g \log(3g))$.

Временная сложность одной итерации алгоритма:

$$\begin{aligned} T_{GA} &= T_{parent\_selection} + g \cdot T_{selection} + 2g \cdot T_{mutation} + 2g \cdot T_{fitness} + T_{evolution} \\ &= O[g + 2g|M_a|^2 + 2g|M_a|^2 + 2g(|M_a|^2 + 2n|M_{a,t}|) + 3g \log(3g)] \\ &= O(6g|M_a|^2). \end{aligned}$$

3.4 Выводы

В данной главе рассматривается проблема построения таблицы маршрутов для решения задачи балансировки трафика. Основным критерием оценки сбалансированности таблицы маршрутов RT является минимизация максимальной загруженности канала связи в таблице маршрутов $\pi_{max}(RT)$.

В данной главе представлены два разработанных алгоритма построения сбалансированной таблицы маршрутов, а также имевшийся базовый алгоритм, с которым в дальнейшем разработанные алгоритмы будут сравниваться.

Первый алгоритм основан на обходе маршрутного графа при помощи поиска вширь. Временная сложность алгоритма $O(N^2)$, где N – число узлов в исследуемом множестве. Второй алгоритм основан на генетическом алгоритме поиска решения. Сложность одной итерации этого алгоритма – $O(gN^2)$, где g – количество индивидуумов в популяции.

Глава 4. Алгоритмы выбора достижимого множества узлов требуемого размера

Во время работы высокопроизводительной вычислительной системы необходимо в зависимости от состояния сети уметь предоставлять достижимое множество узлов требуемого размера, если это возможно. Данная глава посвящена решению этой задачи.

Определение 4.1. *Состоянием сети $I(N, C)$ будем называть пару множеств (S_N, S_C), где $S_N \subseteq N$ – множество недоступных узлов, $S_C \subseteq C$ – множество недоступных каналов связи.*

Задача выбора узлов формулируется следующим образом. Для сети $I(N,C)$ и ее состояния (S_N, S_C), функции маршрутизации R_{A^*}, чисел m и T активных и транзитных узлов требуется разработать алгоритм выбора достижимого множества вычислительных узлов $M_{a,t}$, такого что $|M_a| = m$, $|M_{a,t}| \leqslant m + T$, $M_{a,t} \cap S_N = \varnothing$.

Рассмотрим алгоритм решения задачи выбора узлов в сети $I(N, C)$ полным перебором. Всего вариантов расположения m узлов в сети $\binom{|N|}{m}$. Для проверки достижимости каждого набора узлов требуется $O(m^2)$ операций. Даже для небольших систем $\binom{|N|}{m} * O(m^2)$ очень велико, эта функция растет очень быстро, такой алгоритм не может быть применен на практике. Поэтому необходима разработка алгоритма выбора узлов, работа которого занимает приемлемое время.

В результате работы алгоритма выбора узлов на выходе может быть набор решений, поэтому необходимо выбрать наилучшее. При этом в этих решениях уже может быть предложена таблица маршрутов. Для отбора решений используются критерии в следующем порядке:

1. минимизация числа транзитных узлов;
2. минимизация фрагментированности сети после выделения множества узлов;
3. минимизация диаметра построенной таблицы маршрутов, где диаметр – это максимальная длина маршрута в исследуемом множестве;
4. минимизация максимальной загруженности $\pi_{max}(RT)$ построенной таблицы маршрутов.

Критерий о минимизации числа транзитных узлов самый важный, так как он необходим для эффективного использования аппаратных ресурсов вычислительной системы. Второй критерий также призван максимально эффективно использовать вычислительную систему. Третий критерий позволяет минимизировать максимальную задержку внутри вычислительной системы, выделяемое множество узлов при этом становится более квадратным. И последний критерий максимизирует сбалансированность получаемой таблицы маршрутов.

В разделе 4.1 описан анализ множества различных прямоугольников максимально возможного размера, которые характеризуют меру фрагментированности системы, и предложен алгоритм поиска таких прямоугольников. В разделе 4.2 описывается общий вид алгоритма перебора многомерных прямоугольников, далее в разделах 4.3 и 4.4 представлены основанные на общем алгоритме имевшийся ранее базовый алгоритм выбора узлов и разработанный алгоритм. Также в разделе 4.5 описан еще один разработанный алгоритм выбора узлов, основанный на подходе расширения многомерных прямоугольников. Таким образом, всего в данной главе предложено два новых алгоритма выбора узлов.

В дальнейшем необходимо сравнить алгоритмы по сформулированным критериям, а также оценить пригодность с точки зрения применения на практике по времени выполнения алгоритма. Исследование качества работы алгоритмов и сравнение по времени их работы будет представлено в разделе 5.4.

4.1 Оценка фрагментированности сети

Определение 4.2. *Прямоугольник максимально возможного размера в сети* $I(N,C)$ *с состоянием* (S_N, S_C) *(или прямоугольник максимального размера, обозначение ПМР, англ. Max Space Size) – это многомерный прямоугольник, состоящий только из доступных узлов (т.е. из узлов множества* $N \setminus S_N$*), который нельзя расширить ни в одну из его сторон сети.*

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 недоступным узлом. Множество различных ПМР характеризуют меру фрагментированности сети.

Алгоритм $Find\_MSSs(S)$ поиска различных ПМР реализован следующим образом. Из множества $N \setminus S_N$ выбирается узел u_1. Выбранный узел последовательно расширяется во все стороны, пока это возможно. Полученное множество узлов обозначим MSS_1. На следующем этапе выбирается узел $u_2 \in N \setminus S_N \setminus MSS_1$ и аналогичным образом строится множество MSS_2. Алгоритм продолжается до тех пор, пока множество $N \setminus S_N \setminus \bigcup\limits_{iter=1}^{Iters} MSS_{iter}$ не пусто, где $Iters$ – число итераций алгоритма. Псевдокод алгоритма представлен на рисунке 4.1. Обозначим множество различных MSS_{iter}, как $MSSs$. Важно отметить, что каждый прямоугольник строится независимо от остальных прямоугольников в предположении доступности всех изначально свободных узлов $N \setminus S_N$.

Иллюстрация работы алгоритма приведена на рисунках 4.2а и 4.2б, на которых в двумерной решетке узлы множества S_N закрашены красным. Жирным контуром на рисунке 4.2а выделен узел, из которого поочередным расширением построен двумерный ПМР MSS, который обозначен пунктирной линией. Узлы, выделенные полужирным пунктиром, соответствуют множеству узлов $N \setminus S_N \setminus MSS$, не входящих в построенный прямоугольник. Из этих узлов будут строиться последующие прямоугольники. Все построенные ПМР показаны на рисунке 4.2б.

В качестве критерия оценки сети $I(N, C)$ с состоянием $S = (S_N, S_C)$ на основе прямоугольников максимального размера вводится функция $\varphi(I, S)$ от числа найденных ПМР, которая тем больше, чем большее число прямоугольников максимального размера имеется в сети:

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

где MSS_{max}^{nnodes} – количество узлов в ПМР максимального размера, $|MSS_{max}|$ – число таких ПМР.

Для каждого найденного достижимого множества $M_{a,t}$ оценивается значение функции $\varphi(I, S')$, где S' – состояние сети после предполагаемого выделения узлов, т.е. $S' = (S_N \cup M_{a,t}, S_C \cup C(M_{a,t}))$. Для увеличения утилизации сети требуется выбирать решения с наибольшим значением функции φ.

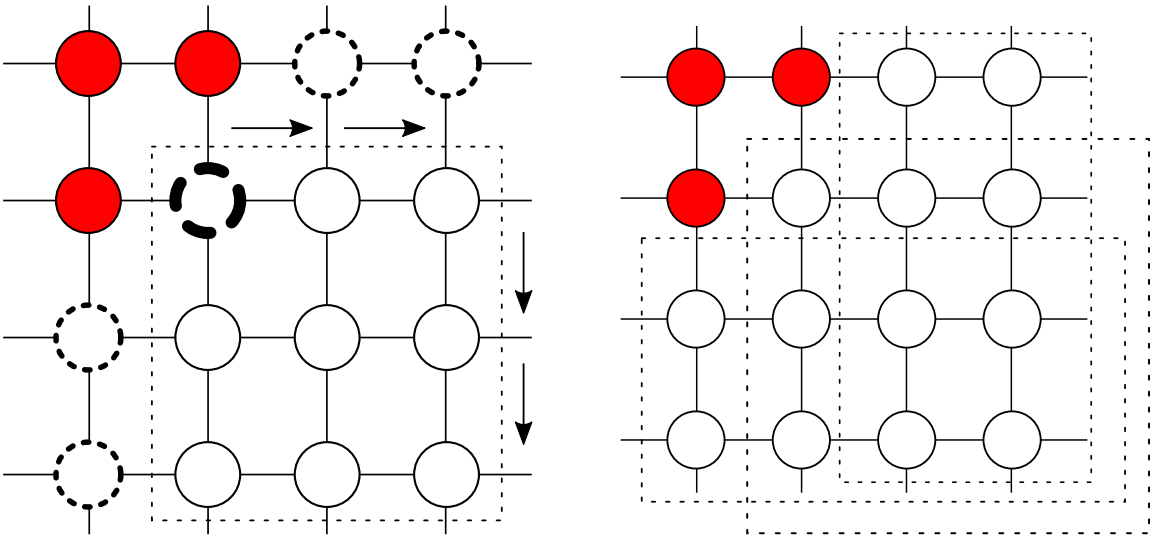
```
Input:
S -- массив, характеризующий состояние сети
       S[u] для узла u может принимать следующие значения:
             0 - free, 1 - locked, 2 - discovered
Output:
MSSs -- массив прямоугольников максимального размера
Find_MSS(S)
{
    Iter = 1
    dirs -- массив доступных направлений в торе,
                                  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
}
```

Рисунок 4.1 — Псевдокод алгоритма Find_MSS поиска прямоугольников максимального размера.

4.2 Общий алгоритм перебора n-мерных прямоугольников

Любой n-мерный прямоугольник можно описать двумя параметрами: размер прямоугольника $P = (p_1,...,p_n)$ и его расположение в сети. Алгоритм перебора n-мерных прямоугольников ищет всевозможные прямоугольники, которые можно вписать в рассматриваемую топологию сети. На вход алгоритму подается сеть I, её состояние S, требуемое число узлов m, максимально допустимое число транзитных узлов T.

На первом этапе работы алгоритма получается набор всевозможных n-мерных прямоугольников таких, что $m \leqslant |P| = \prod_{i=1}^{n} p_i \leqslant m + T$



а) Построение прямоугольника максимально возможного размера

б) Все прямоугольники максимально возможного размера, построенные алгоритмом

Рисунок 4.2 — Пунктиром показаны прямоугольники максимально возможного размера.

и $\forall i = \overline{1,n}\ \ 0 < p_i \leqslant d_i$. Сложность перебора таких прямоугольников $T = O(\prod_{i=1}^{n} d_i) = O(N)$, где N – число узлов сети.

На втором этапе алгоритма для каждого найденного n-мерного прямоугольника подбирается его расположение в системе. В наихудшем случае таких расположений N.

На выходе алгоритма получается набор всевозможных n-мерных прямоугольников заданного размера с определенным положением в исследуемой сети.

4.3 Базовый алгоритм выбора множества узлов перебором n-мерных прямоугольников

Ранее предложенный базовый алгоритм выбора узлов в сети I с состоянием (S_N, S_C) реализован следующим способом. На вход алгоритму подается сеть I, состояние сети, требуемое число узлов m, максимально допустимое число транзитных узлов T.

На первом этапе аналогично общему алгоритму перебору n-мерных прямоугольников строятся всевозможные разложения чисел $m, ..., m+1, ..., m+T$.

В разделе 3.1 описывается построение таблиц маршрутов, используемых в базовом алгоритме и вводятся ограничения на выделяемые множества. Таким образом, формально набор достижимых прямоугольников можно описать следующим образом:

$$m \leqslant |P| = \prod_{i=1}^{n} p_i \leqslant m + T, \tag{4.1}$$

$$\forall i = \overline{1,n}, 0 < p_i \leqslant \lceil d_i/2 \rceil \text{ или } p_i = d_i. \tag{4.2}$$

Требование к размеру множителя p_i вытекает из требования прохождения трафика внутри выделяемого множества.

На втором этапе происходит сортировка полученных систем по максимальному диаметру. На последнем этапе для каждого расположения каждого из полученных прямоугольников, отсортированных по диаметру, проверяется, что в покрытии отсутствуют недоступные ресурсы. Первое найденное решение будет являться результатом работы алгоритма. Псевдокод алгоритма представлен на рисунке 4.3.

Если полученный n-мерный прямоугольник покрывает множество вычислительных узлов размера больше, чем m, то множество M_a активных и M_t транзитных узлов, таких что $|M_a| = m$, $M_a \cup M_t = P$ выбираются случайно.

```
Input:
I -- сеть
S -- состояние сети I (состояние каналов связи и узлов сети)
m -- размер искомого множества доступных узлов в сети I
T -- допустимое число транзитных узлов

Output:
routing_system -- достижимая система

get_patterns_base(num) -- функция, возвращающая всевозможные разложения числа num
                          с введенными ограничениями

find_systems_base()
{
    patterns = []
    for i in range(m, m + T) {
        patterns += get_patterns_base(i)
    sort(patterns)
    for pattern in patterns {
        for n in S.Nodes() {
            PositiveAngle = n
            NegativeAngle = n + pattern
            if not BrokenResource(PositiveAngle, NegativeAngle, I, S):
                return [PositiveAngle, NegativeAngle]
}
```

Рисунок 4.3 — Псевдокод базового алгоритма выбора достижимого множества узлов.

Базовый алгоритм выбора узлов обладает рядом недостатков, среди которых:

1. В алгоритме ограничен перебор прямоугольников для того, чтобы возможно было применить базовый алгоритм построения таблиц маршрутов. Из-за этого происходит потеря возможных решений задачи выбора узлов. В частности, нельзя выбирать, возможно, достижимое множество узлов с отказавшими ресурсами.
2. Отсутствует возможность выделения для задачи множества узлов, по форме отличного от многомерного прямоугольника.

4.4 Улучшенный алгоритм выбора множества узлов перебором n-мерных прямоугольников

Идея данного алгоритма заключается в попытке анализа всевозможных n-мерных прямоугольников на достижимость при помощи маршрутного графа сети. На вход алгоритму подается сеть I, состояние сети (S_N, S_C), требуемое число узлов m, максимально допустимое число транзитных узлов T. На выходе алгоритма – достижимые множества вычислительных узлов, удовлетворяющих заданным в задаче условиям.

На первом этапе алгоритма происходит построение всевозможных n-мерных прямоугольников с помощью алгоритма, описанного выше. Из полученных прямоугольников отбираются те, в которых есть необходимое число активных узлов. Так как прямоугольник может покрывать число узлов больше требуемого, то необходимо выделить активные M_a и транзитные M_t узлы, такие что $|M_a| = m$ и $M_t = P \setminus M_a$. В текущей версии алгоритма они выбираются случайным образом, причем происходит несколько таких попыток (настраиваемый параметр).

Для каждого расположения каждого n-мерного прямоугольника в сети I проверяется достижимость выбранных узлов и строится таблица маршрутов. Заметим, что этапы проверки достижимости и построения таблицы маршрутов можно объединить, так как в основе этих алгоритмов лежит поиск вширь в графе (BFS). То есть для каждого n-мерного прямоугольника происходит попытка построения таблицы маршрутов, при возникновении неразрешимой ситуации (во время выполнения BFS не была достигнута вершина из множества активных узлов) множество помечается недостижимым и не рассматривается больше. В результате работы алгоритма получаем набор достижимых множеств узлов сети с числом активных узлов $|M_a| = m$.

Полную временную сложность алгоритма можно оценить как:

$$T = O\Big(C(n)|M_{a,t}||M_a|N^2\Big),$$

где $C(n)$ – константа 2.3, N – количество узлов в сети.

В результате работы алгоритма на выходе получается набор решений. При этом в этих решениях уже предложена таблица маршрутов, построенная алгоритмом, основанном на обходе маршрутного графа при помощи поиска вширь. При необходимости можно перестроить таблицы маршрутов с использованием

генетического алгоритма. Псевдокод улучшенного алгоритма поиска узлов методом перебора n-мерных прямоугольников представлен на рисунке 4.4.

```
Input:
I -- сеть
S -- состояние сети I (состояние каналов связи и узлов сети)
m -- размер искомого множества доступных узлов в сети I
T -- допустимое число транзитных узлов

Output:
routing_systems -- набор достижимых систем

find_systems_parse()
{
    routing_systems = []
    for i in range(m, m + T) {
        patterns = get_patterns(i)
        for n in S.Nodes() {
            for pattern in patterns {
                PositiveAngle = n
                NegativeAngle = n + pattern
                if AvailableNodes(PositiveAngle, NegativeAngle) < m {
                    continue
                }
                ActiveNodes, TransitNodes = SelectActiveNodes(PositiveAngle,
                                                    NegativeAngle, m, I, S)

                RT = construct_RT_bfs(ActiveNodes, TransitNodes, I, S)
                if RT {
                    routing_systems += [(PositiveAngle, NegativeAngle,
                                                        ActiveNodes, RT)]
                }
            }
        }
    }
    return routing_systems
}
```

Рисунок 4.4 — Псевдокод улучшенного алгоритма выбора узлов перебором n-мерных прямоугольников.

Стоит заметить, что при такой организации алгоритма возможно нахождение в качестве решений те множества, которые не являются прямоугольниками.

Такая ситуация возникает, если в покрытии присутствуют сломанные или отказавшие узлы сети, но при этом доступно требуемое число активных узлов, и они достижимы. Пример такой ситуации представлен на рисунке 4.5, на котором показан поиск 4 узлов в сети с топологией решетка 3×3 и 4-мя отказавшими или занятыми узлами и допустимым числом транзитных узлов 2. На рисунке пунктирной линией обозначены недоступные ресурсы. Большим прямоугольником выделено множество узлов, подходящих для решения. Хотя они и покрываются прямоугольником 2×3, в результате будут выбраны 4 узла, не являющиеся прямоугольником.

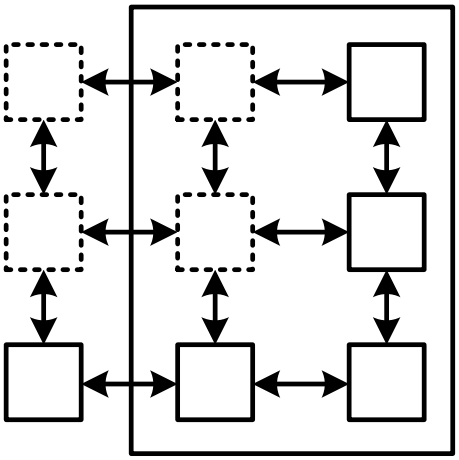


Рисунок 4.5 — Пример поиска 4 узлов в сети с топологией решетка 3×3, 4 недоступными узлами и допустимым числом транзитных узлов 2.

Таким образом, данный алгоритм снимает недостатки базового алгоритма, которые становятся его достоинствами:

1. В алгоритме снимается ограничение на перебор прямоугольников для того, чтобы было возможно применить базовый алгоритм построения таблиц маршрутов. Из-за этого появляются новые возможные решения задачи.
2. Присутствует возможность выделения для задания множества узлов, по форме отличного от многомерного прямоугольника.

Однако данный алгоритм имеет потенциальный недостаток, связанный со слишком большим объемом перебора, данный недостаток может быть нивелирован, если в какой-то момент перебор остановить.

4.5 Алгоритм выбора множества узлов равномерным расширением

В качестве второго варианта решения задачи выбора узлов предлагается приближенный алгоритм выбора множества узлов равномерным расширением. Идея алгоритма заключается в том, что из каждого узла тора происходит попытка построить n-мерный прямоугольник поочередным (равномерным) расширением в разные стороны. Ниже алгоритм будет называться алгоритмом равномерного расширения.

На вход алгоритму подается размер искомого множества m, сеть I, состояние сети (S_N, S_C), максимально допустимое число транзитных узлов T. На выходе алгоритма получается набор достижимых множеств узлов сети требуемого размера.

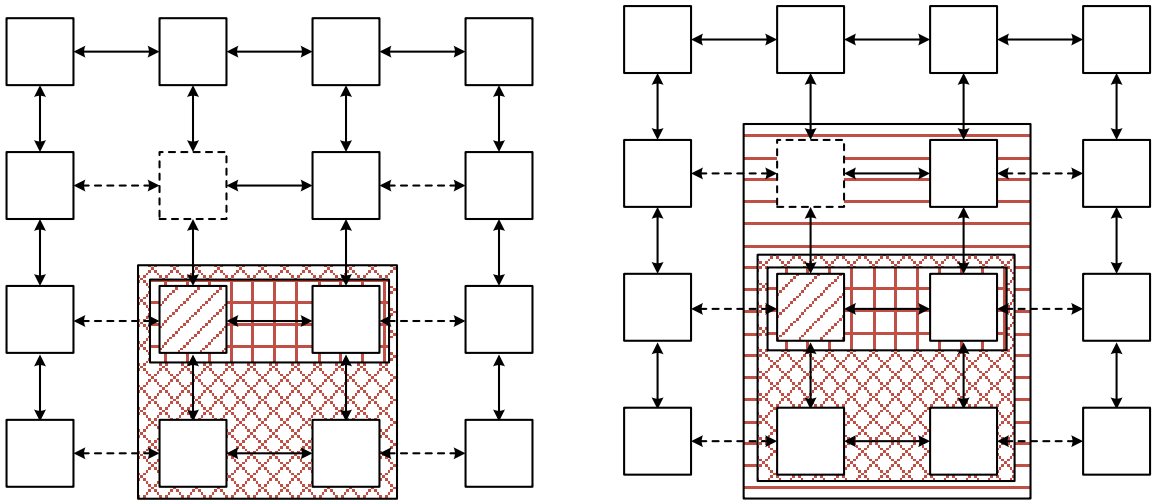
Алгоритм состоит из трех этапов. На первом из каждого узла сети производится расширение во все стороны с покрытием только доступных ресурсов. На втором этапе проводится сортировка полученных систем и удаление одинаковых. На третьем этапе производится расширение получившихся систем с добавлением областей с недоступным ресурсами. Псевдокод алгоритма поиска узлов методом равномерного расширения представлен на рисунке 4.6.

Рассмотрим алгоритм подробнее. На первом этапе алгоритм выделяет прямоугольники без недоступных ресурсов. Такие прямоугольники хороши тем, что не требуют проверки множества узлов сети на достижимость. Для каждого узла сети u^i строится многомерный прямоугольник следующим образом. Каждый прямоугольник можно задать двумя узлами в противоположных углах: Pa и Na. Первоначальный прямоугольник состоит из одной вершины: $Pa = Na = u^i$. Затем происходят попытки увеличения прямоугольника в каждом направлении тора по очереди с проверкой, что в захваченной области нет недоступных ресурсов. Расширение прямоугольника происходит путем перемещения одного из его углов в выбранном направлении: Pa в случае расширения в положительном направлении, Na – в случае отрицательного. После каждого расширения происходит проверка числа захваченных узлов, если это число больше m, то первый этап завершается. Также первый этап завершается, если отсутствуют направления, в которые можно расширить прямоугольник. Пример работы алгоритма представлен на рисунке 4.7. В результате работы алгоритма будет найдено достижимое множество, состоящее из 5 активных узлов.

```
Input:
I -- сеть
S -- состояние сети I (состояние каналов связи и узлов сети)
m -- размер искомого множества доступных узлов в сети I
T -- допустимое число транзитных узлов

Output:
routing_systems -- набор достижимых систем

ExtendNonBroken(I, S) -- функция поочередного расширения в разные стороны.
                      Покрываются только доступные ресурсы. Расширение
                      происходит до тех пор, пока это возможно или пока
                      не покроется необходимое число узлов.
ExtendBroken(I, S)    -- функция поочередного расширения в разные стороны.
                      На каждом шаге проверяется достижимость добавленного множества.
                      Расширение происходит до тех пор, пока не покроется
                      необходимое число узлов.

find_systems_parse()
{
    routing_systems = []
    for system in S.AvailableNodes {
        system.ExtendNonBroken(I, S)
        routing_systems += system
    }
    delete_duplicates(routing_systems)
    for system in routing_systems {
        if system.size() < m {
            system.ExtendBroken(I, S)
        }
        if system.size < m {
            routing_systems.delete(system)
        }
    }
    return routing_systems
}
```

Рисунок 4.6 — Псевдокод алгоритма выбора узлов равномерным расширением.

Оценим сложность первого этапа алгоритма. За все время обхода нужно проверить $O(2n|M_{a,t}| + |M_{a,t}|) = O((2n+1)|M_{a,t}|)$ узлов и каналов связи. Этот алгоритм нужно применить ко всем узлам сети. Получаем следующую



а) Первый этап б) Третий этап

Рисунок 4.7 — Пример работы алгоритма равномерного расширения в сети с топологией двухмерная решетка с отказами. Пунктиром -- отмечены недоступные каналы связи и узлы, ▨ – стартовый узел алгоритма, # – первый шаг первого этапа, ❖ – второй шаг первого этапа, ☰ – шаг третьего этапа.

оценку сложности: $T_{s_1} = O((2n+1)|M_{a,t}|N)$, где N – количество узлов сети, n – размерность сети.

Для того, чтобы сократить дальнейшую работу, на втором этапе происходит удаление одинаковых построенных прямоугольников при помощи предварительной сортировки. Если на втором этапе находятся прямоугольники нужного размера, алгоритм заканчивается. Так как каждый прямоугольник задается с помощью двух узлов сети, то его можно описать набором $2n$ чисел. Чтобы сравнить все прямоугольники, необходимо $O(Nlog_2N)$ сравнений. Таким образом, оценка сложности второго этапа $T_{s_2} = O(Nlog_2N)$.

На третьем этапе происходят попытки расширить полученные прямоугольники в стороны с недоступными ресурсами сети, на каждом шаге проверяется достижимость результирующего множества узлов. При расширении получается новое множество узлов сети $M'' = M \cup M'$, где M – достижимое множество узлов, полученное на предыдущем шаге алгоритма; M' – добавленное множество узлов сети на текущем шаге алгоритма (состоящее только из доступных узлов), полученное путем расширения множества узлов M в выбранном направлении. Чтобы проверить достижимость множества M'', необходимо проверить наличие маршрутов между каждым узлом множества M' и каждым узлом множества M''. Для этого из каждой соответствующей вершины множества M' можно пройти поиском вширь по маршрутному графу $RG(V, E)$

и графу $RG^T(V,E)$, полученному из графа $RG(E,V)$ путем обращения связей (стартовая вершина теперь будет U^j_{end}, а конечная – U^i_{begin}). Таким образом, для каждой вершины u^i из M' можно получить множество узлов u^j, для которых существует маршрут в одну сторону и обратно. В случае, если будут достижимы все соответствующие вершины множества M'' из всех соответствующих вершин множества M', то множество M'' является достижимым.

В наихудшем случае во всех расширениях прямоугольника присутствовали сломанные каналы связи, а значит, для всех узлов множества $M_{a,t}$ пришлось выполнить поиск вширь по маршрутному графу $RG(V,E)$ и графу $RG^T(V,E)$, поэтому сложность третьего этапа $T_{s_3} = O(2C(n)|M_{a,t}||M_a|N)$.

Полная временная сложность алгоритма равномерного расширения:

$$\begin{aligned} T_{expand} &= T_{s_1} + T_{s_2} + T_{s_3} \\ &= O\Big((2n+1)|M_{a,t}|N + Nlog_2N + 2C(n)|M_{a,t}||M_a|N\Big) \\ &= O\Big(2C(n)|M_{a,t}||M_a|N)\Big). \end{aligned}$$

Значение константы $C(n)$ 2.3 довольно велико, для сети с размерностью $n = 4$ значение $C(n) = 769$. Однако $C(n)$ не меняется с увеличением множества узлов сети, поэтому предложенный алгоритм является полиномиальным.

В результате работы алгоритма получается набор достижимых множеств размера больше или равного m. Аналогично предыдущему алгоритму выбираются активные и транзитные узлы, и строится таблица маршрутов генетическим алгоритмом или алгоритмом с использованием маршрутного графа.

4.6 Выводы

В данной главе сформулирована задача выбора в сети I с состоянием (S_N, S_C) достижимого множества вычислительных узлов требуемого размера m с допустимым числом транзитных узлов T.

Рассмотрен разработанный ранее базовый алгоритм выбора множества узлов перебором n-мерных прямоугольников. Данный алгоритм использует базовый алгоритм построения таблиц маршрутов, в результате при выборе узлов имеются недостатки, связанные с потерей возможных решений задачи: нельзя выбирать достижимое множество узлов с недоступными ресурсами сети, отсутствует возможность выделения для задания множества узлов, по форме отличного от многомерного прямоугольника.

В данной главе предложен улучшенный алгоритм выбора множества узлов перебором n-мерных прямоугольников. Данный алгоритм основан на построении таблиц маршрутов при помощи маршрутного графа, в результате нивелируются перечисленные для базового алгоритма недостатки. Показано, что временная сложность алгоритма оценивается как $O(N^4)$, где N – количество вычислительных узлов в сети.

Также предложен алгоритм выбора множества узлов равномерным расширением. Идея алгоритма заключается в том, что из каждого узла тора происходит попытка построить n-мерный прямоугольник поочередным (равномерным) расширением в разные стороны. Алгоритм равномерного расширения имеет временную сложность $O(N^3)$.

В качестве критерия оценки выбора одного решения из набора решений используется упорядоченный набор критериев минимизации числа транзитных узлов, минимизации фрагментации сети, минимизации диаметра и максимальной загруженности π_{max} построенной таблицы маршрутов.

Глава 5. Исследование разработанных алгоритмов и утилизации вычислительных систем

Данная глава посвящена исследованию разработанных алгоритмов. В разделе 5.1 приведено исследование отказоустойчивости сети на основе разработанного алгоритма определения достижимости множества вычислительных узлов. Раздел 5.2 посвящен исследованию алгоритмов построения сбалансированной таблицы маршрутов. В разделе 5.3 описана разработанная имитационная модель вычислительной системы с заданной сетью, с помощью которой в разделе 5.4 исследованы алгоритмы выбора узлов, а в разделе 5.5 исследована эффективность работы суперкомпьютера с использованием разработанных алгоритмов.

5.1 Исследование отказоустойчивости вычислительной системы

При некотором количестве недоступных каналов связи теряется достижимость узлов сети, а значит построение таблицы маршрутов невозможно. Разработанный алгоритм определения достижимости узлов сети позволяет провести исследование отказоустойчивости сети, то есть выяснить, при скольких недоступных каналах связи все еще сохраняется достижимость узлов сети.

В данном разделе исследуется маршрутизация R_{A^-} без нарушения правила порядка направлений для первого и последнего нестандартного шага. В подразделе 5.1.1 исследована маршрутизация R_{A^*} с возможностью нарушения правила порядка направлений для первого и последнего нестандартного шага.

Исследование отказоустойчивости коммуникационной сети с маршрутизацией R_{A^-} проводилось на всевозможных сетях с числом узлов до 1024. Длина измерения ограничена: $2 \leqslant d_i \leqslant 8$. В конфигурации с длиной измерения равной 2 сеть объединялась в решетку по данному измерению, во всех остальных – в кольцо. Исследовались 2-х, 3-х и 4-х мерные топологии.

Для каждой топологии сети случайно выбирались недоступные каналы связи. Для сети с числом узлов до 769 число недоступных каналов связи увеличивалось с шагом 2, для остальных – с шагом 10. Для каждого числа недоступных каналов связи проводилось 10 равномерно случайных попыток выбора недоступных каналов и проверки достижимости сети. Проверялась достижимость всех вычислительных узлов сети.

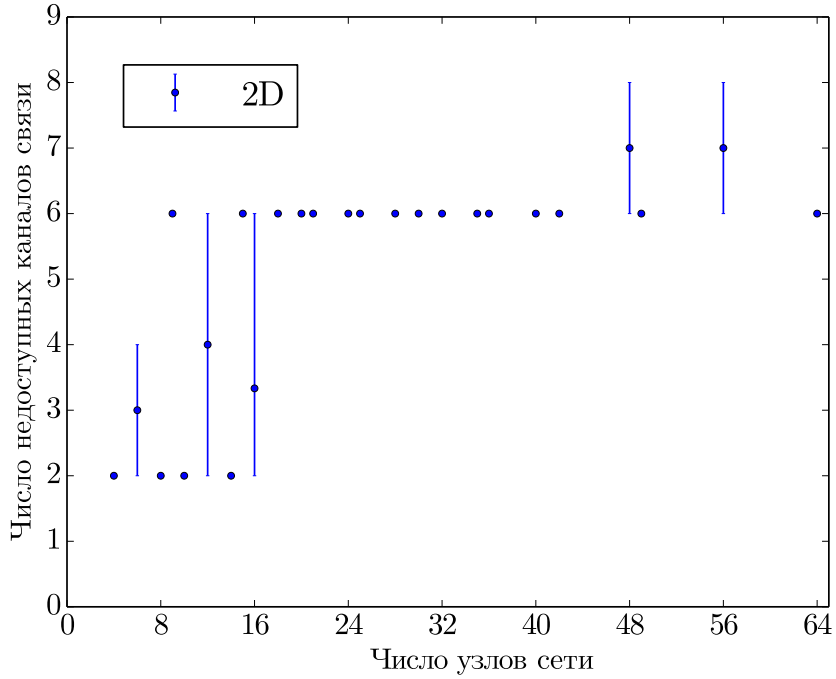


Рисунок 5.1 — Исследование потери достижимости для 2-х мерных топологий.

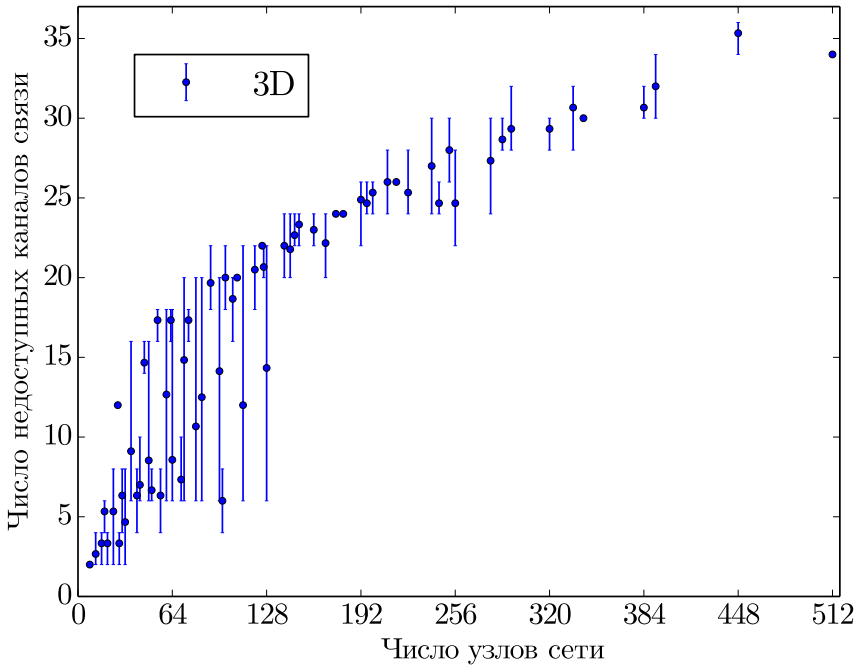


Рисунок 5.2 — Исследование потери достижимости для 3-х мерных топологий.

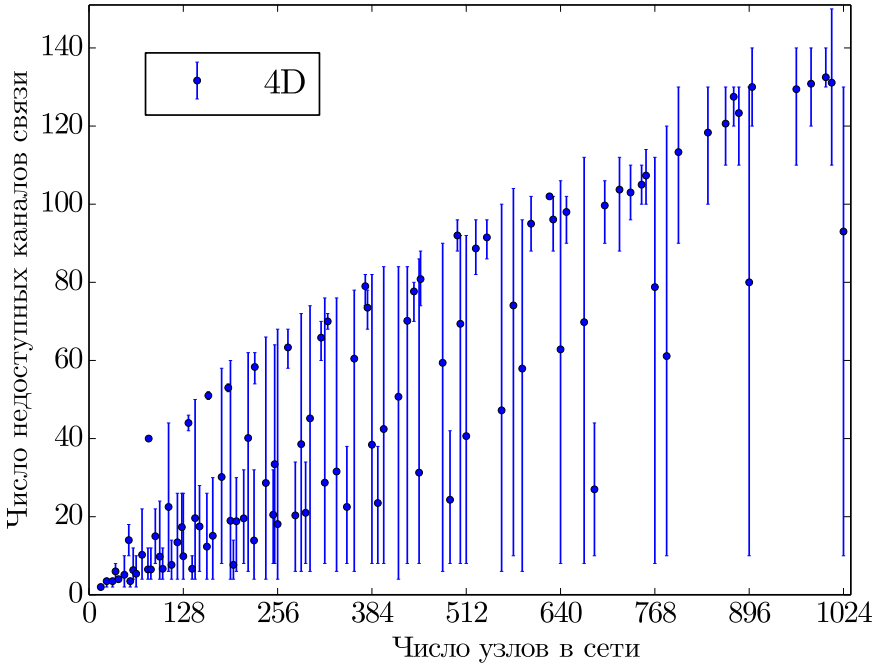


Рисунок 5.3 — Исследование потери достижимости для 4-х мерных топологий.

На рисунках 5.1, 5.2 и 5.3 представлены результаты проведенного исследования. По оси X обозначен размер исследуемой сети. По оси Y – число недоступных каналов связи, при котором за 10 итераций не удалось получить достижимое множество узлов сети. Так как различные топологии могут иметь одинаковое число узлов, то по оси Y представлено минимальное, максимальное и среднее значение (точка) по таким топологиям. Скачки среднего на графиках объясняются тем, что топологии сетей с короткими длинами измерений менее отказоустойчивые. Особенно это касается топологий с длиной измерения 2, в которой узлы соединяются в решетку.

5.1.1 Исследование отказоустойчивости сети с маршрутизацией, нарушающей правило порядка направлений для First Step/Last Step

В данном разделе исследуется маршрутизация R_{A^*} с возможностью нарушения правила порядка направлений для первого и последнего нестандартного шага в сравнении с маршрутизацией R_{A^-}, в которой нет возможности нарушения правила порядка направлений.

Исследование проводилось на всевозможных сетях с числом узлов до 128. Длина измерения ограничена: $2 \leqslant d_i \leqslant 8$. В конфигурации с длиной изме-

рения равной 2 сеть объединялась в решетку по данному измерению, во всех остальных – в кольцо. Исследовались 2-х, 3-х и 4-х мерные топологии. Число узлов 128 превышает количество узлов в самом большом суперкомпьютере на основе сети Ангара, поэтому указанные сети охватывают необходимые практические случаи.

Для каждой топологии сети случайно выбирались недоступные каналы связи. Число недоступных каналов связи выбиралось с шагом 1. Для каждого числа недоступных каналов связи происходило 100 попыток получения достижимого множества узлов сети при использовании различных вариантов маршрутизации. При невозможности получить достижимое множество данное число каналов связи фиксировалось для каждого из вариантов маршрутизации.

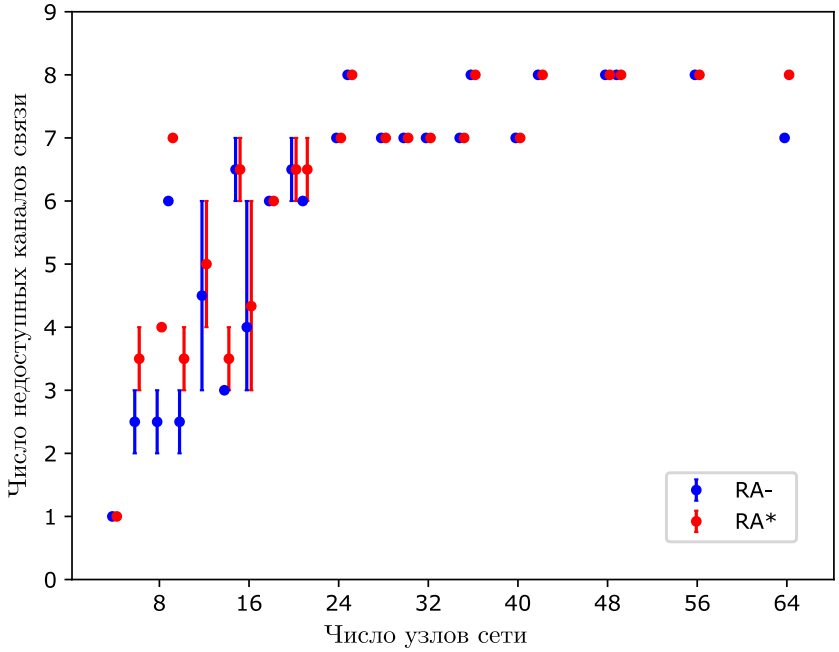


Рисунок 5.4 — Сравнение потери достижимости при использовании маршрутизаций R_{A^*} с возможностью нарушения и R_{A^-} без нарушения правила порядка направлений для 2-х мерных топологий.

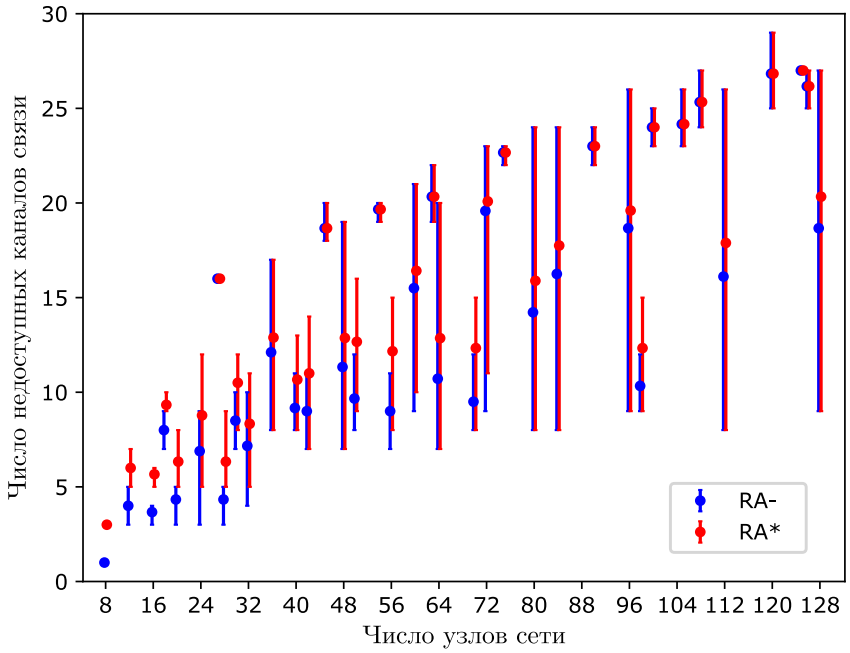


Рисунок 5.5 — Сравнение потери достижимости при использовании маршрутизаций R_{A^*} с возможностью нарушения и R_{A^-} без нарушения правила порядка направлений для 3-х мерных топологий.

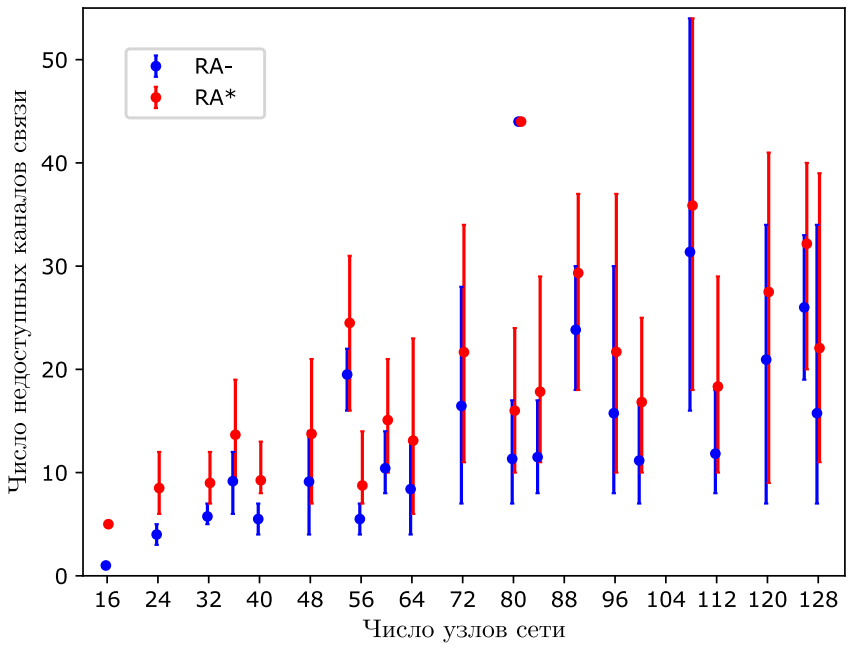


Рисунок 5.6 — Сравнение потери достижимости при использовании маршрутизаций R_{A^*} с возможностью нарушения и R_{A^-} без нарушения правила порядка направлений для 4-х мерных топологий.

На рисунках 5.4, 5.5 и 5.6 представлены результаты проведенного исследования. По оси X обозначен размер исследуемой сети. По оси Y – число каналов связи, при котором за 100 итераций не удалось получить достижимое множество узлов.

В среднем возможность нарушения правила порядка направлений в маршрутизации R_{A^*} по сравнению с маршрутизацией R_{A^-} без возможности нарушения данного правила позволяет увеличить число случайно сломанных каналов связи до потери достижимости на 4,9%, 8,2% и 34% для сетей с топологией 2D, 3D и 4D-тор, соответственно.

5.2 Исследование алгоритмов построения таблиц маршрутов

В главе 3 при разработке алгоритма построения таблиц маршрутов сформулированы задачи исследования:

– сравнить алгоритм по критерию оценки таблиц маршрутов по сравнению с базовым алгоритмом для сетей без отказов;
– оценить пригодность алгоритма с точки зрения применения на практике по времени выполнения алгоритма.

В данном разделе представлены результаты исследований по поставленным задачам.

5.2.1 Сравнение по критерию оценки построенных таблиц маршрутов

Исследование алгоритмов построения таблиц маршрутов заключается в рассмотрении качества построенных при помощи различных алгоритмов балансировки трафика таблиц маршрутов.

В качестве критерия оценки таблицы маршрутов выбрана максимальная загруженность канала связи π_{max}. Данный критерий чаще всего используется в литературе для оценки сбалансированности таблицы маршрутов.

На **первом этапе** исследования качества рассматривались сети без недоступных ресурсов (отказов). В этом случае возможно сравнение разработанных алгоритмов с базовым алгоритмом построения таблиц маршрутов, который изначально существовал в системном программном обеспечении сети Ангара для

поддержки работы вычислительной системы при условии отсутствия отказов. Данный алгоритм описан разделе 3.1.

Рассматривались всевозможные сети с максимальной размерностью $\max\limits_{i\in\overline{1,n}} d_i \leqslant 8$, количество измерений тора $2 \leqslant n \leqslant 4$, число узлов $\prod\limits_{i\in\overline{1,n}} d_i \leqslant 128$. В сетях с длиной измерения равной 2 сеть объединялась в решетку по данному измерению, во всех остальных – в кольцо. Число узлов 128 превышает количество узлов в самом большом суперкомпьютере на основе сети Ангара, поэтому указанные сети охватывают необходимые практические случаи. Таблицы маршрутов строились для всех узлов каждой сети, все узлы считаются активными.

Таблица 2 — Среднее, минимальное и максимальное значение коэффициента сбалансированности *balance factor* по всем рассматриваемым сетям указанной размерности без недоступных каналов связи.

	средний *balance factor*, % (минимальный – максимальный, %)		
	2D	3D	4D
Базовый	33% (0% – 80%)	72% (0% – 200%)	145% (0% – 382%)
Генетический	26% (0% – 74%)	59% (0% – 184%)	117% (0% – 350%)
BFS	46% (0% – 110%)	83% (0% – 212%)	159% (0% – 364%)

Определение 5.1. *Коэффициент сбалансированности таблицы маршрутов RT достижимого множества* $balance\ factor = (\frac{\pi_{max}(RT)}{\pi_{perfect}} - 1) * 100\%$.

Для каждой рассматриваемой сети и построенной таблицы маршрутизации, рассматривался коэффициент сбалансированности, который характеризует, насколько для данной сети полученная максимальная загруженность таблицы маршрутов хуже, чем идеальная загруженность. Иными словами, коэффициент сбалансированности показывает отклонение приближенного решения задачи балансировки сетевого трафика от оптимального решения.

В таблице 2 для каждого алгоритма построения таблиц маршрутов и для каждой размерности тора приведены среднее, минимальное и максимальное значения коэффициента сбалансированности по всем рассматриваемым сетям.

Необходимо отметить, что каждый из алгоритмов для каждой размерности на некоторых сетях позволяет получить оптимальное решение, поэтому минимальное значение коэффициента сбалансированности для всех случаев равно нулю. Для двухмерных топологий алгоритм BFS на основе поиска

вширь в среднем получает средний коэффициент сбалансированности 46%, что означает, что максимальная загруженность в среднем на 46% больше, чем оптимальное решение, но не более, чем на 110%. Коэффициент сбалансированности для BFS на 13% хуже, чем для базового алгоритма и на 20%, чем для генетического. Генетический алгоритм в данном эксперименте показывает наилучшие результаты. Следующим по качеству идет базовый алгоритм, но он дает меньшую свободу при выборе достижимого множества узлов и не применим для сетей с отказами.

Максимальный (средний) проигрыш по характеристике коэффициента сбалансированности по всем сетям для алгоритма BFS базовому алгоритму составляет 30% (13%), а по сравнению с генетическим алгоритмом – 36% (32%).

На **втором этапе** исследования рассмотрены сети с недоступными каналами связи. На этом этапе рассмотрены всевозможные сети с максимальной размерностью тора $\max\limits_{i\in\overline{1,n}} d_i \leqslant 8$, количество измерений тора $2 \leqslant n \leqslant 4$, числом узлов $\prod\limits_{i\in\overline{1,n}} d_i \leqslant 128$. В сетях с длиной измерения равной 2 сеть объединялась в решетку по данному измерению, во всех остальных – в кольцо.

Для каждой топологии сети (множество узлов – N, все узлы активные, транзитных узлов нет) случайно выбирались недоступные каналы связи. Количество недоступных каналов связи выбиралось от 0 до некоторого F_{min}. Так как нельзя сказать точно, при каком минимальном числе недоступных каналов связи F_{min} при любом их выборе множество N будет недостижимым, то это число подбиралось эмпирически. Для этого для исследуемого числа недоступных каналов связи i проводилось до 10 попыток случайного выбора недоступных каналов связи. Если множество N достижимо с учетом выбранных недоступных каналов, то происходит расчет характеристик и переход к следующему количеству недоступных каналов связи. Если для всех 10 попыток множество узлов N оставалось недостижимым, то $F_{min} = i - 1$.

Таблица 3 — Среднее, минимальное и максимальное значение коэффициента сбалансированности *balance factor* по всем рассматриваемым сетям указанной размерности и всем рассматриваемым вариантам выбора недоступных каналов связи.

	средний *balance factor*, % (минимальный – максимальный, %)		
	2D	3D	4D
BFS	84% (0% – 236%)	126% (0% – 340%)	196% (0% – 627%)
Генетический	66% (0% – 222%)	84% (0% – 287%)	134% (0% – 529%)

Также как и для предыдущего исследования, для каждого алгоритма балансировки трафика производился расчет коэффициента сбалансированности. В таблице 3 для каждого алгоритма построения таблиц маршрутов и для каждой размерности тора приведены среднее, минимальное и максимальное значения коэффициента сбалансированности по всем рассматриваемым сетям и вариантам выбора недоступных каналов связи. Алгоритм на основе BFS показывает средний коэффициент сбалансированности на 18%, 42% и 58% больше, чем генетический алгоритм на 2-х, 3-х и 4-х мерных топологиях соответственно. Исследования на реальных вычислительных системах показывают, что данное ухудшение коэффициента сбалансированности незначительно влияют на производительность прикладных приложений, поэтому алгоритм BFS применим на практике. Данные исследования на реальных вычислительных системах не входили в задачи диссертационной работы и поэтому в тексте не представлены.

5.2.2 Сравнение по времени работы

Кроме качества получаемых таблиц маршрутов с практической точки зрения имеет значение время работы алгоритмов. Время работы алгоритма построения таблицы маршрутов не должно быть очень большим. На практике в системе управления заданиями Slurm на вычислительном кластере по умолчанию на работу плагина по выделению узлов отведено не более 10 секунд, поэтому это время взято за ограничение сверху на построение таблицы маршрутов и выбор узлов.

Исследование времени работы алгоритма на основе BFS проведено на различных сетях с максимальной размерностью тора $\max\limits_{i\in\overline{1,n}} d_i \leqslant 8$, количество измерений тора $2 \leqslant n \leqslant 4$, числом узлов сети $\prod\limits_{i\in\overline{1,n}} d_i \leqslant 1024$. В конфигурации с длиной измерения равной 2 сеть объединялась в решетку по данному измерению, во всех остальных – в кольцо.

Для каждой топологии сети случайно выбирались недоступные каналы связи. Для сетей с числом узлов до 769 число недоступных каналов связи увеличивалось с шагом 2, для остальных с шагом 10. Для каждого числа недоступных каналов связи проводилось 10 экспериментов, в каждом из которых сломанные каналы выбирались случайно. Проверялась достижимость всех узлов сети.

Для каждой сети и для каждого числа недоступных каналов связи с достижимым множеством узлов сети строились таблицы маршрутов с помощью алгоритма на основе поиска вширь в графе. На рисунках 5.7а, 5.7б и 5.7в представлено среднее время работы алгоритма построения таблицы маршрутов на основе поиска вширь в графе для 2-х, 3-х и 4-х мерных топологий.

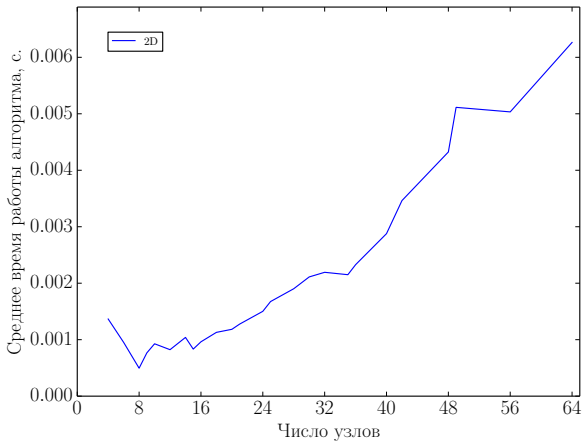
Исследование времени работы генетического алгоритма построения таблиц маршрутов проведено на различных сетях с максимальной размерностью тора $\max\limits_{i\in\overline{1,n}} d_i \leqslant 8$, количество измерений тора $2 \leqslant n \leqslant 4$ и числом узлов сети $\prod\limits_{i\in\overline{1,n}} d_i \leqslant 128$. В сетях с длиной измерения равной 2 сеть объединялась в решетку по данному измерению, во всех остальных – в кольцо. Исследовались сети с отказами, по аналогии с предыдущим разделом.

Для каждой топологии и для каждого числа сломанных каналов связи с достижимым множеством узлов сети строились таблицы маршрутов с помощью генетического алгоритма. На рисунке 5.7г представлено среднее время работы генетического алгоритма построения таблицы маршрутов для 2-х, 3-х и 4-х мерных топологий.

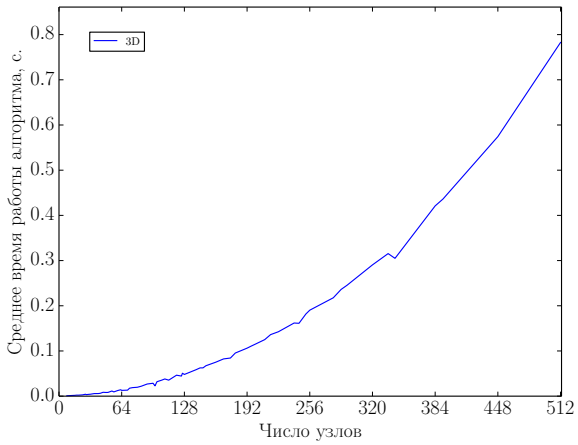
Время работы генетического алгоритма построения таблицы маршрутов для сетей с количеством узлов больше 48 при эксплуатации вычислительного кластера становится недопустимо большим. Время работы алгоритма на основе поиска вширь в графе на сетях с числом узлов до 1024 показывает допустимые результаты.

Выводы

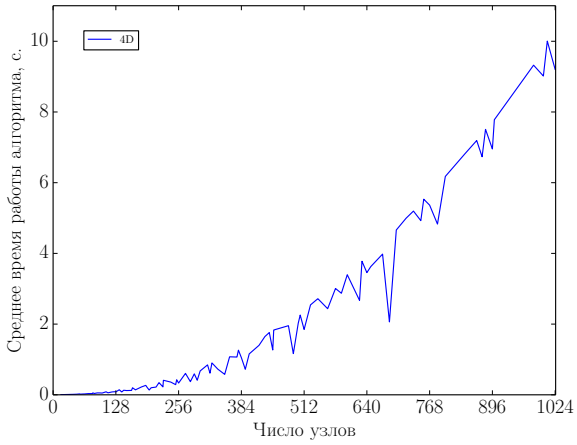
Учитывая большие временные затраты на генетический алгоритм построения таблиц маршрутов, а также приемлемое качество таблиц маршрутов, которое позволяет получить алгоритм на основе поиска вширь в графе, в дальнейших исследованиях будет использоваться алгоритм построения таблиц маршрутов на основе поиска вширь в графе.



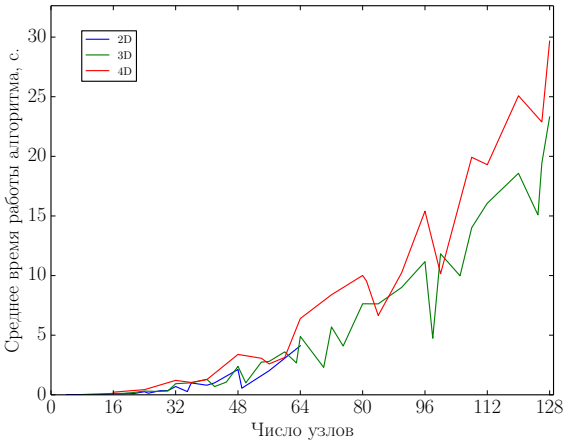
а) На основе поиска вширь для 2-х мерных топологий



б) На основе поиска вширь для 3-х мерных топологий



в) На основе поиска вширь для 4-х мерных топологий



г) Генетический алгоритм

Рисунок 5.7 — Среднее время работы алгоритмов построения таблицы маршрутов.

5.3 Имитационная модель вычислительной системы

Для оценки утилизации ресурсов вычислительной системы (кластера) с заданной сетью в диссертационной работе разработана имитационная модель вычислительной системы. На вход имитационной модели подается поток пользовательских заданий $Q = W^1,...,W^k$. На выходе выдается полное время работы всего кластера T, время работы каждого узла T_i и время начала предоставления ресурсов для каждого задания. Используя эти данные, можно оценить эффективность работы вычислительного кластера.

Введем некоторые понятия, необходимые для описания работы имитационной модели. *Очередью заданий* Q_{now} назовем набор заданий из потока, для которых не выделялись ресурсы и время их появления в потоке W^i_{start} меньше текущего имитируемого 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 набора Q_{now}. *Временем занятости узла* u сети I назовем время, на которое узел u был выделен для некоторого задания W. В начальный момент времени $t = 0$ время занятости каждого из узлов сети 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 Q в очередь заданий Q_{now}. Тогда *временем ожидания модели* T_{sleep} назовем минимальное время до изменения состояния модели: $T_{sleep} = min(T_S, T_{queue})$.

Алгоритм работы имитационной модели устроен следующим образом. Если окно заданий Q_{window} не пусто, модель выполняет процедуру поиска достижимого множества узлов для каждого задания из данного окна по очереди. Если удалось найти решение, то модель выделяет найденные ресурсы на необходимое время, а также удаляет это задание из очереди. Если решение не было найдено, то модель выполняет процедуру поиска для следующего задания из окна Q_{window}. Если ни одно решение ни для одного задания из окна не было найдено, время имитационной модели сдвигается на время ожидания T_{sleep}, а время занятости каждого занятого узла u сети I уменьшается на T_{sleep}. Если очередь и поток заданий пусты, а все узлы перешли в состояние свободных,

то модель завершает свою работу. Схема работы имитационной модели представлена на рисунке 5.8.

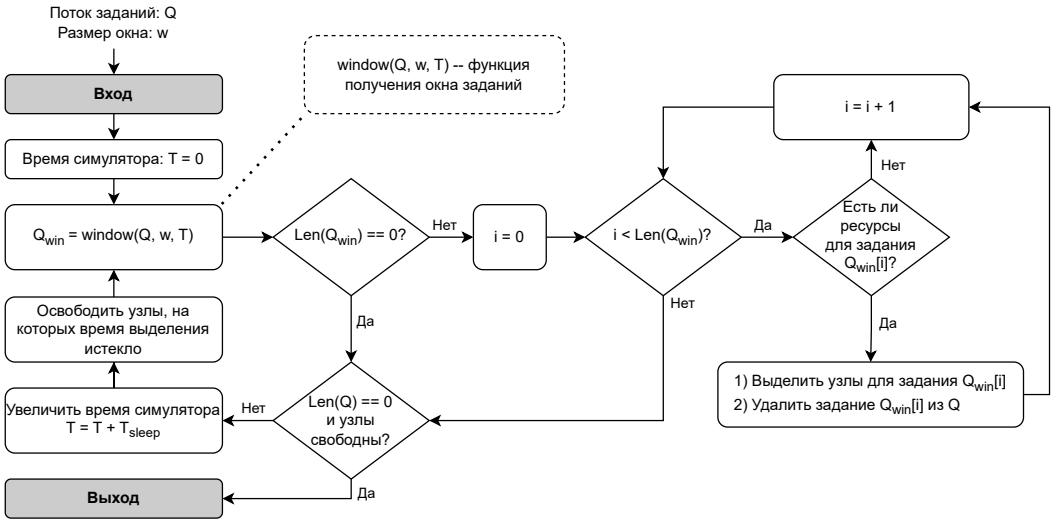
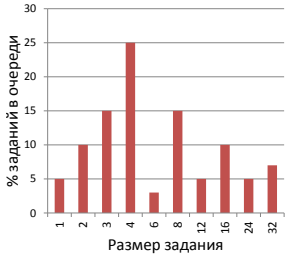


Рисунок 5.8 — Схема работы имитационной модели вычислительного кластера.

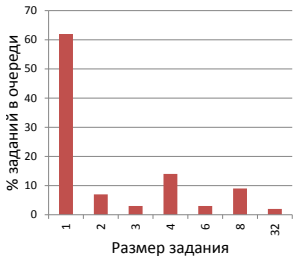
5.3.1 Очередь заданий пользователей для модели

Каждое задание $W^i = (W^i_{start}, W^i_N, W^i_t)$ из очереди характеризуется временем появления задания в потоке W^i_{start}, продолжительностью W^i_t и количеством запрашиваемых ресурсов (размером задания) W^i_N. Для исследования утилизации вычислительной системы разработан алгоритм построения потока (очереди) заданий пользователей в зависимости от размера вычислительных систем (в узлах сети). Примечание: в дальнейшем в тексте в качестве понятия поток заданий будет часто использоваться понятие очередь заданий.

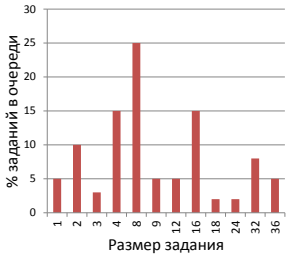
Распределение долей размеров заданий в очереди для разных систем размером от 32 до 144 узлов показано на одном из рисунков 5.9 и 5.10.



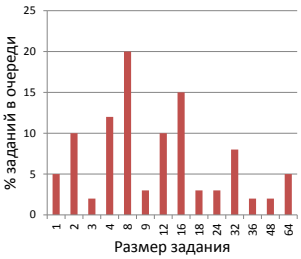
а) Для систем из 32 узлов.



б) Для систем из 32 узлов, суперкомпьютер «Десмос».

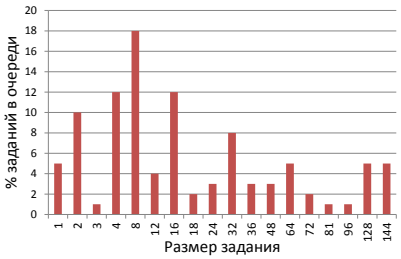


в) Для систем из 36 узлов.



г) Для систем из 64 узлов.

д) Для систем из 96 узлов.



е) Для систем из 144 узлов.

Рисунок 5.9 — Распределения долей размеров заданий для систем с размерами 32 – 144 узлов.

Распределение долей размеров заданий, представленное на рисунке 5.9б, соответствует реальному распределению пользовательских заданий, полученному на гибридном суперкомпьютере «Десмос» на базе сети Ангара. Суперкомпьютер «Десмос», установленный в ОИВТ РАН, имеет топологию сети 4D тор 4x2x2x2 [95]. В дальнейшем системы с данной очередью будем помечать символом s – 4x2x2x2s и 4x4x2s. Распределения, представленные на рисунках 5.9а и 5.9в–5.9е – синтетические, основанные на предположении о том, что чаще всего встречаются задания с требуемым числом узлов, равным степеням двойки. Вероятности для других размеров узлов равны 0.

Для описания распределений систем с большим числом узлов (от 256 до 1024) выделено 3 следующих типа размеров заданий.

Тип 1. Большие задания с числом узлов, равным степени двойки и не меньше заданного порогового значения.

Тип 2. Малые задания с числом узлов, равным степени двойки.

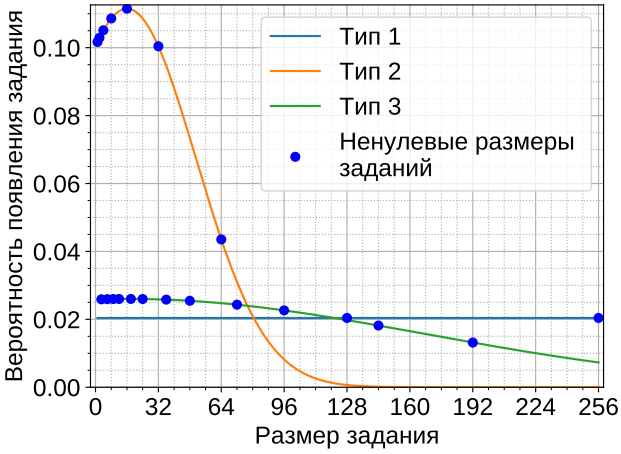
Тип 3. Задания с числом узлов, кратным степени двойки и числам 3 или 9.

Заданиям Типа 1 выставляется равная вероятность и задается их общая вероятность. Для заданий Типов 2 и 3 вероятности задаются плотностью вероятности для нормального распределения (по функции Гаусса $g(x) = ae^{-\frac{(x-b)^2}{2c^2}}, a = \frac{1}{\sigma\sqrt{2\pi}}, b = \mu, c = \sigma$) с заданными параметрами среднеквадратичного отклонения σ и математического ожидания μ. Функция распределения вероятностей размеров заданий задана дискретно. Для размеров заданий, не подходящих ни под один тип, вероятность выставляется равной 0. Для возможных размеров заданий сумма их вероятностей равна 1.

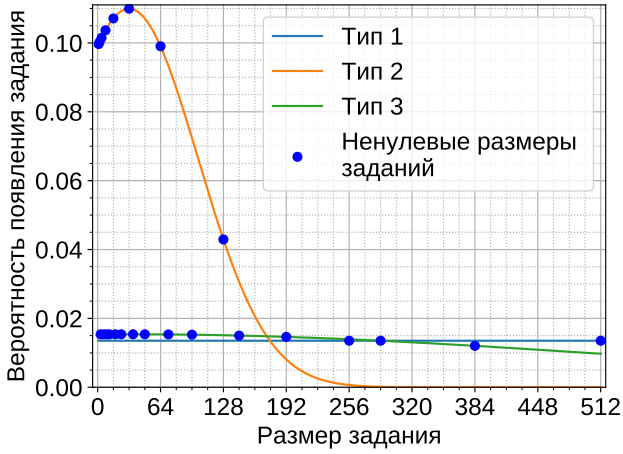
В таблице 4 представлены параметры для каждого типа задания и каждого размера вычислительной системы. На рисунке 5.10 представлены результирующие функции плотностей вероятностей появления задания в очереди: 5.10а – для систем с числом узлов 256; 5.10б – 512; 5.10в – 768; 5.10г – 1024.

Таблица 4 — Параметры распределения вероятности для систем с числом узлов 256, 512, 768, 1024.

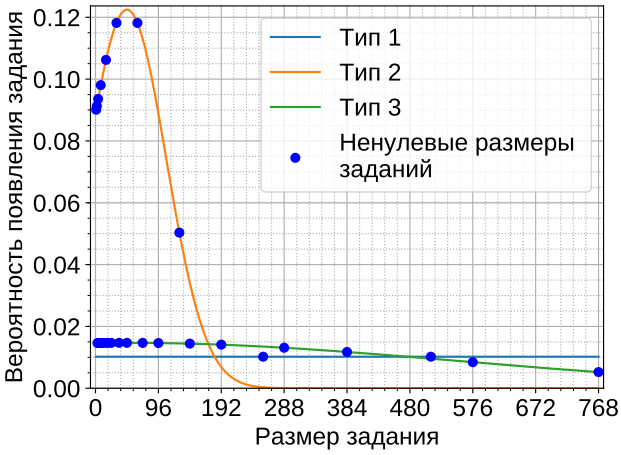
Число узлов	Пороговый размер, с которого размер задания считается большим	Общая вероятность заданий Типа 1	Параметры функции Гаусса для заданий			
			Типа 2		Типа 3	
			μ	σ	μ	σ
256	128	0.04	16	35	17	150
512	256	0.03	32	70	33	500
768	256	0.02	48	60	49	500
1024	256	0.04	64	50	65	150



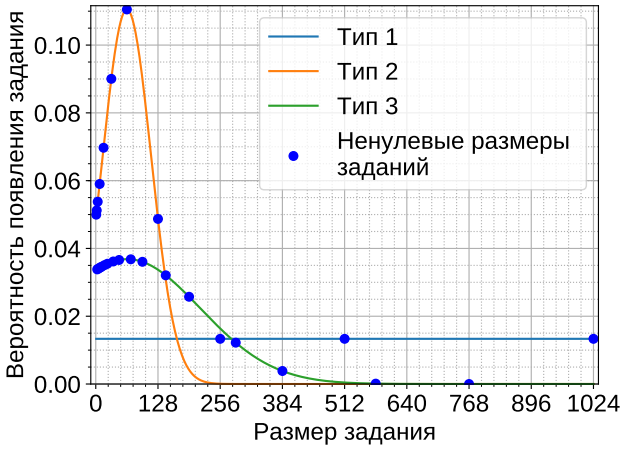
а)



б)



в)



г)

Рисунок 5.10 — Распределения долей размеров заданий для систем с размерами 256, 512, 768, 1024 узлов.

Для оценки продолжительности задания в очереди был проведен анализ запускаемых заданий на суперкомпьютере «Десмос». Было проанализировано 4972 заданий за 118 дней. Максимальная продолжительность задания на кластере равно одним суткам. В множестве запущенных заданий есть задания с ошибками (занимают очень мало времени), тестовые задания и нормальные пуски – длительные запуски.

На рисунке 5.11 представлена аппроксимирующая кривая продолжительности запускаемых заданий на суперкомпьютере «Десмос» за исследуемый промежуток времени. По оси X на диапазон от 0 до 1 отложены все задания, отсортированные по возрастанию продолжительности. По оси Y представлено требуемое время для задания в отношении к максимальному времени задания.

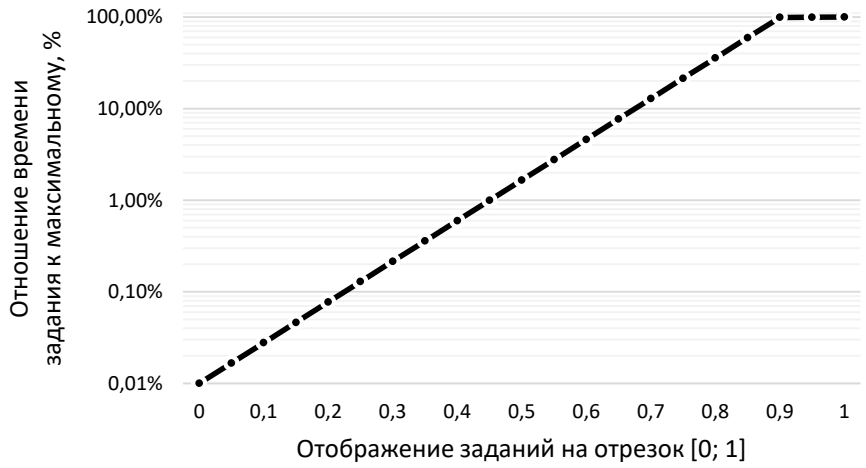


Рисунок 5.11 — Распределение продолжительности заданий.

Функция разбита на две составляющие: на интервале $[0; 0{,}9]$ представляет из себя 10 в степени линейной функции, такой что в точке 0 оно принимает значение 0,01%, а в точке 90 – значение 99%; на интервале $[0{,}9; 1]$ представляет линейную функцию и принимает значения от 99% до 100%.

На рисунке 5.12 представлен алгоритм построения очереди для имитационной модели вычислительной системы. Продолжительность заданий задается с помощью описанной функции на рисунке 5.11, для этого генерируется случайное число в диапазоне $[0; 1]$, и по нему определяется отношение продолжительности задания к максимальному времени задания (вызов get_task_time на рисунке 5.12).

Все задания равномерно случайно размещаются на временной шкале в диапазоне $\left[0; 60^2 * 24 * 30\right]$ секунд для распределений, полученных на суперкомпьютере «Десмос», и в диапазоне $\left[0; 4 * 60^2 * 24 * 30\right]$ секунд – для остальных распределений вне зависимости от числа требуемых узлов. Общее число заданий обеспечивает теоретическую утилизацию кластера в 80%.

Потребленными ресурсами задания W^i назовем произведение размера задания и продолжительности задания $P^i = W_t^i * W_N^i$. Задания генерируются до тех пор, пока сумма потребляемых ресурсов заданий не превысит 80% ресурсов кластера на заданном диапазоне времени имитации. Очередь заданий сортируется по времени появления задания.

```
Input:
N    -- размер вычислительной системы
Tsim -- Время симуляции
Tmax -- Максимальная продолжительность задания

Output:
Q -- очередь заданий

queue_load(vector<Task> &Q)
{
    sum = 0;
    for( int i = 0; i < Q.size(); ++i )
        sum += Q[i].delay * Q[i].nodes;
    return sum;
}

tasks_queue()
{
    vector <Task> Q;
    while( queue_load(Q) < Tsim * N * 0.8 ) {
        nodes = get_task_size(N);
        delay = get_task_time(Tmax, rand(0, 1));
        start = random(0, Tsim);
        Q += [ Task(start, nodes, delay) ]
    }
    Q.sort_by_start_time();
    return Q
}
```

Рисунок 5.12 — Псевдокод алгоритма построения очереди для имитационной модели вычислительной системы.

5.4 Исследование алгоритмов выбора множества узлов

В главе 4 при разработке алгоритма выбора достижимого множества узлов заданного размера сформулированы задачи исследования:

– сравнить алгоритмы по сформулированным критериям выбора лучшего решения из набора;
– оценить пригодность алгоритма с точки зрения применения на практике.

В данном разделе представлены результаты исследований по поставленным задачам.

5.4.1 Сравнение алгоритмов улучшенного перебора n-мерных прямоугольников и равномерного расширения

Исследование алгоритмов выбора множества узлов проводилось на системах с топологиями 4x4x4x4, 8x4x4 и 4x4x4 без сломанных каналов связи.

Для каждой топологии сети запускался алгоритм выбора множества узлов с числом узлов от 1 до максимального. Таблица маршрутов для сети строилась алгоритмом на основе поиска вширь по маршрутному графу без нарушения правила порядка направления для FS/LS.

Напомним список критериев выбора множества узлов из набора решений в рассматриваемой задаче выбора множества узлов:

1. минимизация числа транзитных узлов;
2. минимизация фрагментации вычислительной системы после выделения множества;
3. минимизация диаметра построенной таблицы маршрутов, где диаметр – это максимальная длина маршрута в исследуемом множестве;
4. минимизация максимальной загруженности π_{max} построенной таблицы маршрутов.

Решения алгоритма выбора множества узлов сортируются следующим образом. Сначала выбираются решения с наилучшим первым критерием, затем из выбранных решений отбираются решения с наилучшим следующим критерием и так далее.

В данном подразделе вместо критерия максимальной загруженности π_{max} рассматривался критерий минимизации отклонения $\sigma(4,R)$ построенной табли-

цы маршрутов от идеальной загруженности. Такая замена возможна, потому что указанные характеристики зависимы друг относительно друга.

В исследовании рассматривались улучшенный алгоритм выбора множества узлов перебором n-мерных прямоугольников (см. раздел 4.4), а также алгоритм равномерного расширения (см. раздел 4.5). Базовый алгоритм перебора n-мерных прямоугольников не рассматривался из-за его очевидных недостатков – ограничения при поиске и невозможность выбора систем, не являющихся прямоугольниками, которые не позволяют его использовать.

На рисунке 5.13 представлено исследование улученного алгоритма выбора множества узлов перебором n-мерных прямоугольников в топологии 4x4x4. В результате работы алгоритма находится множество решений, из которых выбирается одно согласно выставленным критериям. На графиках для каждого из исследуемых критериев представлено максимальное значение критерия из всех возможных решений, минимальное значение и значение для выбранного решения (жирным маркером).

Первым критерием идет минимизация числа транзитных узлов, поэтому на рисунке 5.13а можно наблюдать достижение минимума на всех размерах искомых множеств. На рисунках 5.13г и 5.13в представлено достижение минимизации максимальной загрузки канала и минимизации отклонения от идеальной таблицы маршрутов. На рисунке 5.13б представлено достижение минимального размера диаметра. Выбор неминимального диаметра для некоторых систем обусловлен условием минимизации числа транзитных узлов. Аналогично и для характеристики отклонения от идеальной загруженности.

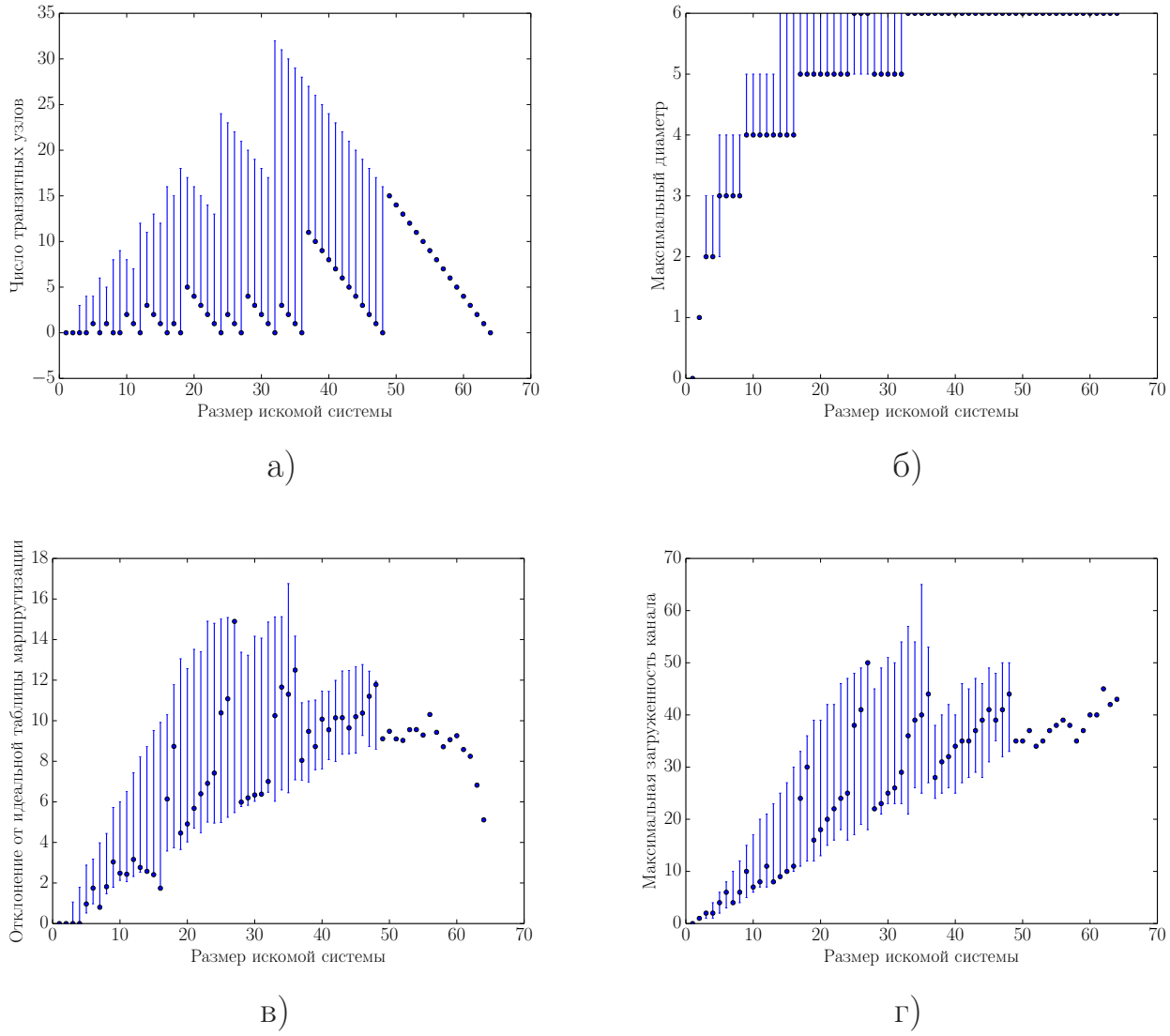


Рисунок 5.13 — Исследование улучшенного алгоритма выбора множества узлов перебором n-мерных прямоугольников в топологии 4x4x4.

На рисунке 5.14 представлено исследование алгоритма выбора множества узлов равномерным расширением в той же топологии 4x4x4. Ввиду специфики работы алгоритма и того, что поиск происходил в пустой системе без отказов, выбора транзитных узлов рассматриваемый алгоритм не предлагает. Это можно наблюдать на рисунке 5.14а. Рассматриваемый алгоритм минимизирует полученный диаметр вычислительной системы, это видно по рисунку 5.14б.

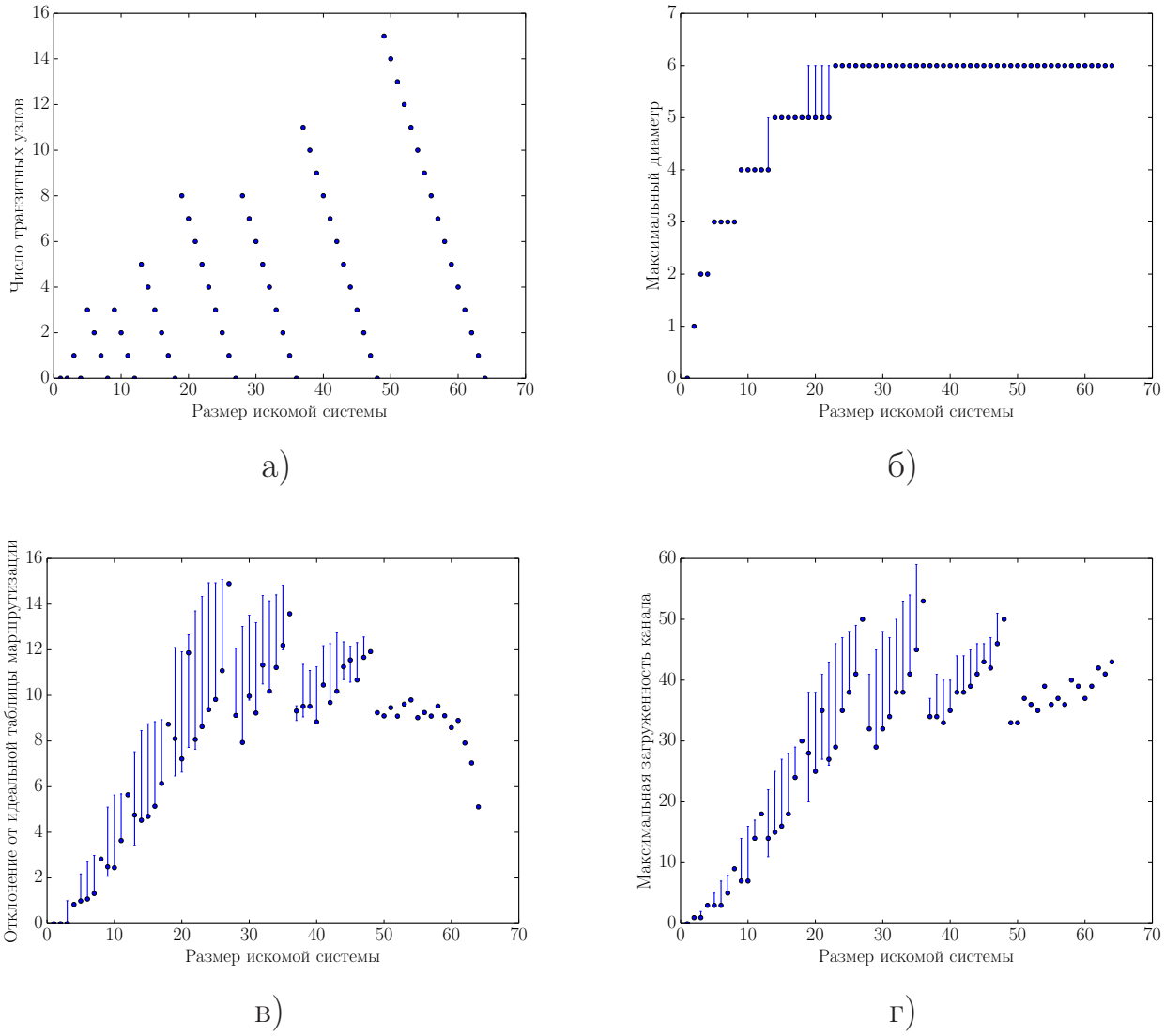


Рисунок 5.14 — Исследование алгоритма выбора множества узлов равномерным расширением в топологии 4x4x4.

Для остальных сетей: 8x4x4 – рисунок 5.15 для алгоритма перебора n-мерных прямоугольников и рисунок 5.16 для алгоритма равномерного расширения; 4x4x4x4 – рисунки 5.17 и 5.18, соответственно.

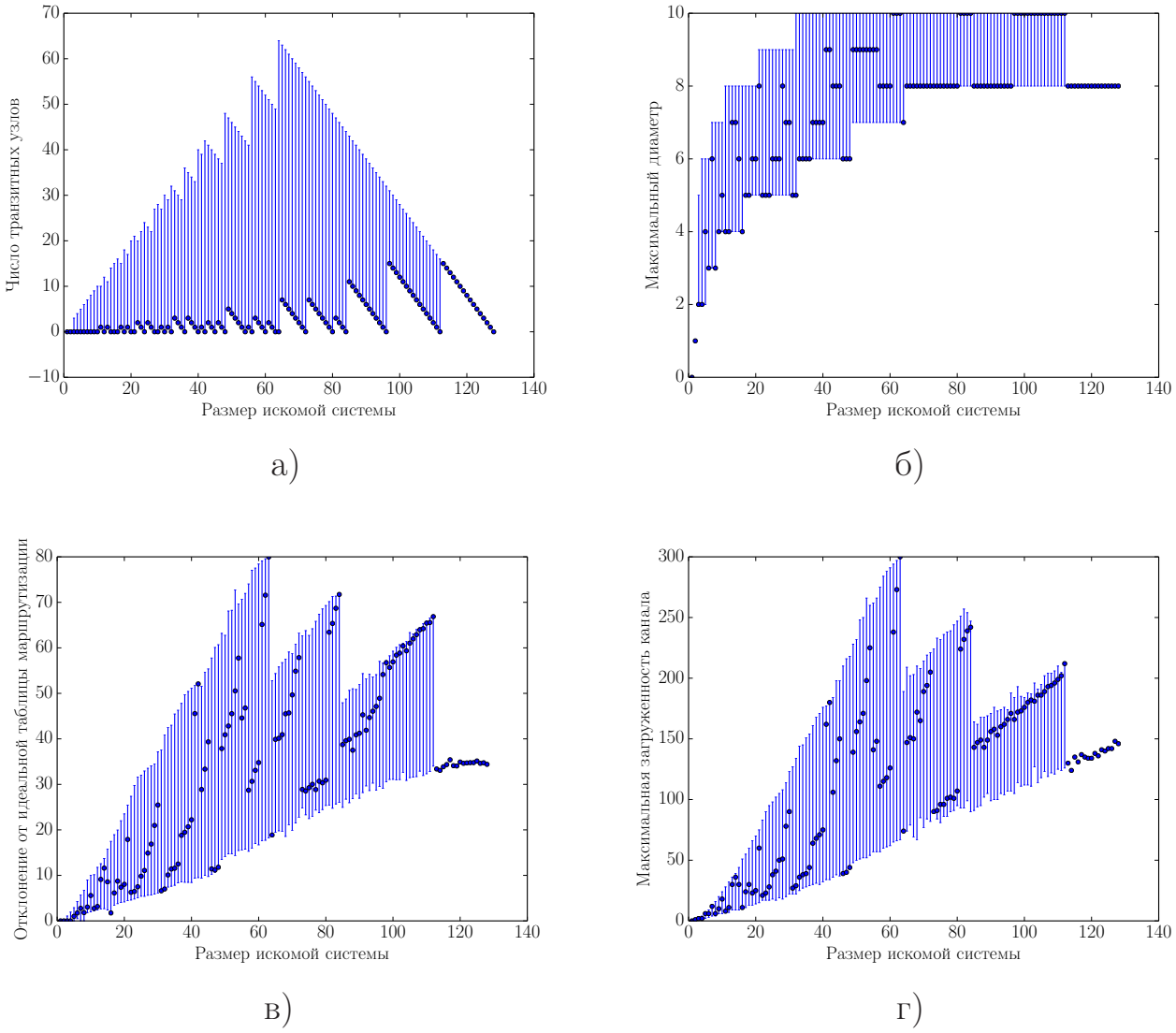


Рисунок 5.15 — Исследование улучшенного алгоритма выбора множества узлов перебором n-мерных прямоугольников в топологии 8x4x4.

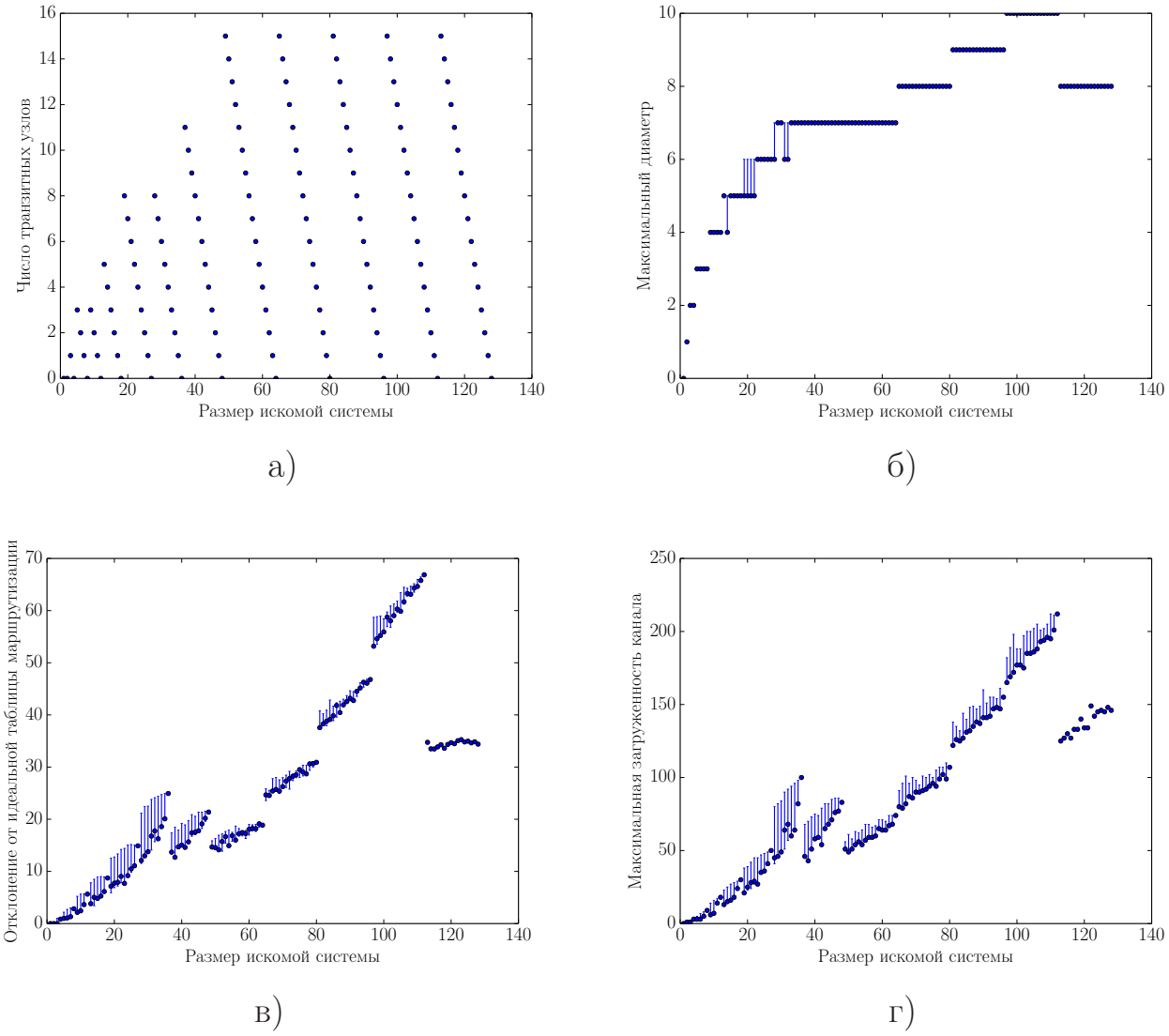


Рисунок 5.16 — Исследование алгоритма выбора множества узлов равномерным расширением в топологии 8x4x4.

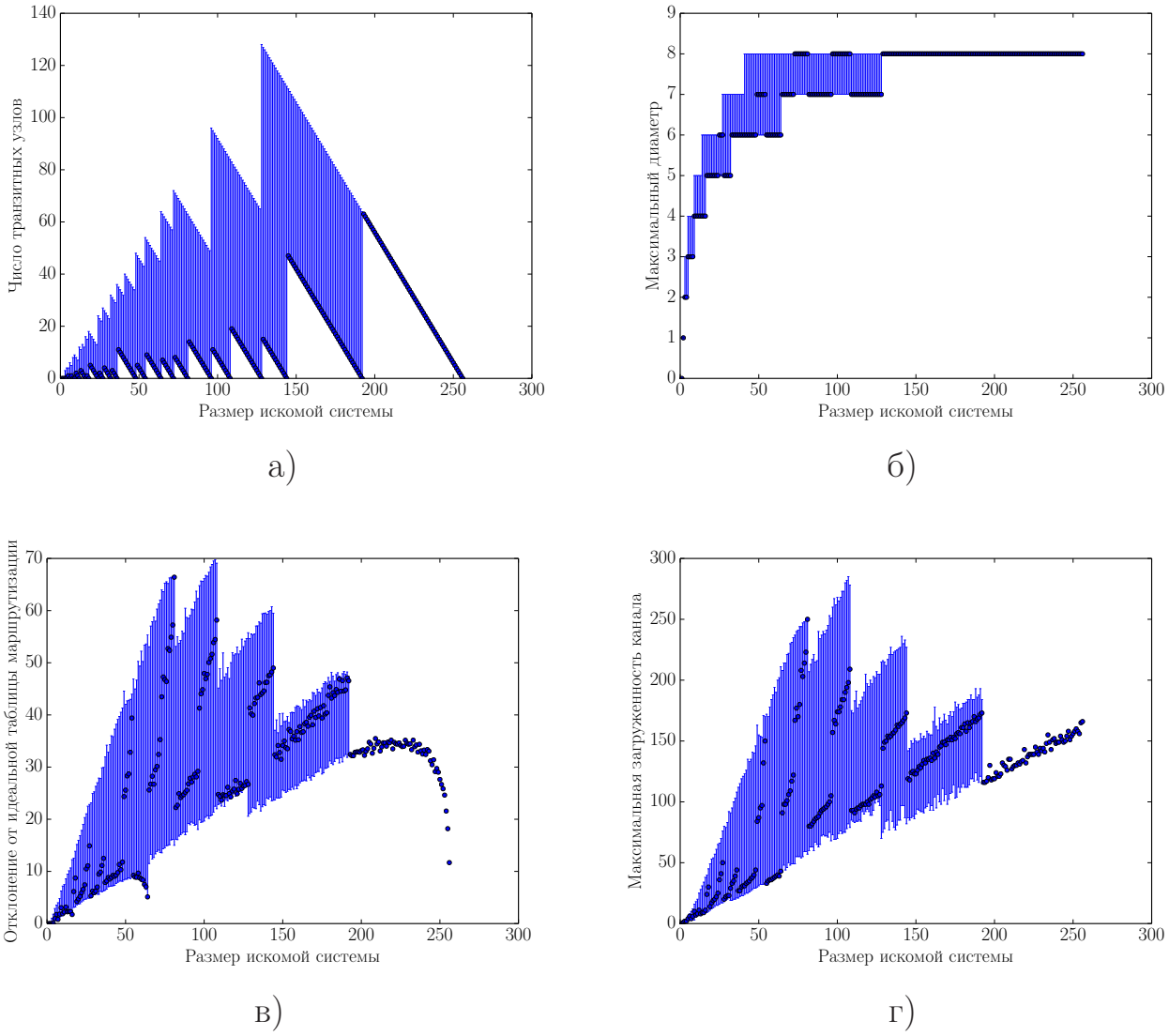


Рисунок 5.17 — Исследование улучшенного алгоритма выбора множества узлов перебором n-мерных прямоугольников в топологии 4x4x4x4.

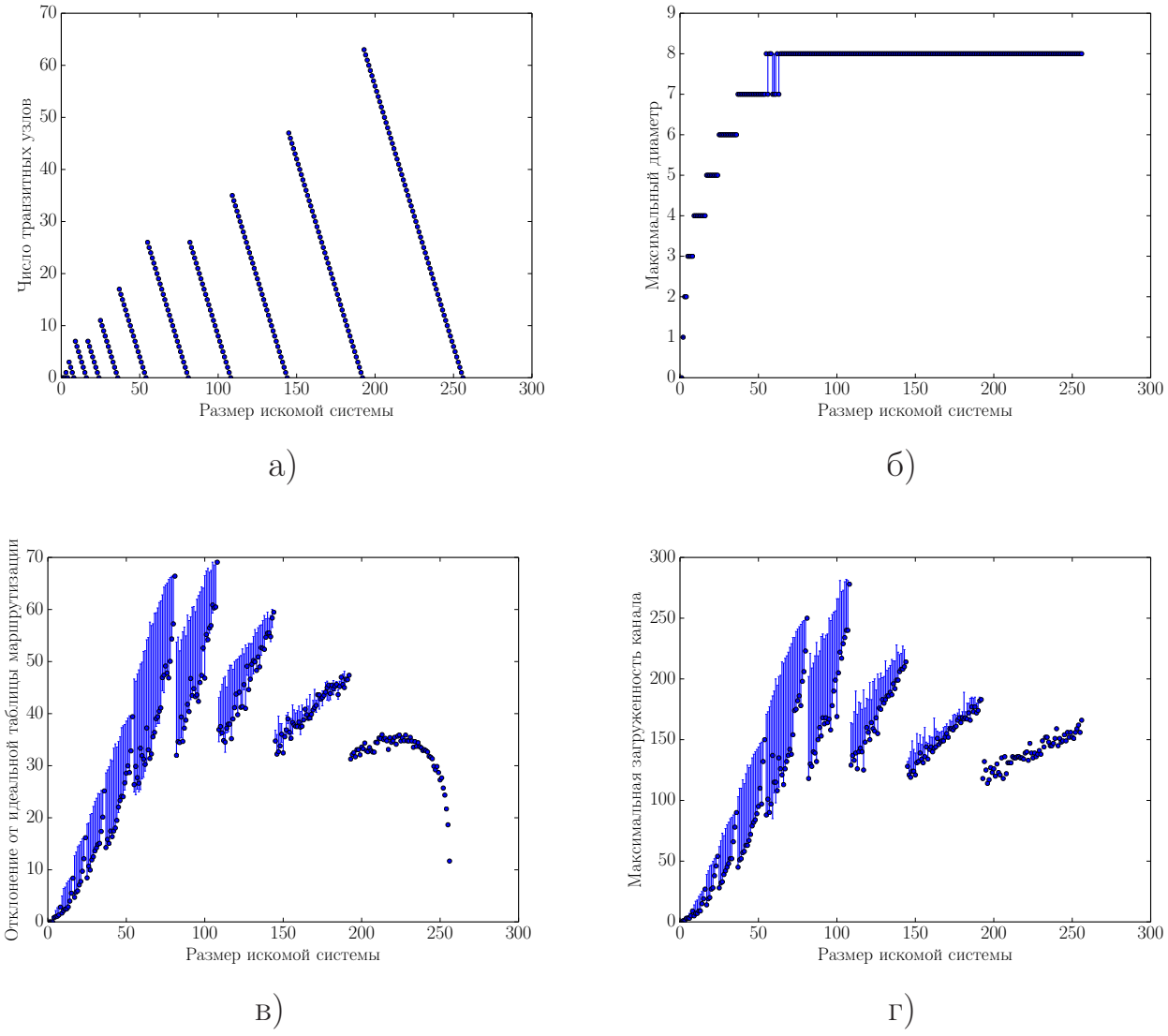


Рисунок 5.18 — Исследование алгоритма выбора множества узлов равномерным расширением в топологии 4x4x4x4.

В качестве промежуточного вывода можно заметить, что полученные значения критериев демонстрируют выбранный порядок критериев. Алгоритм равномерного расширения рассматривает меньшее количество вариантов решений и, следовательно, предоставляет меньше возможностей для оптимизации критериев.

В таблице 5 представлено среднее число найденных решений для каждой сети по всем поискам и среднее время, затраченное на поиск и выбор решения. Алгоритм равномерного расширения предлагает меньшее число решений, в результате для меньшего числа требуется построить таблицу маршрутов, что сокращает время поиска системы.

Однако скорость выбора достижимого множества, то есть полученное среднее время получения одного решения, деленное на количество найденных решений, для улучшенного алгоритма перебора n-мерных прямоугольников оказывается выше по сравнению с алгоритмом равномерного расширения. Таким образом, при применении улучшенного алгоритма перебора на практике можно ограничить перебор заданным числом вариантов решений и получить меньшее время работы алгоритма.

Таблица 5 — Среднее число найденных решений и среднее время, затраченное на поиск и выбор системы.

	Ул. перебор n-мерных пр.-ов.		Равномерного расширения	
	ср. число систем	ср. время, с	ср. число систем	ср. время, с
4x4x4	298	0,14	30	0,02
8x4x4	1069	1,42	43	0,12
4x4x4x4	4270	22,4	91	0,83

В таблице 6 представлено относительное сравнение алгоритмов выбора множества узлов. Для каждой топологии сети изучено, на сколько улучшенный алгоритм перебора n-мерных прямоугольников предложил значение критерия лучше, чем алгоритм равномерного расширения в процентном соотношении. Для сети 8x4x4 ухудшение таких характеристик, как диаметр и отклонение от идеальной таблицы связано с выбором решений с меньшим числом транзитных узлов, в результате такого выбора получаются плотные системы, в которых на имеющееся число каналов связи приходится больше маршрутов, что приводит к ухудшению этих характеристик.

Таблица 6 — Среднее отношение получаемых критериев алгоритмом перебора n-мерных прямоугольников к алгоритму равномерного расширения.

Порядок критерия		4x4x4x4	8x4x4	4x4x4
1	Число транзитных узлов	81%	55%	81%
2	Оценка фрагментации	141%	101%	133%
3	Диаметр	96%	104%	97%
4	Критерий оценки таблиц маршрутов	85%	125%	90%

В результате проведенного эксперимента улучшенный алгоритм перебора n-мерных прямоугольников показал лучшие результаты и показывает приемлемые результаты по времени и хорошие характеристики полученных решений. Алгоритм равномерного расширения работает быстрее и может использоваться для оценки наличия требуемого достижимого множества.

В результате проведенных исследований для дальнейшей работы в качестве основного выбран алгоритм улучшенного перебора n-мерных прямоугольников. Построение таблиц маршрутов происходит при помощи алгоритма поиска вширь по маршрутному графу.

5.4.2 Сравнение алгоритмов улучшенного перебора n-мерных прямоугольников и базового

Для того, чтобы сравнить в одинаковых условиях во время работы вычислительной системы разработанный улучшенный алгоритм перебора n-мерных прямоугольников с базовым алгоритмом, который изначально работал на вычислительных кластерах с сетью Ангара, проведено исследование сравнения количества возможных решений задачи выбора узлов, найденных с помощью разработанного алгоритма и с помощью базового алгоритма. В исследовании рассматривались различные топологии сетей с числом узлов от 32 до 256. Для каждой топологии запускалась модель с окном заданий размера 1 и соответствующей синтетической очередью заданий, которые описаны в разделе 5.3.1. Имитационная модель обрабатывала очередь с помощью базового алгоритма, при этом запоминалось количество найденных решений для каждого запуска базового алгоритма. Для каждого состояния имитационной модели перед каждым запуском базового алгоритма также запускался разработанный алгоритм выбора узлов, для которого также запоминалось количество найденных решений. Окно размера один выбрано ввиду того, что сравниваются возможности исследуемых алгоритмов, а не политики обработки очереди. Описанные условия приближают сравнение к реальным системам, а представленный набор сетей шире набора топологий существующих суперкомпьютеров на основе сети Ангара.

В разработанном алгоритме применялся сформулированный в главе 4 порядок критериев отбора решений. Допустимое число транзитных узлов T

выбрано равным 100%, то есть $T = m$, где m – требуемое число вычислительных узлов.

В таблице 7 представлено среднее количество найденных решений для обоих алгоритмов для каждой сети и их отношение. Разработанный улучшенный алгоритм выбора узлов перебором n-мерных прямоугольников предлагает в 2–12 раз больше вариантов решений, чем базовый.

Таблица 7 — Исследование среднего количества решений, найденных базовым и улучшенным алгоритмами перебора на имитационной модели.

Топология	Кол-во узлов	Среднее число найденных систем		Отношение разработанного алг. к базовому
		Базовый алгоритм	Разработанный алгоритм	
4x4x2	32	8,29	28,54	3,44
4x2x2x2	32	8,26	26,08	3,16
4x3x3	36	11,41	29,20	2,56
3x3x2x2	36	11,49	22,79	1,98
4x4x4	64	15,28	92,85	6,08
4x4x2x2	64	15,27	75,15	4,92
6x4x4	96	39,18	237,54	6,06
4x4x3x2	96	32,69	166,15	5,08
8x6x3	144	95,08	479,96	5,05
4x4x3x3	144	98,93	520,61	5,26
8x8x4	256	214,93	1179,75	5,49
4x4x4x4	256	162,29	1929,15	11,89

5.5 Исследование утилизации вычислительной системы

В результате проведенных исследований в качестве основных выбраны улучшенный алгоритм перебора n-мерных прямоугольников и построение таблиц маршрутов при помощи алгоритма поиска вширь по маршрутному графу. Общая полученная схема работы алгоритма поиска достижимого множества со сбалансированной таблицей маршрутов, которая реализована в плагине для Slurm, представлена на рисунке 5.19. При каждом запуске плагина для выбора узлов сначала считывается состояние системы о занятых узлах и сломанных каналах связи, затем инициализируется маршрутный граф, затем в рамках

улучшенного алгоритма перебора n-мерных прямоугольников происходит поиск всех разложений заданного числа требуемых узлов и допустимых транзитных узлов, затем происходит проверка достижимости найденных вариантов методом поиска вширь в графе (BFS) и затем выбирается наилучший ответ.

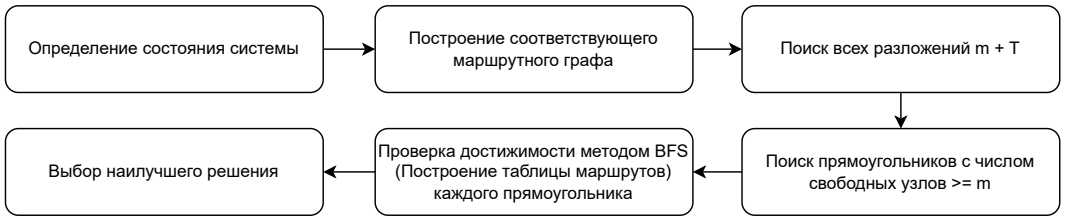


Рисунок 5.19 — Схема работы выбора достижимого множества узлов, реализованная в плагине для Slurm.

Для того, чтобы продемонстрировать работу вычислительной системы с использованием разработанных алгоритмов и оценить эффективность функционирования вычислительной системы, необходимо провести исследование ее утилизации при выполнении очереди заданий пользователей. Исследования проводились с помощью разработанной имитационной модели вычислительной системы.

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 в данной работе по аналогии с работой [96].

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 узла сети, T – время работы вычислительного кластера, T_i – полезное время работы i-го узла сети. Примечание: когда узел выделен как транзитный, считается, что он занят, но не выполняет полезную работу.

Обозначим *значение времени нахождения задания в очереди относительно запрошенного времени* как $T^i_{delay} = \frac{Q^i}{W^i_t}$, где W^i_t – запрошенное время (длительность) 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 $T_{mean} = \frac{\sum_{i=1}^{k} T^i_{delay}}{k} = \frac{1}{k}\sum_{i=1}^{k} \frac{Q^i}{W^i_t}$, где k – число различных заданий в потоке.

В диссертационной работе имитируется работа вычислительной системы и системы управления ресурсами Slurm. В работе рассматривается используемая по умолчанию в Slurm политика FCFS, в которой задания выполняются строго по порядку поступления в очередь. Также в работе рассмотрена оптимизация алгоритма FCFS: имитационной модели позволено обработать следующее задание в рамках некоторого окна, если для предыдущего задания в данный момент нельзя выделить ресурсы.

Оптимизирующая политика выделения ресурсов Backfill не рассматривается, так как для ее работы необходимо указывание пользователями максимального времени, необходимого для выполнения задания, а в реальных вычислительных системах с сетью Ангара пользователи не указывают это время. Также не рассматривается применение политики multifactor для справедливого распределения ресурсов кластера, так как в рассматриваемых вычислительных системах эта политика не используется.

В разделе 5.3 описана работа имитационной модели вычислительной системы и представлен алгоритм создания очереди пользовательских заданий. В текущем разделе представлены следующие исследования.

- Результаты исследования разработанных алгоритмов на реальном суперкомпьютере приведены в разделе 5.5.1.
- Исследование нескольких модификаций алгоритма выбора узлов и влияния критерия выбора достижимого множества на утилизацию ресурсов вычислительного кластера и среднее значение времени нахождения задания в очереди приведены в разделе 5.5.2.
- Исследование влияния размера окна на утилизацию ресурсов вычислительного кластера для потока пользовательских заданий для малых и средних систем до 144 узлов приведено в разделе 5.5.3 и для больших систем от 144 до 1024 узлов приведено в разделе 5.5.4.
- В разделе 5.5.4 представлены выводы.

5.5.1 Результаты на реальном суперкомпьютере

В имитационной модели вычислительной системы имеется ряд упрощений, в ней не учитываются задержки на запуск пользовательских заданий и на подготовку ресурсов к использованию. В связи с этим проведено исследование утилизации ресурсов на реальном суперкомпьютере «Десмос», установленном в ОИВТ РАН, с целью сравнения имитационной модели с реальным суперкомпьютером. Также данное исследование позволяет оценить реальный эффект внедрения разработанных в диссертационной работе алгоритмов.

Суперкомпьютер «Десмос» состоит из 32 вычислительных узлов, объеденных сетью Ангара с топологией сети 4x2x2x2. Управление ресурсами осуществляется с помощью менеджера ресурсов Slurm. Для запуска пользовательских заданий длительностью более 15 минут выделены следующие партиции: max1n, в которой можно выделить задания размера [1,2] узлов; max8n – [4,8]; max16n – [8, 16]; max32n – [16,32] узлов. Ввиду этих особенностей распределение размеров заданий для системы из 32 узлов, представленное на рисунке на рисунке 5.9a, было модифицировано, из него исключен размер задания равный 3. Итоговое распределения представлено на рисунке 5.20.

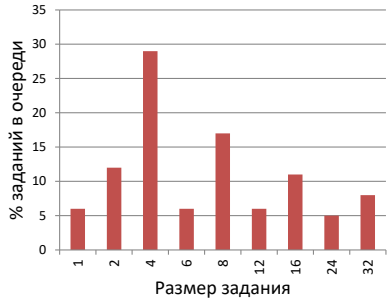


Рисунок 5.20 — Модифицированное распределение размеров заданий для системы размером 32 узла.

Для того, чтобы сократить время эксперимента на реальном суперкомпьютере, результирующая тестовая очередь была пропорционально уменьшена (масштабирована) в 35 раз, то есть все времена запусков и продолжительности заданий сокращены в 35 раз. Время инициализации (запуска) задания на суперкомпьютере «Десмос» не превышает 20 секунд. Чтобы покрыть задержки на запуск задания, минимальное время задания выставленно в 1 минуту для масштабированной очереди. Таким образом, распределение времен заданий удовлетворяет тем же законам, что и на рисунке 5.11, за исключением ограничения минимального времени задания.

Для исследования создана очередь с распределением, представленным на рисунке 5.20. Время старта заданий распределялось равномерно на временной шкале в диапазоне $[0; 60^2 * 24 * 14]$ секунд (до масштабирования 1/35). Очередь заданий обеспечивает утилизацию суперкомпьютера 80% без учета топологии сети. При моделировании очереди на имитационной модели применялась схема с размером окна равного 1 (фактически FIFO), на суперкомпьютере «Десмос» в менеджере Slurm настроена политика выбора заданий при помощи алгоритма алгоритма обратного заполнения (backfill). Допустимое число транзитных узлов выбрано равным 100% – то есть, $T = m$, где m – требуемое число вычислительных узлов, T – параметр алгоритма перебора n-мерных прямоугольников. Порядок критериев отбора решений соответствует порядку, сформулированному в главе 4.

Таблица 8 — Результаты выполнения очереди заданий на суперкомпьютере «Десмос» и моделирования той же очереди на имитационной модели при работе улучшенного (разработанного) алгоритма перебора и базового алгоритмов выбора узлов.

Характеристика	Базовый алгоритм		Разраб. алгоритм		Сравнение	
	Сим.	Реал.	Сим.	Реал.	Сим.	Реал.
Утилизация U	58,54%	59,35%	65,80%	67,00%	7,26%	7,65%
Ожидание T_{mean}	17,02	18,28	7,34	8,63	2,32	2,12

В таблице 8 представлены результаты выполнения описанных выше в данном подразделе очередей заданий на суперкомпьютере «Десмос» и моделирования той же очереди на имитационной модели при работе разработанного улучшенного алгоритма перебора и базового алгоритма выбора узлов. В качестве результатов приводится утилизация и относительное время ожидания задания в очереди. Улучшение утилизации на суперкомпьютере при применении разработанного алгоритма выбора узлов по сравнению с базовым сохраняется по отношению к исследованию на имитационной модели и составляет 7,65%. Аналогично улучшилась и оценка ожидания задания в очереди, которое сократилось для суперкомпьютера в 2,12 раз. Разница значений на суперкомпьютере и модели, в первую очередь, обусловлена политикой выбора задания – backfill на суперкомпьютере и FIFO для модели. В то же время адекватность имитационной модели подтверждается разницей полученной утилизации на модели и суперкомпьютере – не более 1,2% абсолютных пунктов.

Таким образом, эксперимент на реальном суперкомпьютере «Десмос» показывает, что внедрение разработанного алгоритма выбора узлов с применением разработанного алгоритма построения таблицы маршрутов позволило улучшить эксплуатационные характеристики суперкомпьютера.

5.5.2 Влияние критерия выбора достижимого множества узлов на исследуемые характеристики

В данном исследовании рассматривались системы с числом узлов в сети до 144 узлов. Исследуемые топологии представлены в таблице 9. Исследования проводились с помощью имитационной модели вычислительной системы на окнах заданий размера $1, 2, 4, 8, 16, 32, 64, 128$. Очереди заданий задавались в соответствии с разделом 5.3.1. Допустимое число транзитных узлов T выбрано равным 25%, то есть $T = m/4$, где m – требуемое число вычислительных узлов.

Таблица 9 — Малые и средние моделируемые системы.

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

В главе 4 сформулирован порядок критериев отбора решений задачи выбора узлов. В данном разделе вместо критерия максимальной загруженности π_{max} рассматривался критерий минимизации отклонения $\sigma(4,R)$ построенной таблицы маршрутов от идеальной загруженности. Такая замена возможна, потому что указанные характеристики зависимы друг относительно друга. Таким образом, рассматривался следующий список критериев.

1. Минимизация числа транзитных узлов.
2. Минимизация фрагментации вычислительной системы после выделения множества.
3. Минимизация диаметра построенной таблицы маршрутов, где диаметр – это максимальная длина маршрута в исследуемом множестве.

4. Минимизация отклонения $\sigma(4,R)$ построенной таблицы маршрутов от идеальной загруженности.

Критерий о минимизации числа транзитных узлов самый важный, так как он необходим для эффективного использования аппаратных ресурсов вычислительного кластера. Второй критерий также призван максимально эффективно использовать систему при помощи анализа множества различных прямоугольников максимально возможного размера, характеризующих меру фрагментированности сети. Третий критерий позволяет минимизировать максимальную задержку внутри выбранного множества, выделяемое множество узлов при этом становится более квадратным. И последний критерий максимизирует сбалансированность получаемой таблицы маршрутов.

Однако важно показать, как порядок критериев влияет на утилизацию вычислительной системы. Было проведено исследование утилизации и среднего значения времени нахождения задания в очереди для различных порядков критериев:

– T;mss;d;Gi;D – транзитные узлы, функция оценки фрагментированности сети, диаметр, максимальная загрузка канала, отклонение степени 4;
– mss;d;Gi;T;D – функция оценки фрагментированности сети, диаметр, максимальная загрузка канала, транзитные узлы, отклонение степени 4;
– d;Gi;T;D;mss – диаметр, максимальная загрузка канала, транзитные узлы, отклонение степени 4, функция оценки фрагментированности сети.

На рисунке 5.21 представлено среднее значение утилизации вычислительного кластера по всем системам в зависимости от размера окна. На рисунке 5.22 представлено среднее значение времени нахождения задания в очереди по всем системам в зависимости от размера окна. Имитационная модель запускалась с использованием разработанного алгоритма выбора узлов улучшенным перебором многомерных прямоугольников с оценкой фрагментированности сети.

	1	2	4	8	16	32	64	128
d;Gi;T;D;mss	55,93	55,72	57,49	60,45	66,28	71,40	75,60	76,26
mss;d;Gi;T;D	56,23	55,98	57,71	61,24	66,50	72,16	75,86	76,40
T;mss;d;Gi;D	56,36	56,23	57,85	61,29	66,67	71,99	75,70	76,41

Рисунок 5.21 — Результаты исследования утилизации (в среднем по всем системам) в зависимости от порядка критериев.

	1	2	4	8	16	32	64	128
d;Gi;T;D;mss	23464,49	23485,93	21322,81	18041,23	11428,57	5806,10	2115,70	1413,80
mss;d;Gi;T;D	23116,54	23072,24	20691,64	17062,93	11096,44	5196,52	2060,82	1330,43
T;mss;d;Gi;D	22781,89	22992,44	20654,60	17118,71	10765,65	5420,94	2024,73	1288,50

Рисунок 5.22 — Результаты исследования относительного времени ожидания (в среднем по всем системам) в зависимости от порядка критериев.

В среднем по всем размерам окон заданий лучшую утилизацию показала последовательность критериев T;mss;d;Gi;D, которая фактически и выбрана в главе 4. Для удобства усредненные результаты представлены в таблице 10. Самые плохие результаты были получены на последовательности критериев d;Gi;T;D;mss, так как в ней критерии, отвечающие за эффективность использования вычислительной системы, идут на последнем месте. В следующих исследованиях используется установленный порядок критериев – T;mss;d;Gi;D.

Таблица 10 — Результаты исследования влияния различных порядков критериев, усредненные по всем размерам окон и всем системам.

	mss;d;Gi;T;D	d;Gi;T;D;mss	T;mss;d;Gi;D
Утилизация, %	65,3	64,9	65,3
Относительное время ожидания задания в очереди	12953	13384	12880

5.5.3 Исследование разработанных алгоритмов по сравнению с базовым на малых и средних системах

В данном исследование рассматривались вычислительные системы с числом узлов сети до 144 узлов. Исследуемые топологии представлены в таблице 9. Исследования проводились с помощью имитационной модели вычислительной системы на окнах заданий размера $1, 2, 4, 8, 16, 32, 64, 128$. Очереди заданий задавались в соответствии с разделом 5.3.1. Допустимое число транзитных узлов T выбрано равным 100%, то есть $T = m$, где m – требуемое число вычислительных узлов. Порядок критериев отбора решений соответствует порядку, сформулированному в главе 4.

В исследовании рассматривались три варианта алгоритма выбора узлов. Среди них два варианта разработанного алгоритма:

– Алгоритм выбора узлов улучшенным перебором многомерных прямоугольников с и без оценки фрагментированности вычислительной системы.
– Базовый алгоритм выбора узлов перебором многомерных прямоугольников, который изначально работал на вычислительных кластерах с сетью Ангара.

Базовый алгоритм обладает рядом недостатков, среди которых, напомним, ограничения при поиске и невозможность выбора систем, не являющихся прямоугольниками.

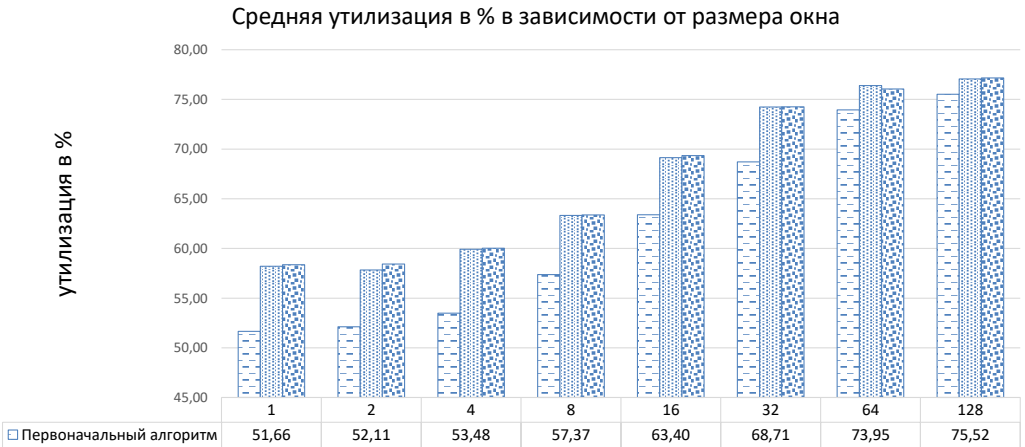
На рисунке 5.23 представлено среднее значение утилизации вычислительного кластера по всем системам в зависимости от размера окна. На рисунке 5.24 представлено среднее относительное время ожидания задания в очереди по всем системам в зависимости от размера окна. В таблице 11 представлены основные результаты проведенного исследования. Так, разработанный алгоритм с приме-

нением оценки фрагментированности дает увеличение утилизации в среднем на 0,11% относительно его же без оценки фрагментированности, а разработанный алгоритм с учетом фрагментации на 5,1% лучше базового. Применение разработанного алгоритма с учетом оценки фрагментированности позволяет сократить относительное время ожидания задания в очереди в 1,01 раз относительно без оценки фрагментированности и в 1,79 раз с оценкой фрагментированности относительно базового алгоритма.

При использовании алгоритма с оценкой фрагментированности увеличение размера окна приводит к увеличению утилизации максимально на 24,4% и к уменьшению среднего времени ожидания задания в очереди до 25,3 раз.

Таблица 11 — Исследование разработанных алгоритмов по сравнению с базовым на малых и средних системах от 32 до 144 узлов.

Сравнение средней утилизация в % по всем размерам окон и системам			
Сравниваемые алгоритмы	Минимальное, %	Среднее, %	Максимальное, %
Разработанный алгоритм с учетом фрагментированности относительно без учета фрагментированности	-2,92 %	0,11 %	2,06 %
Разработанный алгоритм с учетом фрагментированности относительно базового	0,5 %	5,1 %	13 %
Сравнение относительного времени ожидания задания в очереди по всем размерам окон и системам			
Сравниваемые алгоритмы	Минимальное	Среднее	Максимальное
Разработанный алгоритм с учетом фрагментированности относительно без учета фрагментированности	0,68	1,01	1,41
Разработанный алгоритм с учетом фрагментированности относительно базового	1,11	1,79	4,33



	1	2	4	8	16	32	64	128
Первоначальный алгоритм	51,66	52,11	53,48	57,37	63,40	68,71	73,95	75,52
Разр-й алг. без учета ПМР	58,21	57,83	59,91	63,31	69,14	74,24	76,39	77,06
Разр-й алг. с учетом ПМР	58,36	58,42	60,03	63,37	69,35	74,25	76,05	77,17

Рисунок 5.23 — Среднее значение утилизации по всем малым и средним системам от 32 до 144 узлов в зависимости от размера окна.

	1	2	4	8	16	32	64	128
Первоначальный алгоритм	31124	30000	28246	22531	15068	8635	3809	2139
Разр-й алг. без учета ПМР	20586	21226	18401	15114	8602	3778	1623	1112
Разр-й алг. с учетом ПМР	20224	20197	17852	15002	8366	3834	1641	1086

Рисунок 5.24 — Среднее по всем малым и средним системам от 32 до 144 узлов относительное время ожидания в зависимости от размера окна.

На рисунке 5.25 представлено среднее по всем запускам время работы улучшенного алгоритма выбора узлов в зависимости от топологии систем.

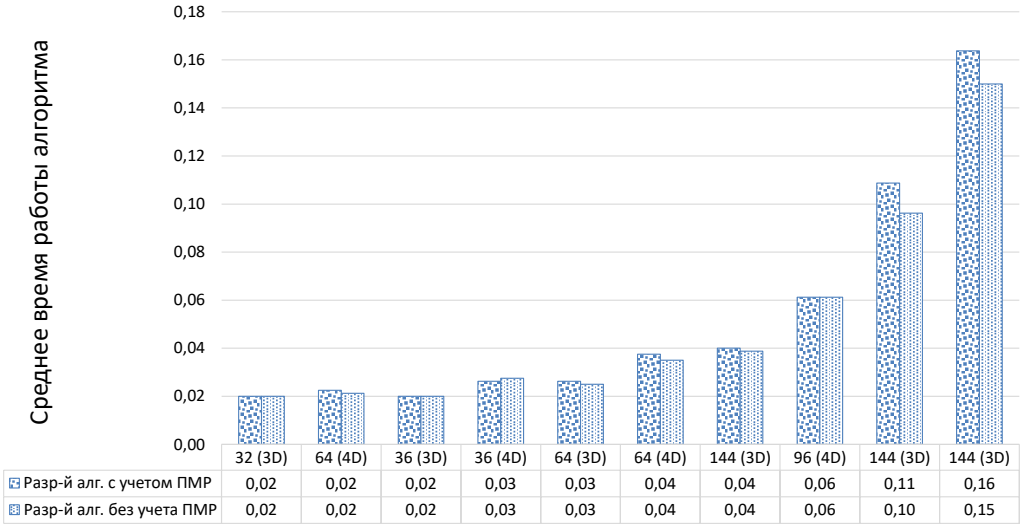


Рисунок 5.25 — Среднее время работы разработанного улучшенного алгоритма перебора в зависимости от топологии систем от 32 до 144 узлов. ПМР обозначает оценку фрагментации системы.

5.5.4 Исследование разработанных алгоритмов по сравнению с базовым на больших системах

В данном исследовании рассматривались системы от 256 до 1024 узлов. Исследуемые топологии представлены в таблице 12. Исследования проводились с помощью имитационной модели вычислительной системы на окнах заданий размера $1, 2, 4, 8, 16, 32, 64, 128$. Очереди заданий задавались в соответствии с разделом 5.3.1. Допустимое число транзитных узлов T выбрано равным 25%, то есть $T = m/4$, где m – требуемое число вычислительных узлов. Порядок критериев отбора решений соответствует порядку, сформулированному в главе 4.

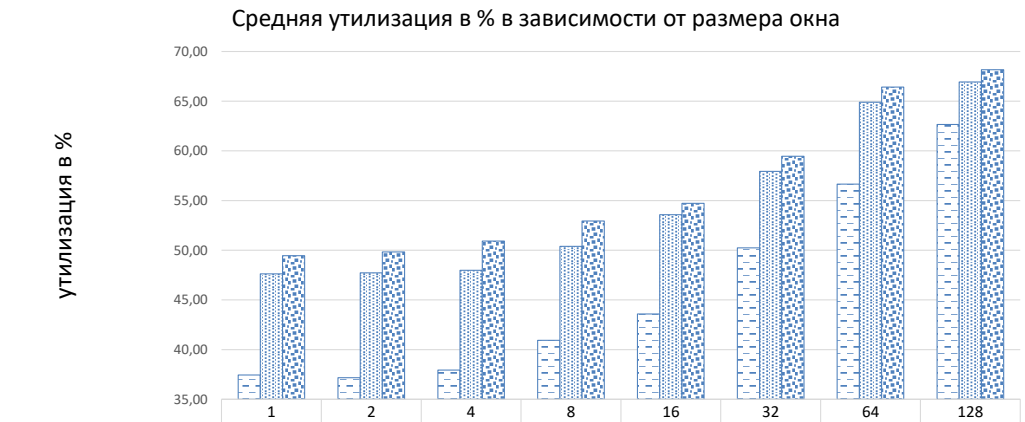
Таблица 12 — Моделируемые системы от 256 до 1024 узлов.

Кол.-во узлов выч. системы	Топология 3х-мерный тор	Топология 4х-мерный тор
256	8x8x4	4x4x4x4
512	8x8x8	8x4x4x4
768	–	8x6x4x4
1024	–	8x8x4x4

Для работы на большом количестве узлов проведена оптимизация алгоритма выбора узлов. Для сокращения перебора в улучшенный алгоритм перебора многомерных прямоугольников добавлено условие, что корректными

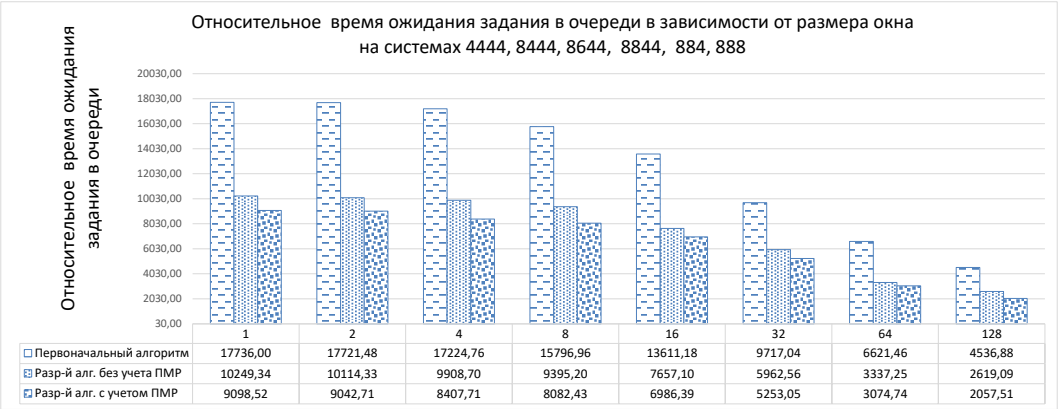
считаются только те решения задачи выбора узлов, которые располагались в непосредственной близости с занятыми или отказавшими узлами.

Имитационная модель запускалась в режиме отсутствия недоступных каналов связи, а также проверялись только те возможные конфигурации, которые являются прямоугольниками. При таких предположениях нет необходимости проверять каждый раз систему на достижимость, поэтому при выборе решения отсутствуют критерии, связанные с таблицей маршрутов. Данный алгоритм, тем не менее, отличается от базового, у которого имеется ограничение перебора: множители p_i варианта решения (см. 4.1) могут быть либо $p_i \leqslant \lceil d_i/2 \rceil$, либо $p_i = d_i$, где d_i – размерность i измерения тора. Из всех возможных найденных решений отбирались с наименьшей фрагментированностью (при условии включенной возможности, регулируемой параметром, см. ниже) и с наименьшим диаметром. Для выбранной системы строилась таблица маршрутов.



	1	2	4	8	16	32	64	128
Первоначальный алгоритм	37,44	37,17	37,94	40,93	43,59	50,25	56,65	62,66
Разр-й алг. без учета ПМР	47,61	47,72	47,98	50,40	53,58	57,93	64,90	66,93
Разр-й алг. с учетом ПМР	49,46	49,83	50,93	52,95	54,71	59,47	66,42	68,17

Рисунок 5.26 — Среднее значение утилизации по всем большим системам от 256 до 1024 узлов в зависимости от размера окна.



	1	2	4	8	16	32	64	128
Первоначальный алгоритм	17736,00	17721,48	17224,76	15796,96	13611,18	9717,04	6621,46	4536,88
Разр-й алг. без учета ПМР	10249,34	10114,33	9908,70	9395,20	7657,10	5962,56	3337,25	2619,09
Разр-й алг. с учетом ПМР	9098,52	9042,71	8407,71	8082,43	6986,39	5253,05	3074,74	2057,51

Рисунок 5.27 — Среднее относительное время ожидания по всем большим системам от 256 до 1024 узлов в зависимости от размера окна.

На рисунке 5.26 представлено среднее значение утилизации вычислительного кластера по всем системам в зависимости от размера окна, а на рисунке 5.27 – среднее значение относительного времени ожидания в очереди по всем системам в зависимости от размера окна.

В таблице 13 представлено сравнительные результаты проведенного исследования. Так, разработанный алгоритм с применением разработанной оценки фрагментированности дает увеличение утилизации в среднем на 1,86% относительно без оценки фрагментированности и на 10,67% относительного базового. Применение разработанного алгоритма с разработанной оценкой фрагментированности позволяет сократить относительное время ожидания задания в очереди в 1,16 раз относительно без оценки фрагментированности и в 1,91 раз относительно базового алгоритма.

Таблица 13 — Исследование разработанных алгоритмов по сравнению с базовым на больших системах от 256 до 1024 узлов.

Сравнение средней утилизации $U, \%$ по всем размерам окон и системам			
Сравниваемые алгоритмы	Минимальное, %	Среднее, %	Максимальное, %
Разработанный алгоритм с учетом фрагментированности относительно без учета фрагментированности	-2,05 %	1,86 %	5,42 %
Разработанный алгоритм с учетом фрагментированности относительно базового	3,36 %	10,67 %	16,46 %
Сравнение относительного времени ожидания задания в очереди T_{mean} по всем размерам окон и системам			
Сравниваемые алгоритмы	Минимальное	Среднее	Максимальное
Разработанный алгоритм с учетом фрагментированности относительно без учета фрагментированности	0,81	1,16	1,55
Разработанный алгоритм с учетом фрагментированности относительно базового	1,19	1,91	3,13

При использовании алгоритма с оценкой фрагментированности увеличение размера окна приводит к увеличению утилизации максимально на 19,8% и к уменьшению среднего относительного значения времени ожидания задания в очереди до 6,6.

На рисунке 5.28 представлено среднее по всем запускам значение времени, затраченное на поиск решения в зависимости от топологии сети.

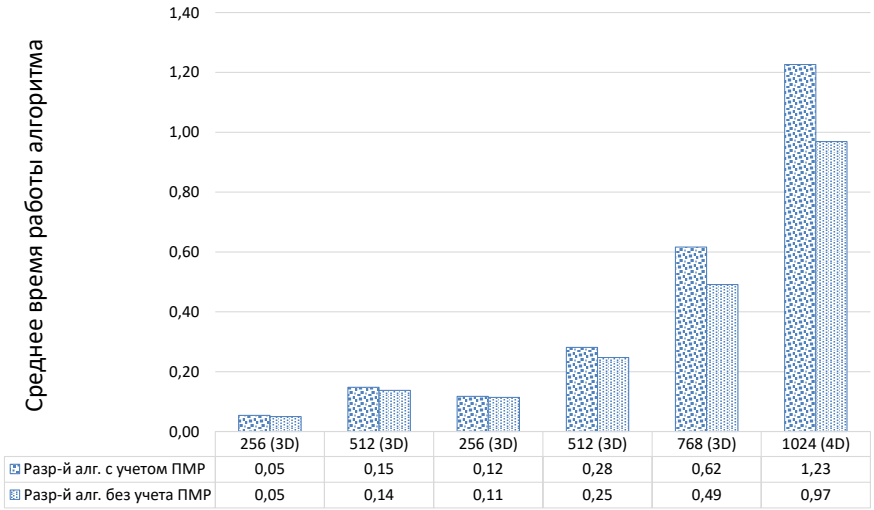


Рисунок 5.28 — Среднее время работы улучшенного алгоритма перебора в зависимости от топологии больших систем от 256 до 1024 узлов. ПМР обозначает оценку фрагментации системы.

Выводы исследования утилизации

В результате исследования утилизации можно сформулировать следующие выводы.

- Эксперимент на реальном суперкомпьютере «Десмос» показывает, что внедрение разработанного алгоритма выбора узлов позволило повысить утилизацию вычислительных ресурсов на 7,65%.
- Сравнение результатов экспериментов на суперкомпьютере «Десмос» и на имитационной модели вычислительной системы позволяет подтвердить адекватность имитационной модели.
- Разработанный алгоритм улучшенного перебора с оценкой фрагментированности дает увеличение утилизации в среднем на 7% относительного базового алгоритма.
- Разработанный алгоритм с применением разработанной оценки фрагментированности позволяет сократить относительное время ожидания задания в очереди в среднем в 1,83 раз относительно базового алгоритма.
- Показано, что время работы разработанного алгоритма выбора узлов удовлетворяет сформулированным ограничениям (не более 10 секунд

по умолчанию в Slurm), таким образом, данный алгоритм может применяться на практике.

5.6 Выводы

В данной главе приводится сравнение разработанных алгоритмов и общее сравнение эффективности работы вычислительных систем при использовании разработанных алгоритмов.

В разделе 5.1 приведено исследование отказоустойчивости сети на основе разработанного алгоритма определения достижимости множества вычислительных узлов. Показано, что в среднем предложенная в диссертационной работе маршрутизация R_{A^*} с возможностью нарушения правила порядка направлений позволяет увеличить общее число сломанных каналов связи до потери достижимости по сравнению с маршрутизацией R_{A^-} без нарушения порядка направлений на 4,9% для 2-х мерных топологий, на 8,2% для 3-х мерных топологий и на 34% для 4-х мерных топологий.

Раздел 5.2 посвящен исследованию алгоритмов приближенного решения задачи построения сбалансированной таблицы маршрутов. При помощи генетического алгоритма возможно получить лучшее по качеству решение, однако время работы генетического алгоритма значительно превышает время работы алгоритма на основе поиска вширь по маршрутному графу. При этом алгоритм на основе поиска вширь позволяет получать приемлемое качество таблицы маршрутов. В связи с этим алгоритм на основе поиска вширь используется в дальнейших исследованиях.

В разделе 5.3 описана разработанная имитационная модель вычислительной системы, с помощью которой в дальнейшем исследованы алгоритмы выбора узлов и исследована эффективность работы суперкомпьютера при использовании разработанных алгоритмов.

В разделе 5.4 показано сравнение алгоритмов выбора достижимого множества вычислительных узлов требуемого размера. Разработанный алгоритм выбора узлов улучшенным перебором многомерных прямоугольников предлагает в 2–12 раз больше вариантов решений, чем базовый алгоритм выбора узлов, который изначально работал на вычислительных кластерах с сетью Ангара.

В разделе 5.5 продемонстрировано исследование характеристик эффективности вычислительных систем с использованием разработанных алгоритмов. Исследование на имитационной модели разработанных алгоритмов по сравнению с базовыми алгоритмами показало, в среднем, улучшение утилизации вычислительных систем на 7%, и улучшение относительного времени ожидания задания в очереди – в 1,83 раза. Эксперимент на реальном суперкомпьютере «Десмос» показывает, что внедрение разработанных алгоритмов выбора узлов и построения таблицы маршрутов позволило повысить утилизацию вычислительных ресурсов на 7,65%.

Заключение

Основные результаты работы заключаются в следующем.

1. Разработан алгоритм построения маршрутного графа для анализа маршрутов в высокоскоростных коммуникационных сетях с топологией «многомерный тор» с произвольным количеством отказавших узлов и каналов связи, а также маршрутизацией, накладывающей ограничение на маршрут сетевого пакета в зависимости от истории его прохождения по сети. Временная сложность алгоритма $O(N^2)$, где N – количество узлов в сети.
2. Разработан алгоритм определения достижимости множества вычислительных узлов сети размера N, временная сложность алгоритма $O(N^2)$. Алгоритм использует возможность программного контроля отсутствия дедлоков в сети Ангара, что позволяет сохранять достижимость сети при большем числе случайно отказавших каналов связи (от 5% до 34%) по сравнению с возложением контроля отсутствия дедлоков на аппаратные возможности сети Ангара.
3. Разработан алгоритм построения таблицы маршрутов для решения задачи балансировки трафика в достижимом множестве узлов размера N, временная сложность алгоритма $O(N^2)$.
4. Разработан алгоритм выбора узлов в сети размера N с учетом её фрагментации, временная сложность алгоритма $O(N^4)$. Алгоритм позволил по сравнению с существовавшими ранее алгоритмами от 2 до 12 раз расширить возможности при выборе множества узлов в сети Ангара в зависимости от потока пользовательских заданий и исследуемой системы.
5. Проведено экспериментальное исследование разработанных алгоритмов, которое по сравнению с базовыми алгоритмами показало, в среднем, улучшение утилизации вычислительных систем на 7%, а относительного времени ожидания задания в очереди – в 1,83 раза.

Список литературы

1. *Mukosey Anatoly, Semenov Alexander, Tretiakov Aleksandr.* Graph Based Routing Algorithm for Torus Topology and Its Evaluation for the Angara Interconnect // *Journal of Parallel and Distributed Computing.* — 2024. — Vol. 183. — P. 104765. — https://doi.org/10.1016/j.jpdc.2023.104765 (дата обращения 02.10.2023). — [WoS: Impact Factor 3.7].

2. *Mukosey Anatoly, Semenov Alexander.* Simulation of Utilization and Energy Saving of the Angara Interconnect // *Lobachevskii Journal of Mathematics.* — 2022. — Vol. 43, no. 4. — Pp. 873–881. — https://doi.org/10.1134/S1995080222070186 (дата обращения 02.10.2023). — [WoS: Impact Factor 0.7].

3. *Mukosey Anatoly, Simonov Alexey, Semenov Alexander.* Extended Routing Table Generation Algorithm for the Angara Interconnect // Supercomputing: 5th Russian Supercomputing Days, RuSCDays 2019. — Vol. 1129. — Springer, 2019. — Pp. 573–583. — https://doi.org/10.1007/978-3-030-36592-9_47 (дата обращения 12.12.2019). — [Scopus: Impact Factor 0.49].

4. *Мукосей А.В., Семенов А.С.* Оптимизация фрагментации при выделении ресурсов для высокопроизводительных вычислительных систем с сетью Ангара // *Вестник ЮУрГУ, серия Вычислительная математика и информатика.* — 2018. — Т. 7, № 2. — С. 50–62. — https://vestnik.susu.ru/cmi/article/view/7507/6253 (дата обращения: 08.10.2018). — [РИНЦ: импакт-фактор 0.524].

5. *Мукосей А.В., Симонов А.С., Семенов А.С.* Оптимизация утилизации при выделении ресурсов для высокопроизводительных вычислительных систем с сетью Ангара // *Вестник ЮУрГУ, серия Вычислительная математика и информатика.* — 2019. — Т. 8, № 1. — С. 5–19. — https://vestnik.susu.ru/cmi/article/view/7691/6847 (дата обращения: 05.12.2019). — [РИНЦ: импакт-фактор 0.524].

6. *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, № 1. —

С. 53–64. — http://num-meth.srcc.msu.ru/zhurnal/tom_2017/pdf/v18r105.pdf (дата обращения: 08.10.2018). — [РИНЦ: импакт-фактор 0.576].

7. *Мукосей А.В., Семенов А.С.* Оптимизация фрагментации при выделении ресурсов для высокопроизводительных вычислительных систем с сетью Ангара. // Параллельные вычислительные технологии (ПаВТ'2018): труды международной научной конференции. — Издательский центр ЮУрГУ, 2018. — С. 310–318. — http://omega.sp.susu.ru/pavt2018/short/021.pdf (дата обращения: 05.12.2019).

8. *Мукосей А.В., Семенов А.С.* Оптимизация утилизации при выделении ресурсов для высокопроизводительных вычислительных систем с сетью Ангара // Суперкомпьютерные дни в России 2018: Труды международной конференции. — М.: Изд-во МГУ, 2018. — С. 831–840. — http://russianscdays.org/files/pdf18/831.pdf (дата обращения: 08.10.2018).

9. *Мукосей А.В., Семенов А.С., Макагон Д.В.* Приближенный алгоритм выбора оптимального подмножества узлов в коммуникационной сети Ангара с отказами // Параллельные вычислительные технологии (ПаВТ'2016): труды международной научной конференции. — Издательский центр ЮУрГУ, 2016. — С. 257–269. — http://omega.sp.susu.ru/PaVT2016/full/053.pdf (дата обращения: 08.10.2018).

10. Early Performance Evaluation of the Hybrid Cluster with Torus Interconnect Aimed at Molecular-Dynamics Simulations / Vladimir Stegailov, Alexander Agarkov, Anatoly Mukosey et al. // International Conference on Parallel Processing and Applied Mathematics. — Springer, 2017. — Pp. 327–336. — https://doi.org/10.1007/978-3-319-78024-5_29 (дата обращения: 08.10.2018). — [Scopus: Impact Factor 0.969].

11. *Мукосей А.В.* Приближенное решение задачи оптимального распределения трафика в коммуникационной сети с топологией «многомерный тор» // Суперкомпьютерные дни в России: Труды международной конференции. — М.: Изд-во МГУ, 2016. — С. 864–874. — http://2016.russianscdays.org/files/pdf16/864.pdf (дата обращения: 08.10.2018).

12. Technology-driven, highly-scalable dragonfly topology / John Kim, William J. Dally, Steve Scott, Dennis Abts // 2008 International Symposium on Computer Architecture / IEEE. — 2008. — Pp. 77–88.

13. *Макагон Д.В., Сыромятников Е.Л.* Сети для суперкомпьютеров // *Открытые системы. СУБД.* — 2011. — № 7. — С. 33–33.

14. *Scott Steven L. et al.* The Cray T3E network: adaptive routing in a high performance 3D torus. — 1996. — http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=7B4A3FAE3AC52B4E4141206064C00923?doi=10.1.1.126.3882&rep=rep1&type=pdf (Accessed 8 October 2023).

15. *Abts Dennis.* The Cray XT4 and Seastar 3-D torus interconnect. — 2010. — https://static.googleusercontent.com/media/research.google.com/ru//pubs/archive/36896.pdf (Accessed 8 October 2023).

16. *Secchi S., Tumeo A., Villa O.* Contention Modeling for Multithreaded Distributed Shared Memory Machines: The Cray XMT // Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing / IEEE Computer Society. — 2011. — Pp. 275–284. — https://doi.org/10.1109/CCGrid.2011.39 (Accessed 9 October 2018).

17. Cray XT4: an early evaluation for petascale scientific simulation / Sadaf R. Alam, Jeffery A. Kuehn, Richard F. Barrett et al. // Proceedings of the 2007 ACM/IEEE Conference on / IEEE. — 2007. — Pp. 1–12. — https://doi.org/10.1145/1362622.1362675 (Accessed 9 October 2023).

18. *Alverson R., Roweth D., Kaplan L.* The Gemini system interconnect // High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on / IEEE. — 2010. — Pp. 83–87. — https://doi.org/10.1109/HOTI.2010.23 (Accessed 9 October 2023).

19. Overview of the Blue Gene/L system architecture / A. Gara, M. Blumrich, D. Chen et al. // *IBM Journal of research and development.* — 2005. — Vol. 49, no. 2.3. — Pp. 195–212. — http://rsim.cs.illinois.edu/arch/qual_papers/systems/19.pdf (Accessed 8 October 2023).

20. Overview of the IBM Blue Gene/P project / Gheorghe Almasi, Sameh Asaad, Ralph E. Bellofatto et al. // *IBM Journal of Research and Development*. — 2008. — Vol. 52, no. 1-2. — Pp. 199–220.

21. *Ajima Yuichiro, Sumimoto Shinji, Shimizu Toshiyuki.* Tofu: a 6D mesh/torus interconnect for exascale computers // *Computer*. — 2009. — Vol. 42, no. 11. — Pp. 36–40.

22. Tofu interconnect 2: system-on-chip integration of high-performance interconnect / Yuichiro Ajima, Tomohiro Inoue, Shinya Hiramoto et al. // International Supercomputing Conference / Springer. — 2014. — Pp. 498–507. — https://doi.org/10.1007/978-3-319-07518-1_35 (Accessed 9 October 2018).

23. The IBM Blue Gene/Q interconnection fabric / Dong Chen, Noel Eisley, Philip Heidelberger et al. // *IEEE Micro*. — 2012. — Vol. 32, no. 1. — Pp. 32–43. — https://www.researchgate.net/publication/220290398_The_IBM_blue_geneQ_interconnection_fabric (Accessed 8 October 2018).

24. The Tofu interconnect D / Yuichiro Ajima, Takahiro Kawashima, Takayuki Okamoto et al. // 2018 IEEE International Conference on Cluster Computing / IEEE. — 2018. — Pp. 646–654.

25. Scalable communication architecture for network-attached accelerators / Sarah Neuwirth, Dirk Frey, Mondrian Nuessle, Ulrich Bruening // High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on / IEEE. — 2015. — Pp. 627–638. — https://doi.org/10.1109/HPCA.2015.7056068 (Accessed https://doi.org/10.1109/HPCA.2015.7056068).

26. Silicon Cube: a Supercomputer Specially Designed for Meteorological Applications / Chaoqun SHA, Pingzhong Yan, Dongming Qin et al.

27. The IBM Blue Gene/Q interconnection network and message unit / Dong Chen, Noel A. Eisley, Philip Heidelberger et al. // SC'11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis / IEEE. — 2011. — Pp. 1–10.

28. Managing System Software for Cray XE and Cray XT Systems. — Cray Document, 2010.

29. Отечественная коммуникационная сеть 3D-тор с поддержкой глобально адресуемой памяти / А.А. Корж, Д.В. Макагон, А.А. Бородин и др. // *Вестник Южно-Уральского государственного университета. Серия: Математическое моделирование и программирование.* — 2010. — № 35 (211). — С. 41–53. — https://cyberleninka.ru/article/n/otechestvennaya-kommunikatsionnaya-set-3d-top-s-podderzhkoy-globalno-adresuemoy-pamyati.pdf (дата обращения: 08.10.2023).

30. Кристалл для Ангары / И.А. Жабин, Д.В. Макагон, А.С. Симонов и др. // *Суперкомпьютеры.* — 2013. — № 4 (16). — С. 46–49. — http://www.nicevt.ru/publications/item/download/73_d0a15c16daed23eb287a092e7bfd9542 (дата обращения: 08.10.2018).

31. *Басалов В.Г., Вялухин В.М.* Адаптивная система маршрутизации для отечественной системы межпроцессорных обменов СМПО-10G // *Вопросы атомной науки и техники. Серия: Математическое моделирование физических процессов.* — 2012. — № 3. — С. 64–70.

32. Опыт разработки коммуникационной сети суперкомпьютера «СКИФ-Аврора» / И.А. Адамович, Климов А. В., Ю.А. Климов и др. // *Программные системы: теория и приложения.* — 2010. — Т. 1, № 3.

33. Паутина: высокоскоростная коммуникационная сеть / Ю.А. Климов, А.Б. Шворин, А.Ю. Хренов и др. // *Программные системы: теория и приложения.* — 2015. — Т. 6, № 1 (24). — http://www.mathnet.ru/links/59720a5514d14a610e33ae83d3fb758c/ps158.pdf (дата обращения: 08.10.2018).

34. *Dally William James, Towles Brian Patrick.* Principles and practices of interconnection networks. — Elsevier, 2004.

35. Adaptive bubble router: a design to improve performance in torus networks / Valentin Puente, Ramón Beivide, José A. Gregorio et al. // Parallel Processing, 1999. Proceedings. 1999 International Conference on / IEEE. — 1999. — Pp. 58–67. — https://doi.org/10.1109/ICPP.1999.797388 (Accessed 9 October 2018).

36. Blue Gene/L torus interconnection network / Narasimha R. Adiga, Matthias A. Blumrich, Dong Chen et al. // *IBM Journal of Research and*

Development. — 2005. — Vol. 49, no. 2.3. — Pp. 265–276. — https://www.cs.ucr.edu/~mart/CS260/IBM-BlueGene-Torus-Journal.pdf (Accessed 8 October 2023).

37. *Glass Christopher J., Ni Lionel M.* The turn model for adaptive routing // *ACM SIGARCH Computer Architecture News.* — 1992. — Vol. 20, no. 2. — Pp. 278–287. — https://www.computer.org/csdl/proceedings/isca/1992/509/00/00753324.pdf (Accessed 10 October 2023).

38. *Chiu Ge-Ming.* The odd-even turn model for adaptive routing // *IEEE Transactions on parallel and distributed systems.* — 2000. — Vol. 11, no. 7. — Pp. 729–738.

39. *Krevat Elie, Castanos José G., Moreira José E.* Job scheduling for the BlueGene/L system // Workshop on Job Scheduling Strategies for Parallel Processing / Springer. — 2002. — Pp. 38–54.

40. An overview of the Blue Gene/L system software organization / George Almási, Ralph Bellofatto, José Brunheroto et al. // European Conference on Parallel Processing / Springer. — 2003. — Pp. 543–555.

41. *Vishnu Abhinav, ten Bruggencate Monika, Olson Ryan.* Evaluating the potential of Cray Gemini interconnect for PGAS communication runtime systems // 2011 IEEE 19th Annual Symposium on High Performance Interconnects / IEEE. — 2011. — Pp. 70–77.

42. Measuring Congestion in High-Performance Datacenter Interconnects / Saurabh Jha, Archit Patke, Jim Brandt et al. // 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20). — 2020. — Pp. 37–57. — https://www.usenix.org/system/files/nsdi20-paper-jha.pdf (Accessed 8 October 2023).

43. *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.

44. *Qiao Wenjian, Ni Lionel M.* Adaptive routing in irregular networks using cut-through switches // Parallel Processing, 1996. Vol. 3. Software.,

Proceedings of the 1996 International Conference on / IEEE. — Vol. 1. — 1996. — Pp. 52–60. — https://pdfs.semanticscholar.org/c389/ca953ba3af43ea7781a036005b6d5d968ad9.pdf (Accessed 10 October 2023).

45. *Hoefler Torsten, Schneider Timo, Lumsdaine Andrew.* Optimized routing for large-scale InfiniBand networks // High Performance Interconnects, 2009. HOTI 2009. 17th IEEE Symposium on / IEEE. — 2009. — Pp. 103–111. — https://spcl.inf.ethz.ch/Publications/.pdf/hoefler-ib-routing.pdf (Accessed 10 October 2023).

46. *Domke Jens, Hoefler Torsten, Nagel Wolfgang E.* Deadlock-Free Oblivious Routing for Arbitrary Topologies // 2011 IEEE International Parallel & Distributed Processing Symposium. — IEEE, 2011.

47. *Dally William J., Seitz Charles L.* Deadlock-free message routing in multiprocessor interconnection networks // *IEEE Transactions on computers.* — 1987. — Vol. 100, no. 5. — Pp. 547–553. — https://authors.library.caltech.edu/26930/1/5231-TR-86.pdf (Accessed 9 October 2023).

48. *Schwiebert Loren.* Deadlock-free oblivious wormhole routing with cyclic dependencies // *IEEE Transactions on Computers.* — 2001. — Vol. 50, no. 9. — Pp. 865–876.

49. *Domke Jens, Hoefler Torsten, Matsuoka Satoshi.* Routing on the dependency graph: a new approach to deadlock-free high-performance routing // Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing / ACM. — 2016. — Pp. 3–14. — https://htor.inf.ethz.ch/publications/img/nue_deadlock_free_routing_hpdc16.pdf (Accessed 8 October 2023).

50. *Bulut Eyuphan, Geyik Sahin Cem, Szymanski Boleslaw K.* Conditional shortest path routing in delay tolerant networks // World of Wireless Mobile and Multimedia Networks (WoWMoM), 2010 IEEE international symposium on a / IEEE. — 2010. — Pp. 1–6. — https://doi.org/10.1109/WOWMOM.2010.5534960 (Accessed 9 October 2018).

51. *Fall Kevin.* A delay-tolerant network architecture for challenged internets // Proceedings of the 2003 conference on Applications, technologies, architectures,

and protocols for computer communications / ACM. — 2003. — Pp. 27–34. — https://doi.org/10.1145/863955.863960 (Accessed 9 October 2018).

52. A survey and evaluation of topology-agnostic deterministic routing algorithms / J.C. Sancho, T. Rokicki, M. Koibuchi et al. // *IEEE Transactions on Parallel and Distributed Systems*. — 2012. — no. 3. — Pp. 405–425. — https://doi.org/10.1109/TPDS.2011.190 (Accessed 9 October 2018).

53. *Sancho José Carlos, Robles Antonio, Duato José*. A new methodology to compute deadlock-free routing tables for irregular networks // International Workshop on Communication, Architecture, and Applications for Network-Based Parallel Computing / Springer. — 2000. — Pp. 45–60. — https://pdfs.semanticscholar.org/0f88/cf69374a4e6c1fdf9c44747c43414ba65514.pdf?_ga=2.244172037.1866382581.1536270552-772162659.1536270552 (Accessed 10 October 2023).

54. Autonet: a high-speed, self-configuring local area network using point-to-point links / Michael D. Schroeder, Andrew D. Birrell, Michael Burrows et al. // *IEEE Journal on Selected Areas in Communications*. — 1991. — Vol. 9, no. 8. — Pp. 1318–1335.

55. *Sancho José Carlos, Robles Antonio*. Improving the up*/down* routing scheme for networks of workstations // European Conference on Parallel Processing / Springer. — 2000. — Pp. 882–889. — https://doi.org/10.1007/3-540-44520-X_123 (Accessed 9 October 2018).

56. *Skeie Tor, Lysne Olav, Theiss Ingebjørg*. Layered shortest path (LASH) routing in irregular system area networks // ipdps / IEEE. — 2002. — P. 0162. — http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.18.3962&rep=rep1&type=pdf (Accessed 10 October 2023).

57. *Domke Jens, Hoefler Torsten, Nagel Wolfgang E*. Deadlock-free oblivious routing for arbitrary topologies // 2011 IEEE International Parallel & Distributed Processing Symposium / IEEE. — 2011. — Pp. 616–627.

58. Balanced Dimension-Order Routing for k-ary n-cubes / Jose Miguel Montañana, Michihiro Koibuchi, Hiroki Matsutani, Hideharu Amano // Parallel Processing Workshops, 2009. ICPPW'09. International Conference on / IEEE.

— 2009. — Pp. 499–506. — https://doi.org/10.1109/ICPPW.2009.64 (Accessed 9 October 2018).

59. *Tyagi Shivam.* Extended balanced dimension ordered routing algorithm for 3D-networks // International Conference on Parallel Processing Workshops. — 2009. — Pp. 499–506.

60. The forwarding index of communication networks / Fan Chung, E. Coffman, Martin Reiman, Burton Simon // *IEEE Transactions on Information theory.* — 1987. — Vol. 33, no. 2. — Pp. 224–232.

61. *Towles Brian, Dally William J.* Worst-case traffic for oblivious routing functions // Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures / ACM. — 2002. — Pp. 1–8. — http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=4CC2CBDF375C12A7A3EBE45F2377B023?doi=10.1.1.15.4186&rep=rep1&type=pdf (Accessed 10 October 2023).

62. *Towles Brian, Dally William J., Boyd Stephen.* Throughput-centric routing algorithm design // Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures / ACM. — 2003. — Pp. 200–209.

63. *Hoefler Torsten, Schneider Timo, Lumsdaine Andrew.* Multistage switches are not crossbars: effects of static routing in high-performance networks // 2008 IEEE International Conference on Cluster Computing / IEEE. — 2008. — Pp. 116–125.

64. *Heydemann Marie-Claude, Meyer Jean Claude, Sotteau Dominique.* On forwarding indices of networks // *Discrete Applied Mathematics.* — 1989. — Vol. 23, no. 2. — Pp. 103–123.

65. *Saad Rachid.* Complexity of the forwarding index problem // *SIAM Journal on Discrete Mathematics.* — 1993. — Vol. 6, no. 3. — Pp. 418–427.

66. Application-aware deadlock-free oblivious routing / Michel Kinsy, Myong Hyon Cho, Tina Wen et al. // Proceedings of the 36th annual international symposium on computer architecture. — 2009. — Pp. 208–219. — https://people.csail.mit.edu/devadas/pubs/bsor-isca.pdf (Accessed 10 October 2023).

67. *Abdel-Gawad Ahmed H., Thottethodi Mithuna.* TransCom: Transforming stream communication for load balance and efficiency in networks-on-chip // Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture. — 2011. — Pp. 237–247.

68. *Abdel-Gawad Ahmed H., Thottethodi Mithuna.* Scalable, global, optimal-bandwidth, application-specific routing // 2016 IEEE 24th Annual Symposium on High-Performance Interconnects (HOTI) / IEEE. — 2016. — Pp. 9–18.

69. *Johnson Donald B.* A Note on Dijkstra's Shortest Path Algorithm // *Journal of the ACM.* — 1973. — Vol. 20, no. 3. — Pp. 385–388.

70. *Domke Jens, Hoefler Torsten.* Scheduling-aware routing for supercomputers // Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis / IEEE Press. — 2016. — P. 13. — https://spcl.inf.ethz.ch/Publications/.pdf/domke-hoefler-sched-aware-routing.pdf (Accessed 10 October 2023).

71. *Domke Jens, Hoefler Torsten, Matsuoka Satoshi.* Routing on the Dependency Graph: a New Approach to Deadlock-Free High-Performance Routing // Proceedings of the 25th ACM International Symposium on High-Performance Parallel and Distributed Computing. — HPDC '16. — New York, NY, USA: ACM, 2016. — Pp. 3–14.

72. Multipath Load Balancing for M× N Communication Patterns on the Blue Gene/Q Supercomputer Interconnection Network / Huy Bui, Robert Jacob, Preeti Malakar et al. // 2015 IEEE International Conference on Cluster Computing / IEEE. — 2015. — Pp. 833–840.

73. *Yen Jin Y.* An algorithm for finding shortest routes from all source nodes to a given destination in general networks // *Quarterly of Applied Mathematics.* — 1970. — Vol. 27, no. 4. — Pp. 526–530.

74. Application-aware deadlock-free oblivious routing based on extended turn-model / Ali Shafiee, Mahdy Zolghadr, Mohammad Arjomand, Hamid Sarbazi-Azad // 2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD) / IEEE. — 2011. — Pp. 213–218.

75. Balancing job performance with system performance via locality-aware scheduling on torus-connected systems / Xu Yang, Zhou Zhou, Wei Tang et al. // Cluster Computing (CLUSTER), 2014 IEEE International Conference on / IEEE. — 2014. — Pp. 140–148. — https://doi.org/10.1109/CLUSTER.2014.6968751 (Accessed 9 October 2018).

76. Reducing fragmentation on torus-connected supercomputers / Wei Tang, Zhiling Lan, Narayan Desai et al. // Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International / IEEE. — 2011. — Pp. 828–839. — https://doi.org/10.1109/IPDPS.2011.82 (Accessed 9 October 2018).

77. *Lakner Gary, Knudson Brant et al.* IBM system Blue Gene solution: Blue Gene/Q system administration. — IBM Redbooks, 2013. — P. 282. — http://www.redbooks.ibm.com/redbooks/pdfs/sg247869.pdf (Accessed 8 October 2023).

78. Resource allocation and utilization in the Blue Gene/L supercomputer / Yariv Aridor, Tamar Domany, Oleg Goldshmidt et al. // *IBM Journal of Research and Development*. — 2005. — Vol. 49, no. 2.3. — Pp. 425–436.

79. Improving batch scheduling on Blue Gene/Q by relaxing 5D torus network allocation constraints / Zhou Zhou, Xu Yang, Zhiling Lan et al. // 2015 IEEE International Parallel and Distributed Processing Symposium / IEEE. — 2015. — Pp. 439–448.

80. *Tuncer Ozan, Leung Vitus J., Coskun Ayse K.* Pacmap: Topology mapping of unstructured communication patterns onto non-contiguous allocations // Proceedings of the 29th ACM on International Conference on Supercomputing. — 2015. — Pp. 37–46. — https://dl.acm.org/doi/pdf/10.1145/2751205.2751225 (Accessed 8 April 2023).

81. *Karypis George, Kumar Vipin.* Multilevel k-way partitioning scheme for irregular graphs // *Journal of Parallel and Distributed Computing*. — 1998. — Vol. 48, no. 1. — Pp. 96–129.

82. *Qiao Wenjian, Ni Lionel M.* Efficient processor allocation for 3D tori // Proceedings of 9th International Parallel Processing Symposium. — IEEE Comput. Soc. Press, 1995. — Pp. 466–471.

83. *Choo Hyunseung, Yoo Seong-Moo, Youn Hee Yong.* Processor scheduling and allocation for 3D torus multicomputer systems // *IEEE Transactions on Parallel and Distributed Systems.* — 2000. — Vol. 11, no. 5. — Pp. 475–484. — https://doi.org/10.1109/71.852400 (Accessed 9 October 2018).

84. *Schwiegelshohn Uwe, Yahyapour Ramin.* Analysis of first-come-first-serve parallel job scheduling // SODA / Citeseer. — Vol. 98. — 1998. — Pp. 629–638. — http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.72.3616&rep=rep1&type=pdf (Accessed 8 October 2018).

85. *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. — http://omega.sp.susu.ru/books/conference/PaVT2010/full/059.pdf (дата обращения: 08.10.2018).

86. *Ababneh Ismail, Bani-Mohammad Saad.* A new window-based job scheduling scheme for 2D mesh multicomputers // *Simulation Modelling Practice and Theory.* — 2011. — Vol. 19, no. 1. — Pp. 482–493. — https://doi.org/10.1016/j.simpat.2010.08.007 (Accessed 10 October 2023).

87. *Mu'alem Ahuva W., Feitelson Dror 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. — Pp. 529–543. — https://pdfs.semanticscholar.org/895f/74d91280f865e5d7e2187dd7c5c6913eea25.pdf (Accessed 8 October 2018).

88. *Henderson Robert L.* Job scheduling under the portable batch system // Workshop on Job Scheduling Strategies for Parallel Processing / Springer. — 1995. — Pp. 279–294. — http://dx.doi.org/10.1007/3-540-60153-8_34 (Accessed 9 October 2018).

89. *Staples Garrick.* Torque resource manager // Proceedings of the 2006 ACM/IEEE conference on Supercomputing / ACM. — 2006. — P. 8. — https://doi.org/10.1145/1188455.1188464 (Accessed 9 October 2018).

90. *Jackson David, Snell Quinn, Clement Mark.* Core algorithms of the Maui scheduler // Workshop on Job Scheduling Strategies for Parallel Processing / Springer. — 2001. — Pp. 87–102. — http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.325.5028&rep=rep1&type=pdf (Accessed 8 October 2018).

91. *Gentzsch Wolfgang.* Sun grid engine: Towards creating a compute power grid // Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium on / IEEE. — 2001. — Pp. 35–36. — https://doi.org/10.1109/CCGRID.2001.923173 (Accessed 9 October 2018).

92. Модернизация СУПЗ МВС-1000 / А.В. Баранов, С.В. Смирнов, М.Ю. Храмцов, С.В. Шарф // *Материалы Всероссийской научной конференции «Научный сервис в сети Интернет»*. — 2008. — С. 226–227.

93. Slurm Workload Manager. — https://slurm.schedmd.com/overview.html (Accessed 8 October 2023).

94. Routing in InfiniBand torus network topologies / José Carlos Sancho, Antonio Robles, Pedro Lopez et al. // 2003 International Conference on Parallel Processing, 2003. Proceedings. / IEEE. — 2003. — Pp. 509–518.

95. Hybrid supercomputer Desmos with torus Angara interconnect: performance and efficiency optimisation / E. Dlinnova, G. Smirnov, V. Stegailov et al. // Parallel Computational Technologies: 12th International Conference, PCT 2018, Rostov-on-Don, Russia, April 2–6, 2018, Revised Selected Papers 12 / Springer. — 2018. — Pp. 77–91. — https://www.researchgate.net/profile/Nikolay-Kondratyuk/publication/327229308_Hybrid_Supercomputer_Desmos_with_Torus_Angara_Interconnect_Efficiency_Analysis_and_Optimization_12th_International_Conference_PCT_2018_Rostov-on-Don_Russia_April_2-6_2018_Revised_Selected_Papers/links/5b97bbc4299bf14ad4cc7edb/Hybrid-Supercomputer-Desmos-with-Torus-Angara-Interconnect-Efficiency-Analysis-and-Optimization-12th-International-Conference-PCT-2018-Rostov-on-Don-Russia-April-2-6-2018-Revised-Selected-Papers.pdf (Accessed 8 October 2023).

96. *Баранов А.В., Киселёв Е.А., Ляховец Д.С.* Квазипланировщик для использования простаивающих вычислительных модулей многопроцессорной вычислительной системы под управлением СУППЗ // *Вестник Южно-Уральского государственного университета. Серия: Вычислительная математика и информатика.* — 2014. — Т. 3, № 4. — https://cyberleninka.ru/article/n/kvaziplanirovschik-dlya-ispolzovaniya-prostaivayuschih-vychislitelnyh-moduley-mnogoprotsessornoy-vychislitelnoy-sistemy-pod.pdf (дата обращения: 08.10.2023).

97. *Hoefler Torsten, Snir Marc.* Generic topology mapping strategies for large-scale parallel architectures // Proceedings of the international conference on Supercomputing. — 2011. — Pp. 75–84. — http://ww.w.unixer.de/publications/img/hoefler_snir_topology_mapping.pdf (Accessed 8 April 2023).

98. Первое поколение высокоскоростной коммуникационной сети «Ангара» / И.А. Жабин, Д.В. Макагон, Д.А. Поляков и др. // *Наукоемкие технологии.* — 2014. — Т. 15, № 1. — С. 021–027.

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

100. Отечественная коммуникационная сеть 3D-тор с поддержкой глобально адресуемой памяти для суперкомпьютеров транспетафлопсного уровня производительности / А.А. Корж, Д.В. Макагон, И.А. Жабин, Е.Л. Сыромятников // Параллельные вычислительные технологии (ПаВТ'2010): Труды международной конференции (Уфа, 29 марта 2 апреля 2010 г.). Челябинск: Издательский центр ЮУрГУ. — 2010. — С. 227–237. — http://omega.sp.susu.ru/books/conference/PaVT2010/full/134.pdf (дата обращения: 08.10.2018).

101. *Кагиров Р.Р.* Многомерная задача о рюкзаке: новые методы решения // *Вестник Сибирского государственного аэрокосмического университета им. академика М.Ф. Решетнева.* — 2007. — № 3. — https://cyberleninka.ru/article/n/mnogomernaya-zadacha-o-ryukzake-novye-metody-resheniya.pdf (дата обращения: 08.10.2023).

102. *Gonçalves José Fernando, Resende Mauricio G.C.* A parallel multi-population biased random-key genetic algorithm for a container loading problem // *Computers and Operations Research.* — 2012. — Vol. 39, no. 2. — Pp. 179–190. — https://doi.org/10.1016/j.cor.2011.03.009 (Accessed 9 October 2018).

103. *Аладышев О.С., Киселёв Е.А.* Алгоритм эффективного размещения программ на ресурсах многопроцессорных вычислительных систем // *Программные продукты и системы.* — 2012. — № 4. — С. 18–25. — https://cyberleninka.ru/article/n/algoritm-effektivnogo-razmescheniya-programm-na-resursah-mnogoprotsessornyh-vychislitelnyh-sistem.pdf (дата обращения: 08.10.2018).

104. *Полежаев П.Н.* Симулятор вычислительного кластера и его управляющей системы, используемый для исследования алгоритмов планирования задач // *Вестник Южно-Уральского государственного университета. Серия: Математическое моделирование и программирование.* — 2010. — № 35 (211). — С. 79–90. — https://cyberleninka.ru/article/n/simulyator-vychislitelnogo-klastera-i-ego-upravlyayuschey-sistemy-ispolzuemyy-dlya-issledovaniya-algoritmov-planirovaniya-zadach.pdf (дата обращения: 08.10.2023).

105. *Sharma D. Das, Pradhan Dhiraj K.* A fast and efficient strategy for submesh allocation in mesh-connected parallel computers // Parallel and Distributed Processing, 1993. Proceedings of the Fifth IEEE Symposium on / IEEE. — 1993. — Pp. 682–689. — https://doi.org/10.1109/SPDP.1993.395466 (Accessed 9 October 2018).

106. Task scheduling in distributed computing systems with a genetic algorithm / Sung-Ho Woo, Sung-Bong Yang, Shin-Dug Kim, Tack-Don Han // High Performance Computing on the Information Superhighway, 1997. HPC Asia'97 / IEEE. — 1997. — Pp. 301–305. — https://doi.org/10.1109/HPC.1997.592164 (Accessed 9 October 2018).

107. *Valiant Leslie G.* A Scheme for Fast Parallel Communication // *SIAM J. Comput.* — 1982. — Vol. 11, no. 2. — Pp. 350–361. — https://ldhulipala.github.io/readings/ValiantPermutationRouting.pdf (Accessed 8 October 2018).

108. *Nesson Ted, Johnsson S. Lennart.* ROMM Routing on Mesh and Torus Networks // Proceedings of the Seventh Annual ACM Symposium on Parallel Algorithms and Architectures. — SPAA '95. — New York, NY, USA: ACM, 1995. — Pp. 275–287. — http://doi.acm.org/10.1145/215399.215455 (Accessed 9 October 2018).

109. Locality-preserving Randomized Oblivious Routing on Torus Networks / Arjun Singh, William J. Dally, Brian Towles, Amit K. Gupta // Proceedings of the Fourteenth Annual ACM Symposium on Parallel Algorithms and Architectures. — SPAA '02. — New York, NY, USA: ACM, 2002. — Pp. 9–13.

110. *Linder D.H., Harden J.C.* An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes // *IEEE Transactions on Computers.* — 1991. — Vol. 40, no. 1. — Pp. 2–12.

111. *Abdel-Gawad Ahmed H., Thottethodi Mithuna, Bhatele Abhinav.* RAHTM: routing algorithm aware hierarchical task mapping // Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis / IEEE Press. — 2014. — Pp. 325–335. — https://engineering.purdue.edu/~mithuna/rahtm.pdf (Accessed 10 October 2023).

112. *Duato Jose.* On the design of deadlock-free adaptive routing algorithms for multicomputers: design methodologies // Parle'91 Parallel Architectures and Languages Europe / Springer. — 1991. — Pp. 390–405.