ФЕДЕРАЛЬНОЕ АГЕНСТВО СВЯЗИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФГОБУ ВПО “СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ”

На правах рукописи

Пазников Алексей Александрович

# АЛГОРИТМЫ ОРГАНИЗАЦИИ ФУНКЦИОНИРОВАНИЯ МУЛЬТИКЛАСТЕРНЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ С ИЕРАРХИЧЕСКОЙ СТРУКТУРОЙ

05.13.15 – Вычислительные машины, комплексы и компьютерные сети

ДИССЕРТАЦИЯ
на соискание ученой степени
кандидата технических наук

Научный руководитель
доктор технических наук,
профессор,
член-корреспондент РАН,
заслуженный деятель науки РФ
Хорошевский Виктор Гаврилович

Научный консультант
кандидат технических наук
Курносов Михаил Георгиевич

Новосибирск – 2013

# Содержание

# Список основных сокращений

ВС – вычислительная система.

ИФП СО РАН – Федеральное государственное бюджетное учреждение науки Институт физики полупроводников им. А.В. Ржанова Сибирского отделения Российской академии наук.

ПО – программное обеспечение.

СО АН СССР – Сибирское отделение Академии наук Союза Советских Социалистических Республик.

СУР – система управления ресурсами.

ФГОБУ ВПО “СибГУТИ” – Федеральное государственное образовательное бюджетное учреждение высшего профессионального образования “Сибирский государственный университет телекоммуникаций и информатики”.

ЦПВТ – Центр параллельных вычислительных технологий.

ЭМ – элементарная машина.

MPI – Message Passing Interface.

PGAS – Partitioned Global Address Space.

# Введение

**Актуальность.** В настоящее время возрастающая потребность в высокопроизводительных средствах обработки информации при решении сложных задач науки и техники привела к необходимости создания распределённых вычислительных систем (ВС) [1–3], характеризующихся массовым параллелизмом. В архитектурном плане они представляют собой множества элементарных машин (ЭМ), оснащённых средствами коммуникаций и внешними устройствами. К началу XXI в. получили широкое распространение пространственно-распределённые вычислительные системы – макроколлективы рассредоточенных вычислительных средств (подсистем), взаимодействующих через локальные и глобальные сети связи (включая сеть Internet). Подсистема такой ВС может быть представлена ЭВМ, вычислительным кластером или отдельной проприетарной ВС с массовым параллелизмом. К пространственно-распределённым относятся мультикластерные вычислительные и GRID-системы.

Одним из основных режимов функционирования пространственно-распределённых ВС является мультипрограммный режим обслуживания потоков параллельных задач. В этом режиме в систему (в распределённую очередь) поступает поток задач. Для решения каждой задачи требуется выделять элементарные машины с одной или нескольких подсистем с целью оптимизации заданных показателей эффективности функционирования ВС.

Одним из таких показателей является время обслуживания задачи, которое включает время доставки входных и выходных данных задачи до подсистем, время ожидания в локальных очередях и время выполнения программы на ЭМ.

Актуальной является разработка моделей, методов и программного обеспечения организации функционирования пространственно-распределённых ВС. В моделях и алгоритмах должны учитываться архитектурные свойства современных ВС: большемасштабность, мультиархитектурная организация (наличие

SMP, NUMA-узлов и специализированных ускорителей) и иерархическая структура коммуникационной среды.

После того, как сформирована подсистема ЭМ, необходимо оптимально вложить задачу в неё: распределить ветви по ЭМ так, чтобы минимизировать накладные расходы на межмашинные обмены информацией. Проблема вложения (Task mapping, task allocation, task assignment) в недостаточной степени проработана для пространственно-распределённых ВС, поэтому востребованы алгоритмы оптимизации вложения параллельных программ в мультикластерные и GRID-системы.

Пространственно-распределённые ВС комплектуются из неабсолютно надёжных вычислительных ресурсов (вычислительных узлов, сетевых коммутаторов, процессорных ядер и др.), поэтому немаловажной задачей является разработка средств (математических моделей, методов и программного обеспечения) организации их живучего функционирования.

Отечественные и зарубежные исследования в области распределённых ВС активно ведутся со второй половины XX столетия. Ряд фундаментальных работ посвящен проблемам создания и эксплуатации высокопроизводительных вычислительных средств: проведены исследования по теории функционирования и построению оптимальных (макро)структур ВС, проработаны многие аспекты создания программного обеспечения, исследован широкий круг задач, допускающих эффективную реализацию на распределённых ВС. Построены отечественные вычислительные системы: “Минск-222”, СУММА, МИНИМАКС, МИКРОС, МВС, Эльбрус и др. Создана первая в мире пространственно-распределённая ВС – система АСТРА [4].

Фундаментальный вклад в теорию и практику вычислительных систем и параллельных вычислительных технологий внесли выдающиеся учёные, среди которых Е. П. Балашов, В. Б. Бетелин, В. С. Бурцев, В. В. Васильев, В. В. Воеводин, В. М. Глушков, В. Ф. Евдокимов, Э. В. Евреинов, А. В. Забродин, В. П. Иванников, М. Б. Игнатьев, А. В. Каляев, И. А. Каляев, Л. Н. Коро-

лев, В. Г. Лазарев, С. А. Лебедев, В. К. Левин, Г. И. Марчук, В. А. Мельников, Ю. И. Митропольский, Д. А. Поспелов, И. В. Прангишвили, Д. В. Пузанков, Г. Е. Пухов, А. Д. Рычков, Г. Г. Рябов, А. А. Самарский, В. Б. Смолов, А. Н. Томилин, Я. А. Хетагуров, В. Г. Хорошевский, Б. Н. Четверушкин, Ю. И. Шокин, Н. Н. Яненко, P. Balaji, R. Buyya, S. Cray, J. Dongarra, M. Flynn, I. Foster, A. Gara, D. Grice, W. Gropp, D. Hillis, C. Kesselman, D. L. Slotnick, R. Thakur и др. На основе полученных результатов был создан инструментарий организации функционирования распределённых ВС [1–70].

При решении проблем оптимизации функционирования ВС в мультипрограммных режимах большую роль сыграли фундаментальные работы по исследованию операций и оптимальному управлению выдающихся ученых: В. Л. Береснева, Э. Х. Гимади, В. Т. Дементьева, С. В. Емельянова, Ю. И. Журавлева, А. А. Корбут, С. К. Коровина, Ю. С. Попкова, К. В. Рудакова, D. P. Agrawal, R. Baraglia, S. H. Bokhari, P. Bouvry, A. Gara, G. Karypis, B. W. Kernighan, V. Kumar, S. Lin, R. Perego, K. Steiglitz и др.

В диссертации предложены децентрализованные алгоритмы диспетчеризации параллельных программ в мультикластерных ВС с иерархической структурой и алгоритмы оптимизации вложения в них параллельных программ. Полученные результаты легли в основу инструментария организации функционирования мультикластерных ВС.

**Цели и задачи диссертационной работы.** Цель диссертации заключается в разработке и исследовании алгоритмов и программных средств организации функционирования мультикластерных ВС с иерархической структурой.

В соответствии с целью определены следующие задачи исследования.

1. Анализ архитектурных свойств современных пространственно-распределённых мультикластерных вычислительных и GRID-систем, методов диспетчеризации и вложения в них параллельных программ.

2. Разработка алгоритмов децентрализованной диспетчеризации в мультикластерных ВС параллельных программ с целью минимизации времени их обслуживания.
3. Создание программного инструментария децентрализованной диспетчеризации параллельных программ в мультикластерных ВС.
4. Построение алгоритмов оптимизации вложения в иерархические пространственно-распределённые ВС параллельных программ с целью минимизации времени их выполнения.
5. Реализация программного инструментария субоптимального вложения параллельных MPI-программ в мультикластерные ВС.
6. Разработка средств мониторинга производительности каналов связи и загрузки подсистем мультикластерных ВС.

**Методы исследования.** Для достижения цели и решения поставленных задач применялись методы теории функционирования распределённых вычислительных систем, теории множеств, теории графов, теории алгоритмов и математический аппарат исследования операций. Экспериментальные исследования проводились путём моделирования на пространственно-распределённой мультикластерной вычислительной системе.

**Научная новизна работы.** В диссертационной работе созданы и исследованы алгоритмы организации функционирования мультикластерных ВС с иерархической структурой.

1. Предложено семейство алгоритмов децентрализованной диспетчеризации параллельных программ. Алгоритмы учитывают переменный характер загрузки ресурсов и каналов связи пространственно-распределённых ВС и позволяют обеспечить живучее обслуживание потоков параллельных программ.
2. На основе методов разбиения графов на непересекающиеся подмножества предложены эвристические алгоритмы вложения параллельных программ в мультикластерные ВС. Алгоритмы учитывают все уровни иерархиче-

ской структуры ВС, что позволяет сократить время выполнения информационных обменов в параллельных программах.

3. Выработаны рекомендации по формированию структур логических связей децентрализованных диспетчеров мультикластерных ВС. Создан эвристический алгоритм поиска субоптимальных структур локальных окрестностей диспетчеров, минимизирующий функцию штрафа при обслуживании потоков параллельных программ.

**Практическая ценность работы.** Разработанные в диссертации модели и алгоритмы реализованы в компонентах системного программного обеспечения мультикластерных и GRID-систем.

Предложенные алгоритмы диспетчеризации легли в основу пакета GBroker децентрализованной диспетчеризации параллельных задач в мультикластерных ВС. Применение пакета GBroker позволяет организовать живучее обслуживание потоков параллельных программ. Алгоритмы диспетчеризации характеризуются незначительной вычислительной трудоёмкостью, что обеспечивает их применимость в большемасштабных ВС.

Разработаны программные средства NetMon и DCSMon мониторинга производительности каналов связи и состояния вычислительных ресурсов мультикластерных ВС.

На основе эвристических алгоритмов вложения создан пакет MPIGridMap оптимизации вложения MPI-программ, позволяющий сократить время их выполнения в мультикластерных ВС. Пакет включает средства формирования информационных графов программ и оптимизации их вложения в мультикластерные ВС.

Компоненты программного обеспечения внедрены в действующую пространственно-распределённую мультикластерную ВС Центра параллельных вычислительных технологий ФГОБУ ВПО “СибГУТИ” (ЦПВТ ФГОБУ ВПО “СибГУТИ”) и Лаборатории вычислительных систем Института физики полупроводников им. А.В. Ржанова СО РАН (ИФП СО РАН).

**Реализация и внедрение результатов работы.** Результаты диссертационного исследования нашли применение в работах по созданию и развитию пространственно-распределённой мультикластерной ВС ЦПВТ ФГОБУ ВПО "СибГУТИ" и Лаборатории ВС ИФП СО РАН.

Исследования выполнялись в рамках федеральной целевой программы "Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2007-2013 годы" (госконтракт 07.514.11.4015 "Сверхмасштабируемые средства вложения и отказоустойчивого выполнения параллельных программ для вычислительных систем экзафлопсного уровня производительности") и при выполнении работ по междисциплинарному интеграционному проекту СО РАН № 113 "Методы параллельной обработки данных и моделирование на распределенных вычислительных системах". Работа поддержана грантами Российского фонда фундаментальных исследований № 12-07-31016 (научный руководитель – Пазников А.А.), 12-07-00145, 11-07-00105, 09-07-00095, 08-07-00018, грантами Президента РФ по поддержке ведущих научных школ № НШ-2175.2012.9, НШ-5176.2010.9, НШ-2121.2008.9 и грантом по Программе "У.М.Н.И.К." Фонда содействия развитию малых форм предприятий в научно-технической сфере.

Результаты диссертации внедрены в учебный процесс. Они используются при чтении курсов лекций на Кафедре вычислительных систем ФГОБУ ВПО "СибГУТИ" по дисциплинам "Теория функционирования распределённых вычислительных систем" и "Высокопроизводительные вычислительные системы".

Внедрение результатов диссертационных исследований подтверждено соответствующими актами.

**Достоверность** полученных результатов подтверждается проведёнными экспериментами и моделированием, согласованностью с данными, имеющимися в отечественной и зарубежной литературе, а также экспертизами работы, прошедшими при получении грантов.

**Апробация работы.** Основные результаты работы докладывались и обсуждались на международных, всероссийских и региональных научных конференциях, в том числе:

– Международной конференции “International Conference on Ubiquitous Information Management and Communication (ICUIMC)” (г. Кота-Кинабалу, Малайзия, 2013);

– Международной конференции “Математические и информационные технологии (MIT)” (г. Врнячка Баня, г. Будва, Сербия, Черногория, 2011);

– Международных научных студенческих конференциях “Студент и научно-технический прогресс (МНСК)” (г. Новосибирск, 2008, 2009, 2011, 2012);

– Всероссийской научно-технической конференции “Суперкомпьютерные технологии” (с. Дивноморское Геленджикского района, 2012);

– Российской конференции с международным участием “Распределенные информационные и вычислительные ресурсы (DICR)” (г. Новосибирск, 2010);

– Российской научной конференции с участием зарубежных учёных “Моделирование систем информатики” (г. Новосибирск, 2011);

– Всероссийской конференции молодых ученых по математическому моделированию и информационным технологиям (г. Новосибирск, 2011);

– Российских конференциях “Новые информационные технологии в исследовании сложных структур (ICAM)”, (г. Томск, 2010, Алтайский Край, 2012);

– Российских научно-технических конференциях “Информатика и проблемы телекоммуникаций” (г. Новосибирск, 2008, 2009, 2010, 2011);

– Российской научно-технической конференции “Обработка информационных сигналов и математическое моделирование” (г. Новосибирск, 2012);

– Всероссийских научно-технических конференциях “Научное и технические обеспечение исследований и освоения шельфа Северного Ледовитого океана” (г. Новосибирск, 2010, 2012);

– Всероссийской научной конференции молодых учёных “Наука. Технологии. Инновации” (г. Новосибирск, 2011);

– Сибирской конференции по параллельным и высокопроизводительным вычислениям (г. Томск, 2009).

**Публикации.** По теме диссертации опубликовано 30 работ: 5 – в изданиях из списка ВАК, 2 свидетельства о государственной регистрации программы для ЭВМ, 23 – в материалах всероссийских и международных конференций. Результаты исследований отражены в отчётах по грантам и НИР.

**Личный вклад.** Содержание диссертационной работы и основные результаты, выносимые на защиту, отражают персональный вклад автора в опубликованные работы. Подготовка к публикации полученных результатов проводилась совместно с соавторами, при этом вклад диссертанта был определяющим. Все представленные в диссертации результаты получены лично автором.

**Основные результаты диссертации, выносимые на защиту.**

1. Алгоритмы децентрализованной диспетчеризации в пространственно-распределённых ВС, обеспечивающие минимизацию среднего времени обслуживания параллельных программ и увеличение пропускной способности системы.
2. Программный пакет децентрализованной диспетчеризации параллельных программ в мультикластерных ВС, реализующий живучее обслуживание потоков параллельных программ.
3. Эвристические алгоритмы вложения в пространственно-распределённые ВС параллельных программ, минимизирующие время информационных обменов между параллельными ветвями.
4. Программный инструментарий оптимизации вложения параллельных MPI-программ в иерархические мультикластерные ВС.

**Структура и объем диссертации.** Диссертационная работа состоит из введения, четырёх глав, заключения и списка литературных источников, изложенных на 145 страницах, а также приложения на 1 странице.

**Содержание работы**

**В первой главе** даётся представление о распределённых вычислительных системах с программируемой структурой, раскрывается понятие пространственно-распределённых мультикластерных и GRID-систем. Рассмотрены задачи организации функционирования пространственно-распределённых ВС при обслуживании потоков параллельных программ.

**Во второй главе** рассматривается задача диспетчеризации программ в пространственно-распределённых ВС. Предлагается семейство алгоритмов децентрализованной диспетчеризации в пространственно-распределённых ВС параллельных программ, минимизирующих время их обслуживания. Предлагается алгоритм поиска субоптимальных локальных окрестностей децентрализованных диспетчеров. Приводятся результаты моделирования алгоритмов.

**В третьей главе** предлагаются эвристические алгоритмы вложения в иерархические пространственно-распределённые ВС параллельных программ. Алгоритмы учитывают все иерархические уровни коммуникационной среды системы и минимизируют время выполнения параллельных программ. Проводится сравнительный анализ созданных алгоритмов при вложении реальных MPI-программ.

**В четвёртой главе** описана архитектурная организация пространственно-распределённой мультикластерной ВС, в разработке которой диссертант принимал непосредственное участие. Приведено описание функциональной структуры разработанных на основе предложенных алгоритмов инструментов децентрализованной диспетчеризации и вложения параллельных программ.

**В заключении** сформулированы основные результаты диссертационной работы.

**В приложении** приведено описание структурной организации сегментов мультикластерной ВС.

# Глава 1. Распределённые вычислительные системы с программируемой структурой

## 1.1. Понятие о распределённых ВС с программируемой структурой

### 1.1.1. Модель коллектива вычислителей

Существует два подхода к проектированию и реализации средств обработки информации [2, 5]:

1) создание электронных вычислительных машин (ЭВМ), имитирующих процесс выполнения алгоритма одним человеком-вычислителем;
2) создание вычислительных систем, моделирующих процесс выполнения алгоритма коллективом людей-вычислителей.

ЭВМ является аппаратно-программным комплексом, построенным на основе модели вычислителя, который предназначен для автоматического выполнения логико-вычислительной работы: ввода, обработки, хранения и вывода информации. Согласно идеям Фон-Неймана, вычислительная машина реализует концепцию модели вычислителя и основополагающими принципами построения ЭВМ являются:

1) последовательная обработка информации;
2) фиксированность структуры;
3) неоднородность составляющих устройств и связей между ними.

Применение модели вычислителя для построения высокопроизводительных вычислительных средств ограничено теоретическими и техническими пределами скорости реализации операций (возможностями элементной базы и фундаментальными физическими законами). Данное обстоятельство привело к созданию вычислительных средств, в основе которых лежит модель коллектива вычислителей. Под коллективом вычислителей подразумевается группа вычис-

лительных машин, программно-аппаратным способом настраиваемая на решение общей задачи.

Техническая реализация модели коллектива вычислителей базируется на следующих архитектурных принципах [1–3]:

1) параллелизм при обработке информации;

2) программируемость структуры;

3) однородность конструкции.

Модель коллектива вычислителей основывается на диалектическом отрицании принципов, лежащих в основе модели вычислителя.

Параллелизм при обработке информации предполагает представление алгоритма решения задач с помощью параллельных программ. Параллельная программа – это совокупность связанных ветвей-программ, выполняющих одновременную (параллельную) работу над выделенными им данными и, при необходимости, взаимодействующих между собой [2].

Таким образом, применение данного принципа обеспечивает неограниченность в наращивании производительности вычислительного средства путём увеличения числа вычислителей.

Программируемость структуры предполагает возможность автоматической (программной) настройки проблемно-ориентированных (виртуальных) конфигураций и их перенастройки в процессе функционирования с целью обеспечения адекватности структурам и параметрам решаемых задач и достижения эффективности при заданных условиях эксплуатации.

Однородность конструкции заключается в построении коллектива вычислителей из множества идентичных элементов (модулей), однородным образом связанных между собой.

Вычислительное средство, основанное на модели коллектива вычислителей, называется *вычислительной системой* (ВС) [2, 3].

Наиболее законченное воплощение принципы модели коллектива вычислителей нашли в ВС с программируемой структурой. Разработка теоретических основ и принципов технической реализации ВС, а также создание первых однородных ВС с программируемой структурой были выполнены в СССР к началу 70-х годов XX в.

### 1.1.2. Классификация ВС

Существует несколько способов классификации ВС. Распространённой является классификация, предложенная в 1966 году М. Дж. Флинном [71, 72]. В зависимости от количества обрабатываемых вычислительным средством потоков инструкций и данных, разделяют четыре класса архитектур: SISD (Single Instruction stream / Single Data stream), SIMD (Single Instruction stream / Multiple Data stream), MISD (Multiple Instruction stream / Single Data stream), MIMD (Multiple Instruction stream / Multiple Data stream). Первый класс – SISD – относится к ЭВМ. Архитектуры MISD, SIMD, MIMD относятся к ВС. В соответствии с этими архитектурами допустимо построение нескольких типов ВС: 1) ВС конвейерного типа, 2) ВС матричного типа, 3) мультипроцессорные ВС, 4) распределённые ВС, 5) кластерные ВС, 6) пространственно-распределённые мультикластерные ВС.

Архитектура *конвейерных ВС* представляет собой предельную модификацию архитектуры последовательной ЭВМ. В основе таких систем лежит конвейерный метод обработки информации, а их функциональная структура представляется в виде "последовательности" соединённых элементарных блоков обработки информации. Все блоки функционируют параллельно, но каждый из них выполняет только свою совокупность операций над данными общего потока. Конвейерные ВС относятся к классу MISD-систем.

*Матричные ВС* реализуют возможность одновременного выполнения большого количества операций на элементарных процессорах (ЭП), "объединённых" в матрицу. Каждый ЭП – композиция арифметико-логического устройства

(АЛУ) и локальной памяти (ЛП); назначение ЛП – хранение части данных. Поток команд на матрицу ЭП генерируется устройством управления, которое имеет в своём составе память для хранения программ обработки данных. Такие ВС реализуют SIMD-архитектуру в первоначальном виде. Матричные ВС – это ВС массового параллелизма.

*Мультипроцессорные ВС* – это большая группа систем, к которой, в частности, могут быть отнесены конвейерные и матричные системы. Тем не менее, традиционно к мультипроцессорным ВС относят системы с MIMD-архитектурой, которые состоят из множества (не соединённых друг с другом) процессоров и общей (возможно также секционированной, модульной) памяти. Связь "процессор-память" осуществляется через коммутатор (общую шину и т.п.), а связь "процессор-процессор" – через память.

К *распределённым ВС* относят мультипроцессорные ВС с MIMD-архитектурой, в которых нет общего ресурса (памяти, коммутатора, устройства управления и т.д.). Основные составляющие распределённой ВС (такие, как коммутатор, устройство управления, арифметико-логическое устройство или процессор, память) допускают представление в виде композиции из одинаковых элементов (локальных коммутаторов и устройств управления, локальных процессоров и модулей памяти).

### 1.1.3. Вычислительные системы с программируемой структурой

Архитектура *вычислительных систем с программируемой структурой* полностью основана на модели коллектива вычислителей и представляет собой композицию элементарных машин (ЭМ), взаимодействующих друг с другом. Каждая ЭМ обязательно укомплектована локальным коммутатором, процессором и памятью; может также быть оснащена внешними устройствами. Локальная память ЭМ используется для хранения части данных и ветви параллельной программы. Архитектура ВС с программируемой структурой относится к типу

MIMD. По своим архитектурным возможностям ВС с программируемой структурой не уступают ни одному из перечисленных выше классов систем.

В семействе вычислительных систем с программируемой структурой выделяют пространственно сосредоточенные и распределённые ВС. Характерной особенностью *сосредоточенных* ВС является компактное пространственное размещение средств обработки и хранения информации, при котором среднее время передачи одного машинного слова между функциональными устройствами (процессорами, модулями памяти, ЭМ и др.) сопоставимо со средней длительностью выполнения одной операции в процессоре.

К *пространственно-распределённым* ВС относят системы сложной конфигурации, в которых в качестве функциональных элементов выступают территориально рассредоточенные вычислительные средства, основанные на моделях вычислителя и коллектива вычислителей, и сети связи, реализующие взаимный доступ между средствами обработки информации.

Концепция вычислительных систем с программируемой структурой была предложена в Сибирском отделении АН СССР; первая такая система ("Минск-222") была сконструирована в 1965 – 1966 гг. [1–3]

В 90-х годах XX столетия получают широкое распространение *кластерные вычислительные системы*. В наиболее общем понимании, кластерная ВС или вычислительный кластер – это композиция множества вычислителей, сети связей между ними и программных средств, предназначенная для параллельной обработки информации (выполнения параллельных алгоритмов решения сложных задач). При создании кластерной ВС могут быть использованы как стандартные серийные промышленные компоненты, так и специально разработанные средства.

Архитектура современных ВС существенно отличается от изначальных концепций: подавляющее большинство систем являются *мультиархитектурными*. Они могут выглядеть и как MISD, и как SIMD, и как MIMD в зависимости от уровня рассмотрения их функциональных структур.

### 1.1.4. Структуры коммуникационных сетей распределённых ВС

**Требования, предъявляемые к структуре ВС.** К структурам современных ВС предъявляется ряд требований [2, 3].

1) Возможность вложения параллельного алгоритма решения сложной задачи в структуру ВС. Структура системы должна подходить к достаточно широкому классу решаемых задач; настройка проблемно-ориентированных виртуальных структур не должна быть связана с существенными накладными расходами.
2) Простота адресации ЭМ и возможность "перемещения" выделенных для параллельных программ подсистем в пределах системы. ВС должна предоставлять возможность программистам создавать параллельные программы с виртуальными адресами ЭМ. Следовательно, структура ВС должна допускать реализацию простейшего "механизма" преобразования виртуальных адресов ЭМ в реальные (физические) адреса машин системы. Необходимость организации одновременного решения нескольких задач на ВС (т.е. необходимость разделения пространства элементарных машин между задачами) обуславливает требование простоты перемещения подсистем в пределах системы (при сохранении их топологических характеристик).
3) Осуществимость принципа близкодействия и минимума структурных задержек при межмашинных обменах в ВС. Принцип близкодействия предполагает реализацию обменов информацией между "удалёнными" друг от друга ЭМ через промежуточные ("транзитные") машины системы. Следовательно, в условиях ограниченности числа связей у каждой ЭМ структура должна обеспечивать минимум задержек при "транзитных" передачах информации.
4) Масштабируемость и большемасштабность структуры ВС. Для формирования конфигураций ВС с заданной эффективностью требуется, чтобы

структура допускала наращивание и сокращение числа вершин (машин). Изменение числа ЭМ в системе не должно приводить к коренным перекоммутациям между машинами и (или) к необходимости изменения числа связей для любых ЭМ.

Для достижения высокой производительности ВС при существующих возможностях микропроцессорной техники требуется количество ЭМ порядка $10 - 10^6$. Для поддержки большемасштабности (массового параллелизма) необходимо, чтобы структура ВС обладала способностью эффективно осуществлять межмашинные обмены информацией в условиях невозможности реализации связей по полному графу (например, из-за ограниченности числа выводов с корпусов БИС).

5) Коммутируемость структуры ВС. Вычислительная система должна быть адаптирована к реализации коллективных межмашинных обменов информацией. Следовательно, структура ВС должна обладать способностью осуществлять заданное число одновременных непересекающихся взаимодействий между элементарными машинами.

6) Живучесть структуры ВС. Существенным требованием к ВС является обеспечение работоспособности при отказе её отдельных компонентов или даже подсистем. Основой функциональной целостности ВС как коллектива элементарных машин является живучесть её структуры. Под последним понимается способность структуры ВС обеспечить связность необходимого числа работоспособных ЭМ в системе при ненадёжных линиях межмашинных связей.

7) Технологичность структур ВС. Структура сети межмашинных связей ВС не должна предъявлять особых требований к элементной базе и к технологии производства микропроцессорных интегральных схем. Системы должны быть основаны на использовании массовой технологии, их “вычислительное ядро” должно комплектоваться из массовых микропроцессорных

интегральных схем. Последнее позволит достичь приемлемых значений технико-экономических показателей ВС.

**Структурные характеристики ВС.** Структурные задержки при информационных обменах между ЭМ системы зависят от расстояния (в смысле теории графов) между вершинами структуры, соответствующими взаимодействующим ЭМ [1–3]. В качестве количественных характеристик структурных задержек в вычислительных системах используются диаметр $d$ и средний диаметр $\overline{d}$ структуры.

*Диаметр* определяется как максимальное расстояние на множестве кратчайших путей между парами вершин структуры ВС:

$$d = \max_{i,j}\{d_{ij}\},$$

*средний диаметр* определяется по формуле

$$\overline{d} = (N-1)^{-1}\sum_{l=1}^{d} l \cdot n_l,$$

где $d_{ij}$ – расстояние, под которым понимается минимальное количество рёбер, образующих путь из вершины $i$ в вершину $j$; $i, j \in 0, 1, \ldots, N-1$; $n_l$ – число вершин, расположенных на расстоянии $l$ от любой выделенной вершины (однородного) графа $G$.

Показателем, оценивающим структурную коммутируемость ВС, является вектор-функция

$$\mathfrak{K}(G, s, s') = \{\mathfrak{K}_h(G, s, s')\},\ h \in \{1, 2, \ldots, [N/2]\},$$

в которой координата $\mathfrak{K}_h(G, s, s')$ определяет вероятность реализации в системе $h$ одновременных непересекающихся межмашинных взаимодействий (обменов информацией между ЭМ) при заданных структуре $G$ и коэффициентах готовности $s$ и $s'$ одной ЭМ и линии связи, соответственно; $[x]$ – целая часть числа $x$.

Структурная живучесть ВС оценивается вектор-функцией

$$\mathfrak{L}(G, s, s') = \{\mathfrak{L}_r(G, s, s')\},\ r \in E_2^N = \{2, 3, \ldots, N\},$$

$\mathfrak{L}_r(G, s, s')$ есть вероятность существования подсистемы ранга $r$ (т.е. подмножества из $r$ работоспособных ЭМ, связность которых устанавливается через работоспособные линии связи) при заданных структуре $G$, коэффициентах готовности $s$ и $s'$ элементарной машины и линии связи, соответственно.

Для количественной оценки производительности каналов связи между ЭМ системы используют [73–75] такие показатели, как пропускная способность и латентность. *Пропускная способность канала связи* (*bandwidth*, *throughput*, *channel capacity*) – наибольший объём информации, передаваемый по каналу связи в единицу времени. *Латентность* (*latency*) – это временная задержка при передаче информации между ЭМ системы, вызванная программными и аппаратурными накладными расходами.

Введённые показатели позволяют выполнить с достаточной полнотой анализ структурных возможностей ВС.

**Перспективные структуры ВС.** Наиболее полно изложенным выше требованиям удовлетворяют однородные структуры (т.е. описываемые однородными графами) [1–3, 76, 77]. Такие структуры являются перспективными для формирования масштабируемых и большемасштабных вычислительных систем (в частности, ВС с программируемой структурой и пространственно-распределённых ВС).

В компьютерной индустрии получили широкое распространение $n$-мерные структуры вычислительных систем, известные сейчас как циркулянтные (Circulant Sructures). Впервые они были определены и исследованы в Отделе вычислительных систем Института математики СО АН СССР в начале 70-х годов XX в. и первоначально назывались *$D_n$-графами* [1]. По определению, *$D_n$-граф* или *циркулянтная структура* – это граф $G$ вида $\{N; \omega_1, \omega_2, \ldots, \omega_n\}$, в котором:

– $N$ – число вершин или порядок графа;

– вершины помечены целыми числами $i$ по модулю $N$, следовательно, $i \in \{0, 1, \ldots, N-1\}$;

– вершина $i$ соединена ребром (или является смежной) с вершинами $i \pm \omega_1, i \pm \omega_2, \ldots, i \pm \omega_n (\mathrm{mod}\, N)$;

– $\{\omega_1, \omega_2, \ldots, \omega_n\}$ – множество целых чисел, называемых образующими, таких, что $0 < \omega_1 < \omega_2 < \cdots < \omega_n < (N+1)/2$, а для чисел $N, \omega_1, \omega_2, \ldots, \omega_n$ наибольшим общим делителем является 1;

– $n$ – размерность графа;

– $2n$ – степень вершины в графе.

В качестве примера приведём $D_2$-граф (рис. 1.1) или двумерный циркулянт вида: $\{12; 3, 4\}$.

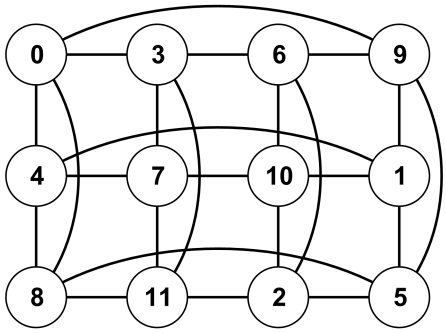



Рис. 1.1. $D_2$-граф: $\{12; 3, 4\}$

Графы $G$ вида $\{N; 1, \omega_2, \ldots, \omega_n\}$, т.е. $D_n$-графы или циркулянты с единичной образующей (Loop Networks – петлевые структуры) интенсивно изучаются в последнее время. Циркулянтные структуры $\{N; 1, \omega_2\}$ широко используются в практике вычислительных систем.

Целые числа $i \in \{0, 1, 2, \ldots, N-1\}$, соответствующие вершинам $D_n$-графа, называют адресами. Адресация вершин в циркулянтных структурах называется диофантовой (в честь древнегреческого математика из Александрии Диофанта, Diophantos, 3 век). В циркулянтных структурах при полном переносе какой-либо подструктуры (всех вершин подструктуры на одно и то же расстояние в одном из направлений) сохраняются все её свойства и адресация вершин. Следовательно, при диофантовой адресации элементарных машин ВС можно простыми средствами реконфигурации осуществить виртуальную адресацию вершин-машин и, следовательно, создавать отказоустойчивые параллельные про-

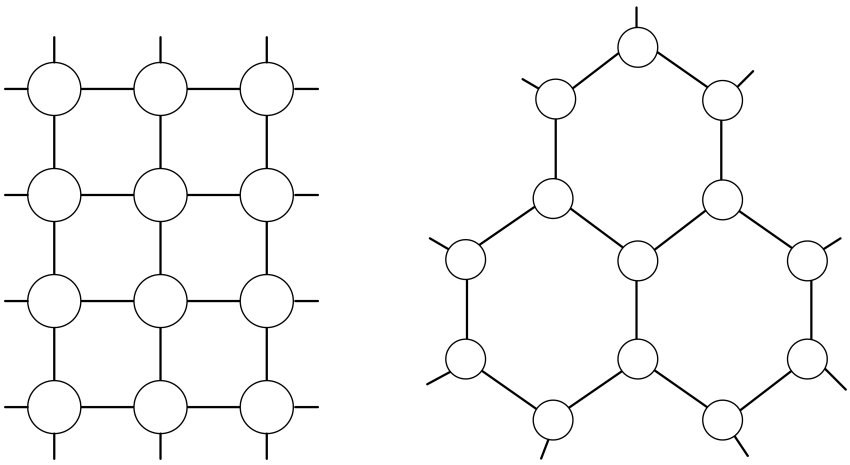
граммы, реализовывать мультипрограммные режимы обработки информации, исключать отказавшие вершины-машины из подсистем, а значит обеспечить живучесть ВС.

При этом алгоритм работы реконфигуратора структуры ВС сводится к изменению адресов $\alpha$ у всех машин подсистемы по формуле:

$$\alpha := [\alpha + (j - i)] \bmod N,\ \alpha \in \{0, 1, \ldots, N-1\},$$

где $i$ – номер ЭМ, исключаемой из подсистемы, а $j$ – номер машины, включаемой в подсистему, $i, j \in \{0, 1, \ldots, N-1\}$.

Среди структур ВС, допускающих масштабирование (изменение числа машин) без коренной перекоммутации уже имеющихся межмашинных связей, можно выделить $Л(N, v, g)$-графы [2, 3, 76, 77] (введённые также в Отделе вычислительных систем ИМ СО АН СССР). В такие графы вкладываются $D_n$-графы. $Л(N, v, g)$-граф – это неориентированный однородный граф с числом и степенями вершин, соответственно, $N$ и $v$ и значением обхвата $g$ (рис. 1.2).



*а* *б*

Рис. 1.2. Фрагменты $Л(N, v, g)$-графов:
$а$ – $v = 4, g = 4$; $б$ – $v = 3, g = 6$

В $Л(N, v, g)$-графах каждая вершина при $v \geq 3$ входит в не менее $v$ кратчайших простых циклов длиной $g$ (длина кратчайшего цикла в графе на-

зывается обхватом). При $v = 2$ Л$(N, v, g)$-граф является простым циклом с $N$ вершинами.

### 1.1.5. Параллельные алгоритмы и программы

Понятие параллельного алгоритма (Parallel Algorithm) относится к фундаментальным в теории вычислительных систем [1, 7, 11, 14, 16, 18, 21, 73, 78–82]. *Параллельный алгоритм* – это описание процесса обработки информации, ориентированное на реализацию в коллективе вычислителей [1–3]. Такой алгоритм, в отличие от последовательного, предполагает одновременное выполнение нескольких операций в пределах одного шага вычислений и, как последовательный алгоритм, сохраняет зависимость последующих шагов от результатов предыдущих.

Параллельный алгоритм решения задачи составляет основу параллельной программы, которая, в свою очередь, влияет на алгоритм функционирования коллектива вычислителей. Запись параллельного алгоритма на языке программирования, доступном коллективу вычислителей, называют *параллельной программой*, а сам язык – *параллельным*. Параллельные алгоритмы и программы следует разрабатывать для тех задач, которые недоступны для решения на средствах, основанных на модели вычислителя. Эти задачи принято называть сложными или трудоёмкими.

Методы и алгоритмы обработки информации, решения задач, как правило, – последовательные. Процесс "приспособления" методов к реализации на коллективе вычислителей или процесс "расщепления" последовательных алгоритмов решения сложных задач называется *распараллеливанием* (*Paralleling*).

Теоретическая и практическая деятельность по созданию параллельных алгоритмов и программ обработки информации называется *параллельным программированием* (*Parallel Programming*).

Качество параллельного алгоритма (или его эффективность) определяется методикой распараллеливания сложных задач. Выделяют два основных под-

хода к распараллеливанию задач [1, 21, 82]: локальное и глобальное (крупноблочное) распараллеливание. Первый подход ориентирован на расщепление алгоритма решения сложной задачи на предельно простые блоки (операции или операторы) и требует выделения для каждого этапа вычислений максимально возможного количества одновременно выполняемых блоков. Он не приводит к параллельным алгоритмам, эффективно реализуемым коллективом вычислителей. В самом деле, процесс такого распараллеливания весьма трудоёмок, а получаемые параллельные алгоритмы характеризуются не только структурной неоднородностью, но и существенно разными объёмами операций на различных этапах вычислений. Последнее является серьёзным препятствием на пути (автоматизации) распараллеливания и обеспечения эффективной эксплуатации ресурсов коллектива вычислителей. Локальное распараллеливание позволяет оценить предельные возможности коллектива вычислителей при решении сложных задач, получить предельные оценки по распараллеливанию сложных задач.

Второй подход ориентирован на разбиение сложной задачи на крупные блоки–подзадачи, между которыми существует слабая связность. Тогда в алгоритмах, построенных на основе крупноблочного распараллеливания, операции обмена между подзадачами будут составлять незначительную часть по сравнению с общим числом операций в каждой подзадаче. Такие подзадачи называют *ветвями параллельного алгоритма*, а соответствующие им программы – *ветвями параллельной программы*.

Пусть $V$ – количество операций, которые необходимо выполнить при решении задачи на ВС; $n$ – число параллельных ветвей или число вычислителей, на которых решается задача, $n \geq 2$; Тогда задачу, для которой выполняется условие

$$V \geq n \cdot 10^l, \tag{1.1}$$

будем называть *сложной*, или *системной*, или *трудоёмкой*, или с большим объёмом вычислений [2].

В соотношении (1.1) $l$ – эмпирический коэффициент, $l \geq 1$. Очевидно, что имеет место зависимость $l$ от быстродействия $v$ каналов связей между вычислителями: $l \to 1$ при $v \to v^*$, где $1/v^*$ – время обращения к локальной памяти в вычислителе.

Задачу, которая имеет небольшой объём вычислений и, следовательно, не допускает эффективного распараллеливания, будем называть *простой*. Простая задача требует для своего решения один вычислитель.

Структура обменов информацией между ветвями параллельной программы характеризуется информационным графом [8, 83–85]. *Информационный граф параллельной программы* (*Task graph*) – это конечный граф, вершинам которого соответствуют параллельные ветви, а ребрам – обмены информацией между ветвями.

## 1.2. Пространственно-распределённые ВС

### 1.2.1. Понятие о пространственно-распределённых ВС

Начало XXI в. ознаменовалось созданием *пространственно-распределённых ВС* как макроколлективов рассредоточенных подсистем (вычислительных кластеров и/или проприетарных систем с массовым параллелизмом), взаимодействующих между собой через локальные и глобальные сети (включая всемирную сеть Internet) [2]. К пространственно-распределённым относятся мультикластерные вычислительные и GRID-системы (рис. 1.3). Характерной чертой таких систем является их большемасштабность: современные GRID-системы (например, Enabling Grids for E-sciencE, Open Science Grid, D-Grid, Grid-5000) насчитывают десятки и сотни подсистем.

Каждая из подсистем функционирует под управлением локальной системы управления ресурсами (СУР) (TORQUE, Altair PBS Pro, SLURM и др.), которая поддерживает очереди пользовательских задач и выделяет для них вычислительные ресурсы (процессорные ядра).

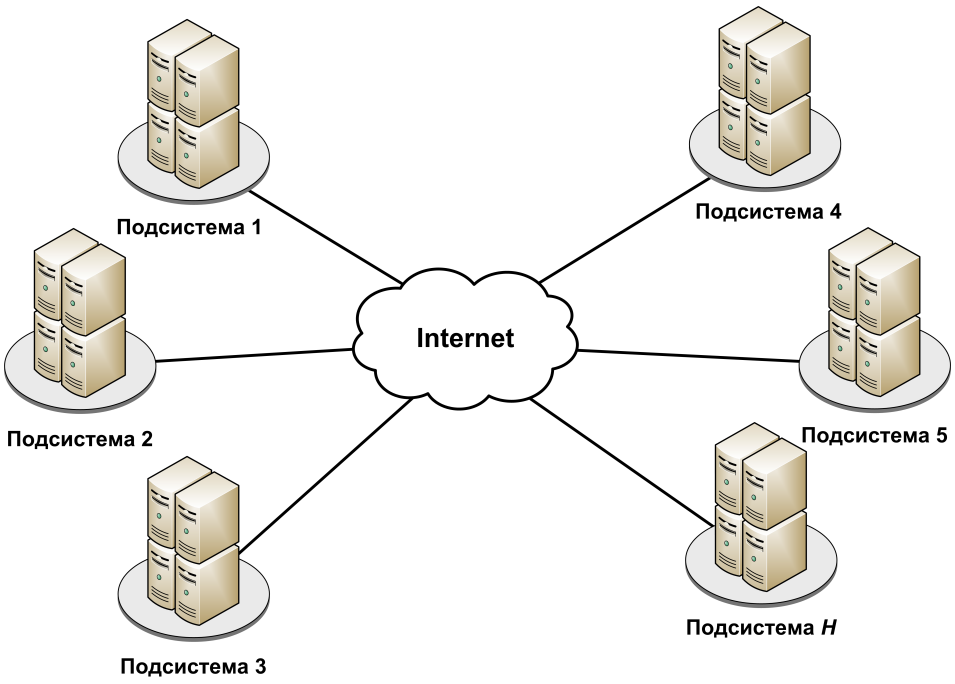



Рис. 1.3. Пространственно-распределённая мультикластерная ВС

Пространственно-распределённая ВС – объединение пространственно удалённых друг от друга сосредоточенных ВС (подсистем), основанное на принципах:

- *параллельности* функционирования сосредоточенных ВС (т.е. способности нескольких подсистем совместно и одновременно решать одну сложную задачу, представленную параллельной программой);
- *программируемости структуры*, т.е. способности автоматически настраивать сеть связи между сосредоточенными ВС;
- *гомогенности состава*, т.е. программной совместимости различных сосредоточенных ВС и однотипности элементарных машин в каждой из них.

Пространственно-рассредоточенные ВС предназначены для реализации параллельных программ (произвольной сложности в монопрограммном и мультипрограммном режимах) на рассредоточенных в пространстве вычислительных ресурсах. В пространственно-распределённых ВС допустимо централизованное и децентрализованное управление вычислительными процессами.

Развитым вариантом пространственно-распределённых ВС являются системы, структура сети связи которых в целом является нерегулярной и которые

комплектуются из программно-совместимых ЭМ. В данном случае сеть связи организуется посредством программно-совместимых системных устройств. Пространственно-распределённые системы, в которых выполняется условие программной совместимости ЭМ, относятся к гомогенным ВС. Программная совместимость обеспечивается посредством промежуточного программного обеспечения (middleware).

В класс пространственно-распределённых ВС входят вычислительные сети. Вычислительные сети выполняют обработку потоков задач, как правило, представленных последовательными программами. В пространственно-распределённых ВС, наряду с режимом работы, характерным для вычислительных сетей, реализуется также режим решения общей сложной задачи. Задача представляется параллельной программой, каждая ветвь которой реализуется на ЭМ различных сосредоточенных ВС. Таким образом, пространственно-распределённые системы относятся к более общему классу средств обработки информации по сравнению с вычислительными сетями и качественно от них отличаются возможностью решения одной сложной задачи на распределённых вычислительных ресурсах.

### 1.2.2. Мультикластерные и GRID-системы

В индустрии высокопроизводительной обработки информации к концу XX в. получили развитие так называемые GRID-технологии [2, 25–29]. На основе этих технологий создаются большемасштабные пространственно-распределённые ВС, способные реализовать суперсложные параллельные программы на своих рассредоточенных ресурсах. В GRID-системе могут использоваться гетерогенные и несовместимые вычислительные средства, однако для их совместной работы над параллельными алгоритмами должны быть применены специальные механизмы обеспечения совместимости (GRID middleware). В настоящее время наиболее интенсивно развивается программный пакет обеспе-

чения совместимости Globus Toolkit [25, 86] (рис. 1.4). Данный пакет включает в себя следующие средства.

1) *Средства авторизации пользователей* (MyProxy, SimpleCA) реализуют сервис сертификатов безопасности X.509. При регистрации пользователь получает ключ безопасности и пароль доступа. Перед работой в системе пользователь должен авторизироваться в центре сертификации (Certificate Authority) и получить временный ключ доступа.
2) *Средства передачи и синхронизации данных* (GridFTP, RLS) выполняют доставку данных авторизованных пользователей между любыми подсистемами ВС. При передаче данных используются механизмы, позволяющие снизить время доставки данных (кэширование, дозагрузка при разрыве соединения и др.).
3) *Средства взаимодействия с локальными СУР* (GRAM). На подсистемах мультикластерных и GRID-систем установлены различные СУР (TORQUE, SLURM, Altair PBS Pro). Подсистема GRAM выполняет постановку в очередь, мониторинг и удаление задач из очереди СУР посредством унифицированного интерфейса.

Пользователь системы может использовать средства пакета Globus Toolkit для решения параллельных задач. Однако для этого он должен выбрать подсистему из числа доступных, сформировать паспорт задачи и запустить параллельную программу на подсистеме. Автоматизированный поиск ресурсов для выполнения параллельных программ осуществляется диспетчером GRID-системы (например, GridWay). В этом случае пользователю достаточно отправить задачу диспетчеру, который выберет подсистему ЭМ для её решения и выполнит доставку входных и выходных файлов в соответствии с информацией, указанной пользователем в паспорте задачи.

Предполагается, что GRID-вычисления – это скоординированное разделение гетерогенных пространственно распределенных вычислительных ресур-

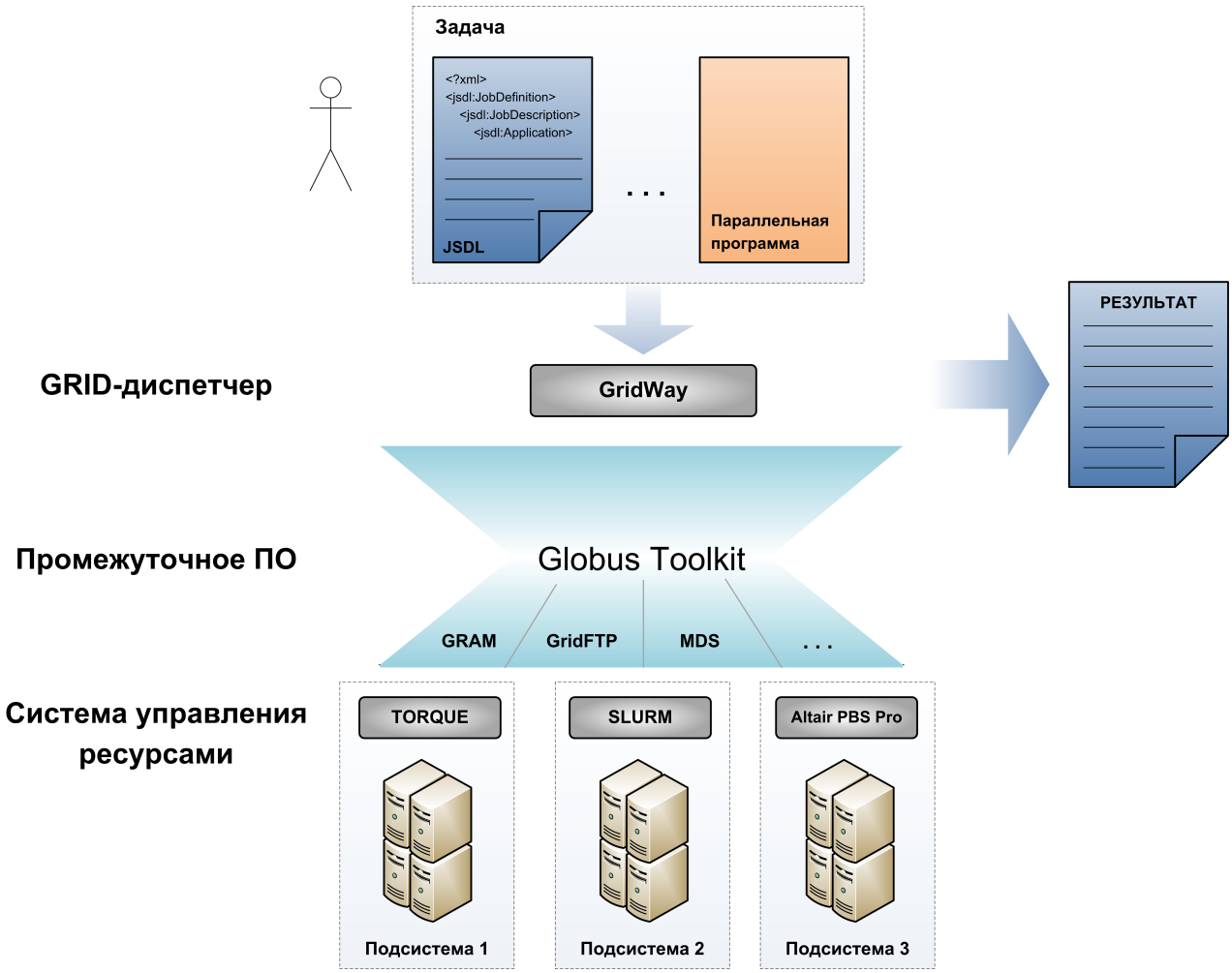



Рис. 1.4. Функциональная схема GRID-системы на базе Globus Toolkit

сов и решение задач в динамически меняющихся виртуальных организациях со многими участниками. Ключевой концепцией GRID является достижение договорённости о разделении ресурсов между поставщиками и потребителями и использование полученного пула ресурсов для различных целей. Это разделение должно контролироваться поставщиками и потребителями ресурсов; в нём должны быть оговорены объёмы ресурсов, условия, на которых они предоставляются и права доступа к ним отдельных пользователей. Пользователи или (и) институты, подчиняющиеся таким правилам, образуют *виртуальную организацию*.

Выделяют ряд основополагающих отличительных особенностей мульти-кластерных и GRID-систем [26–29].

- Координированное использование ресурсов на основе децентрализованного управления. Система интегрирует и координирует ресурсы и пользователей, географически удалённых друг от друга.
- Стандартные, открытые, универсальные протоколы и интерфейсы. Система строится на базе открытых многоцелевых протоколов и интерфейсов, позволяющих решать такие задачи, как аутентификация, авторизация, обнаружение ресурсов и доступ к ресурсам.
- Обеспечение высококачественного обслуживания. Система позволяет использовать ресурсы таким образом, чтобы обеспечивалось высокое качество обслуживания, касающееся, например, таких параметров, как время отклика, пропускная способность, доступность и надёжность.

Важной особенностью мультикластерных и GRID-систем является переменный характер состава и загрузки их ресурсов. В произвольные моменты времени в системе происходят отказы ресурсов (ЭМ, подсистем, линий связи); политики предоставления доступа к ресурсам могут существенно различаться между собой (это вызвано независимым администрированием подсистем). Вследствие этого в произвольные моменты времени могут происходить появление и выход из строя ЭМ и отдельных подсистем, а также изменение производительности каналов связи и ЭМ.

### 1.2.3. Основные режимы функционирования пространственно-распределённых ВС

Использование потенциальных возможностей пространственно-распределённых ВС сильно зависит от того, как организовано функционирование ВС, т.е. как осуществляется процесс решения задач. Методы и алгоритмы организации функционирования ВС должны обеспечить эффективное решение задач и должны быть эффективно реализуемы на ЭВМ и ВС. Эффективность оценивается с точки зрения различных показателей (целевых функций), таких как среднее время обслуживания задач, пропускная способность системы, энергопо-

требление и т.д. Алгоритмы функционирования должны обеспечить оптимальные значения целевых функций.

В зависимости от сложности задач и характера их поступления можно выделить три режима функционирования пространственно-распределённых ВС [1–3]: решение сложной задачи, обработка набора задач, обслуживание потока задач.

Первый режим – *монопрограммный*, т. е. для решения задачи используются все ресурсы ВС. Задача представляется в виде параллельной программы, число ветвей в которой либо фиксировано, либо допускает варьирование в заданном диапазоне. В качестве единицы ресурса выступает элементарная машина подсистемы ВС. Все машины используются для решения задачи.

Второй и третий режимы функционирования ВС относятся к *мультипрограммным*. При работе ВС в этих режимах одновременно решается несколько задач, следовательно, ресурсы делятся между несколькими задачами.

При организации функционирования ВС в случае набора задач учитывается не только количество задач, но и их параметры: число ветвей в программе, время решения или вероятностный закон распределения времени решения и др. Алгоритмы организации функционирования ВС задают распределение задач по машинам и последовательность выполнения задач на каждой машине. В результате становится известным, в какой промежуток времени и на каких машинах какой подсистемы будет решаться любая задача набора. Этот режим является обобщением мультипрограммных режимов для ЭВМ. При обработке наборов параллельных задач ресурсы ВС также распределяются между задачами, однако в любой момент времени задачи решаются на непересекающихся подмножествах машин. В отличие от мультипрограммных режимов работы ЭВМ, которые реализуются разделением времени процессора, обработка наборов задач на ВС осуществляется посредством разделения “пространства” машин.

### 1.2.4. Режим обслуживания потока задач

В диссертационной работе рассматривается функционирование пространственно-распределённых ВС в режиме обслуживания потока задач. В систему (в распределённую очередь) в случайные моменты времени поступают параллельные задачи. Задачу характеризует ранг (количество параллельных ветвей), время решения, требования на архитектуру процессора и программное обеспечение (ПО), размеры и местоположения входных и выходных файлов и т.д.

Для режима обслуживания потока задач в сосредоточенных ВС созданы методы и алгоритмы [1–3, 13], обеспечивающие стохастически оптимальное функционирование ВС. Однако эти алгоритмы не учитывают особенностей пространственно-распределённых ВС и могут не обеспечить в них предельной эффективности функционирования.

В моделях, методах и программном обеспечении организации функционирования пространственно-распределённых ВС должны учитываться архитектурные свойства современных ВС: большемасштабность, мультиархитектурная организация (наличие SMP, NUMA-узлов и специализированных ускорителей) и иерархическая структура коммуникационной среды.

При обслуживании потока задач значимой является задача *диспетчеризации*. Для поступающих от пользователей параллельных задач необходимо определить множество ЭМ с одной или нескольких подсистем с целью оптимизации одного или нескольких показателей эффективности. В системах диспетчеризации необходимо принимать во внимание большемасштабность системы и динамический характер состава и загрузки ресурсов.

Одним из показателей эффективности является время обслуживания задачи, которое включает в себя время доставки входных и выходных данных задачи, время ожидания в локальных очередях на подсистемах и время выполнения параллельной программы на ЭМ.

В отличие от обработки задач набора, параметры задач случайны, следовательно, детерминированный выбор подсистем для решения задач исключён. Также стоит заметить, что в средствах диспетчеризации необходимо предусмотреть возможность выделения ЭМ с нескольких подсистем для решения одной параллельной задачи. Существует класс задач, допускающих эффективную реализацию на ресурсах пространственно-рассредоточенных подсистем. К ним относятся задачи моделирования на основе метода Монте-Карло, задачи имитационного моделирования, явные разностные схемы решения дифференциальных уравнений и др. В информационных графах их параллельных программ можно выделить подмножества ветвей, слабо взаимодействующих между собой. Такие задачи с целью минимизации времени их обслуживания могут быть запущены на ресурсах нескольких подсистем.

После того, как сформирована подсистема, необходимо оптимально вложить задачу в неё: распределить ветви по процессорным ядрам ВС с целью минимизации накладных расходов на информационные обмены между параллельными ветвями. Под эффективным *вложением* (*Task Map, Task Allocation, Task Assignment*) понимается такое распределение ветвей параллельной программы между ЭМ системы, при котором достигаются минимумы накладных расходов на межмашинные обмены информацией.

При вложении необходимо учитывать информационный граф параллельной программы, часто имеющий неоднородную структуру. Также необходимо принимать во внимание преобладающие типы информационных обменов. Например, в параллельных программах на языках семейства Partitioned Global Address Space (PGAS) преобладают обмены короткими сообщениями. Не менее важно учитывать иерархическую структуру ВС. В мультикластерных вычислительных и GRID-системах узлы внутри подсистем связаны быстрыми каналами связи, в то время как узлы разных подсистем взаимодействуют через каналы низкой производительности.

Проблема вложения в недостаточной степени проработана для пространственно-распределённых ВС, поэтому востребованы алгоритмы оптимизации вложения параллельных программ в мультикластерные и GRID-системы. В диссертации предложены алгоритмы вложения в мультикластерные ВС с иерархической структурой параллельных программ с целью минимизации времени их выполнения.

## 1.3. Выводы

1. Пространственно-распределённые мультикластерные и GRID-системы являются современным инструментарием высокопроизводительной обработки информации. Данные системы являются большемасштабными и характеризуются динамическими составом и загрузкой ресурсов.
2. Эффективность эксплуатации пространственно-распределённых ВС существенно зависит от способов организации их функционирования.
3. К значимым проблемам, возникающим при обслуживании потока задач, относится диспетчеризация параллельных программ, поступающих в распределённую очередь системы.
4. Актуальной является задача вложения параллельных программ в пространственно-распределённые ВС. Под оптимальным вложением понимается такое распределение ветвей параллельной программы, при котором достигаются минимумы времени выполнения информационных обменов. При оптимизации вложения необходимо учитывать иерархическую структуру системы.

# Глава 2. Децентрализованная диспетчеризация мультикластерных ВС

## 2.1. Диспетчеризация задач в пространственно-распределённых ВС

### 2.1.1. Задача диспетчеризации параллельных задач

Рассмотрим задачу диспетчеризации параллельных задач в пространственно-распределённых ВС.

Пусть имеется пространственно-распределённая мультикластерная ВС, состоящая из $H$ подсистем; $N$ – суммарное количество ЭМ на подсистемах. Под ЭМ понимается единица вычислительного ресурса, предназначенного для выполнения ветви параллельной программы (например, процессорное ядро). На каждой подсистеме присутствует локальная СУР и децентрализованный диспетчер, который поддерживает свою очередь параллельных задач и осуществляет поиск вычислительных ресурсов для их выполнения (рис. 2.1).

Коллектив диспетчеров представлен в виде ориентированного графа $G(S, E)$, в котором вершинам соответствуют диспетчеры, а рёбрам – логические связи между ними. Наличие дуги $(i, j) \in E$ в графе означает, что диспетчер $i$ может отправлять задачи диспетчеру $j$. Множество всех вершин $j$, смежных вершине $i$, образует её локальную окрестность $L(i) = \{j \in S | (i, j) \in E\}$.

Пусть задача поступила на вход диспетчера подсистемы $i$. Задача содержит программу, входные файлы и ресурсный запрос, в котором указываются ранг $r$ программы (количество параллельных ветвей), размеры $z_1, z_2, \ldots, z_k$ исполняемого и входных файлов программы ($[z_l]$ = байт), а также номера $h_1, h_2, \ldots, h_k$ подсистем на которых размещены соответствующие файлы ($h_l \in S$).

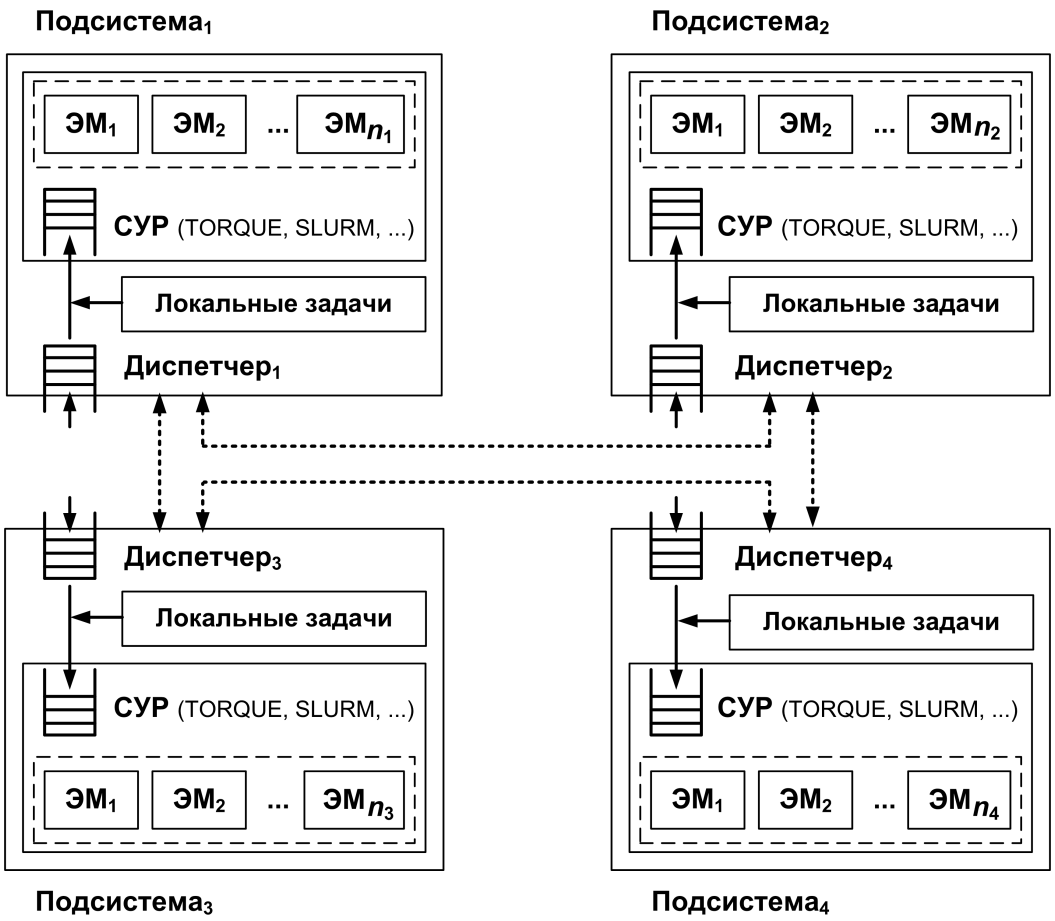



Рис. 2.1. Децентрализованная диспетчеризация
в пространственно-распределённых ВС

Введём обозначения: $n_j$ – количество ЭМ, входящих в состав подсистемы $j \in S = 1, 2, ..., H$; $c_j$ – число свободных ЭМ в подсистеме $j$; $q_j$ – число задач в очереди подсистемы $j$; $s_j$ – число задач, выполняющихся на ЭМ подсистемы $j$; $t_{jk} = t(j, k, m)$ – время передачи сообщения размером $m$ байт между подсистемами $j, k \in S$ ($[t(j, k, m)] = \text{с}$). Считается, что все подсистемы объединены сетью связи.

Требуется отыскать подсистему $j^*$, в которой достигается минимум функции $F(j)$

$$j^* = \underset{j \in L(i) \cup \{i\}}{\operatorname{argmin}} \{F(j)\}, \tag{2.1}$$

$$F(j) = \begin{cases} \dfrac{t_j}{t_{\max}} + \dfrac{c_{\max}}{c_j} + \dfrac{w_j}{w_{\max}}, & \text{если } c_j < r \text{ или } q_j > 0, \\ \dfrac{t_j}{t_{max}}, & \text{иначе.} \end{cases} \tag{2.2}$$

Здесь $t_{\max} = \max\limits_{j \in S(i)}\{t_j\}$; $c_{\max} = \max\limits_{j \in S(i)}\{c_j\}$; $w_j = q_j / n_j$ – количество задач в очереди, приходящееся на одну ЭМ подсистемы $j$; $w_{\max} = \max\limits_{j \in S(i)}\{w_j\}$.

Поясним значение каждого слагаемого в (2.2). $\dfrac{t_j}{t_{\max}}$ – относительное время доставки входных файлов задачи на подсистему $j$, $\dfrac{c_{\max}}{c_j}$ – относительное количество свободных ЭМ в подсистеме $j$, $\dfrac{w_j}{w_{\max}}$ – относительное число задач в очереди, приходящееся на одну ЭМ подсистемы.

Функция $F(j)$ учитывает текущую загрузку подсистем мультикластерных ВС (количество задач в очередях, число свободных процессоров) и время доставки входных файлов до подсистемы [87].

### 2.1.2. Обзор средств диспетчеризации задач в пространственно-распределённых ВС

В настоящее время большинство систем диспетчеризации задач в пространственно-распределённых ВС являются централизованными. Они подразумевают наличие в системе центрального диспетчера, поддерживающего глобальную очередь задач. Основным недостатком такого подхода является то, что отказ диспетчера приводит к неработоспособности всей системы. Помимо этого, в случае использования централизованной системы в большемасштабных ВС возрастают временные затраты на поиск требуемых ресурсов.

На сегодняшний день среди программных пакетов диспетчеризации задач в пространственно-распределённых ВС в первую очередь следует назвать [88, 89]: GridWay [30–34, 44, 90], AppLeS [35, 36], GrADS [37, 38], Nimrod/G [39–41], Condor-G [42], WMS [44, 45]. Пакет GridWay является наиболее распространённым и интенсивно развивающимся GRID-диспетчером. В нём при диспетчеризации преследуется цель минимизации времени обслуживания задач. Поддерживается механизм миграции задач между подсистемами. Реализован инструмент выполнения задач, представленных в виде ориентированных ациклических графов. В пакете AppLeS диспетчеризация выполняется

на уровне самой программы. Это требует от пользователей знания конфигурации системы, что снижает универсальность диспетчера. Пакет GrADS, как и GridWay, реализует миграцию задач и допускает указание зависимостей по данным между задачами. Nimrod/G основан на экономических моделях, обеспечивающих равновесие между условными поставщиками (подсистемами) и потребителями (задачами) вычислительных ресурсов. Диспетчер Condor-G максимизирует пропускную способность системы и поддерживает представление задач в виде ориентированных ациклических графов.

В настоящее время перспективным направлением являются мультиагентные системы диспетчеризации [46–50], к числу которых относится AgentScape [47]. В некоторых системах, таких как DIRAC [43], на подсистемах функционируют агенты, которые при освобождении ресурсов запрашивают задачи из глобальной очереди. Недостатком такого подхода является наличие глобальной очереди. Предлагаемые мультиагентные методы диспетчеризации, а также модели децентрализованной диспетчеризации [51, 91] могут быть использованы и в большемасштабных пространственно-распределённых ВС, однако в них в недостаточной степени учитывается переменный характер состава и загрузки подсистем и каналов связи.

Значительное место занимают работы, связанные с созданием инструментария диспетчеризации задач, имеющих зависимости по данным и представленных в виде ориентированных графов (GRID Workflows) [52–70, 92]. Существующие системы диспетчеризации, такие как Pegasus [52–54], Taverna [56–58], Triana [59–61], ASKALON [62, 63], ICENI [64, 65], GridBus [66], Kepler [67, 68], Karajan [55], GridAnt [69], UNICORE [70], являются централизованными. Кроме того, большинство диспетчеров ориентировано на применение в узкоспециализированной области.

В диссертации предлагается децентрализованный подход к диспетчеризации задач в пространственно-распределённых ВС (рис. 2.1). При этом на каждой подсистеме функционирует диспетчер, взаимодействующий с ограни-

ченным числом других диспетчеров (которые образуют его локальную окрестность). Коллектив диспетчеров реализует распределённую очередь задач и совместно принимает решение о выборе ресурсов. Это позволяет достичь живучести ВС – способности систем продолжать работу при отказах отдельных компонентов и подсистем.

Для распределённых вычислительных систем с программируемой структурой созданы эффективные децентрализованные алгоритмы управления ресурсами [77, 93]. Однако применимость этих методов для пространственно-распределённых ВС ограничена, поскольку в алгоритмах не учитываются производительность каналов связи между подсистемами и возможность образования очередей задач на подсистемах.

В диссертационной работе предлагается четыре алгоритма децентрализованной диспетчеризации, основанные на локально-оптимальном подходе (ДЛО), на репликации задач (ДР), на миграции задач (ДМ) и на комбинированном подходе (ДРМ) [94–111]. Алгоритмы, благодаря механизмам, реализованным в них, позволяют учитывать переменный характер загрузки ресурсов и производительность каналов связи.

Также проводится исследование влияния логической структуры локальной окрестности диспетчеров на эффективность диспетчеризации. Предлагается эвристический алгоритм поиска субоптимальных логических структур локальных окрестностей децентрализованных диспетчеров.

## 2.2. Алгоритмы децентрализованной диспетчеризации задач в пространственно-распределённых ВС

Каждый алгоритм описывает функционирование диспетчера $i$ при поступлении задачи в его очередь. На предварительном шаге всех алгоритмов у системы мониторинга запрашиваются значения параметров $t_{ij}, c_j, s_j, q_j, n_j$ для под-

систем $j \in L(i) \cup \{i\}$ и строится множество $S(i) = \{j | n_j \geq r, j \in L(i) \cup \{i\}\}$ подсистем, подходящих для выполнения параллельной программы.

### 2.2.1. Локально-оптимальный алгоритм диспетчеризации задач (ДЛО)

Алгоритм осуществляет выбор локально-оптимальной подсистемы (рис. 2.2).

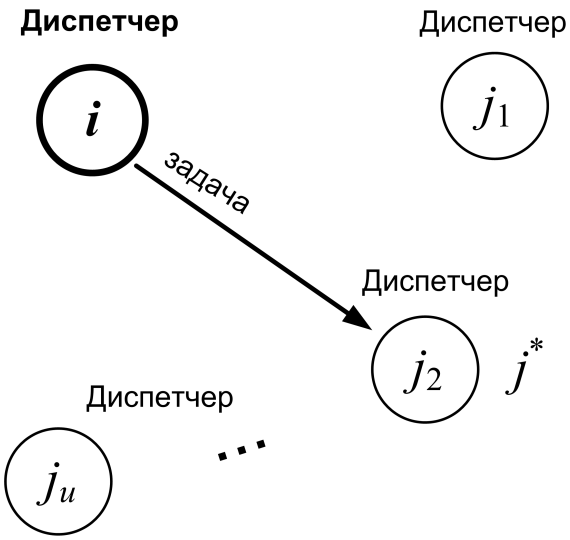



Рис. 2.2. Локально-оптимальный алгоритм диспетчеризации (ДЛО)

Шаг 1. Из окрестности $S(i)$ диспетчера $i$ выбирается подсистема $j^*$ с минимальным значением функции $F(j)$, $j \in S(i)$.

Шаг 2. Задача направляется в очередь локальной СУР подсистемы $j^*$, после чего осуществляется доставка файлов задачи на эту подсистему.

Ранжирование подсистем по значению функции $F(j)$ позволяет учесть время доставки файлов задачи до подсистем, а также их относительную загруженность. Основным недостатком алгоритма является то, что подсистема выделяется для задачи статически: отсутствует возможность перенаправления задачи на другие подсистемы. После назначения задачи на подсистему значительные изменения в загрузке подсистемы может привести к увеличению среднего времени обслуживания задач. Этот алгоритм может быть эффективно использован в системах со стабильной загрузкой подсистем и каналов связи.

Кроме целевой функции (2.1), были предложены другие целевые функции, учитывающие загрузку ресурсов и производительность каналов связи:

$$j^* = \underset{j \in L(i) \cup \{i\}}{\operatorname{argmin}} \{F'(j)\},$$

$$F'(j) = \begin{cases} \dfrac{t_j}{t_{\max}} + \dfrac{c_{\max}}{c_j} + \dfrac{w_j}{w_{\max}}, & \text{если } c_j < r \text{ или } q_j > 0, \\ \dfrac{t_j}{t_{max}}, & \text{иначе.} \end{cases} \tag{2.3}$$

где $w_j = (s_j + q_j)/n_j$, и

$$j^* = \underset{j \in L(i) \cup \{i\}}{\operatorname{argmin}} \{F''(j)\},$$

$$F''(j) = \begin{cases} \dfrac{t_j}{t_{\max}} + \dfrac{w_j}{w_{\max}}, & \text{если } c_j < r \text{ или } q_j > 0, \\ \dfrac{t_j}{t_{max}}, & \text{иначе.} \end{cases} \tag{2.4}$$

где $w_j = q_j/n_j$.

### 2.2.2. Алгоритм диспетчеризации на основе репликации задач (ДР)

В основе данного алгоритма лежит процедура назначения задачи одновременно на несколько подсистем (рис. 2.3).

Шаг 1. Из окрестности $S(i)$ выбирается $m$ подсистем $j_1^*, j_2^*, \ldots, j_m^*$ в порядке неубывания значений функции $F(j)$.

Шаг 2. Задача одновременно направляется в очереди локальных СУР подсистем $j_1^*, j_2^*, \ldots, j_m^*$, после чего осуществляется доставка файлов задачи до этих подсистем.

Шаг 3. С интервалом времени $\Delta_1$ диспетчер $i$ проверяет состояние задачи на подсистемах $j_1^*, j_2^*, \ldots, j_m^*$ и определяет подсистему $j'$, на которой задача запущена на выполнение раньше других подсистем.

Шаг 4. Задача удаляется из очередей локальных СУР подсистем отличных от $j'$.

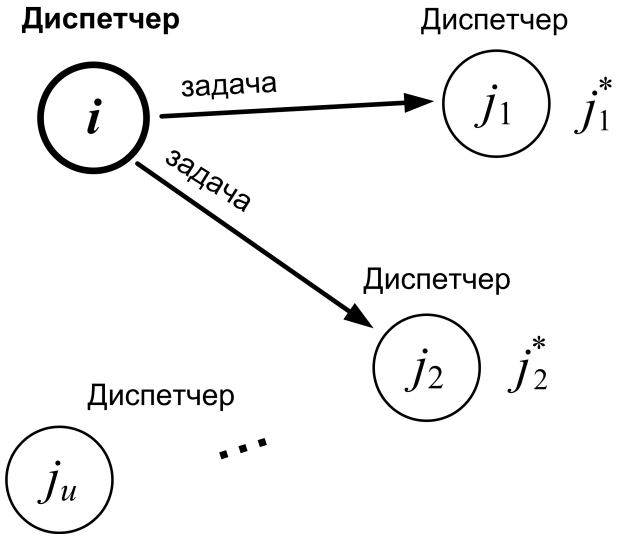



Рис. 2.3. Алгоритм диспетчеризации на основе репликации (ДР)

Механизм репликации позволяет учесть динамически изменяемую загрузку подсистема. В большинстве случаев он значительно загружает коммуникационную сеть и файловую систему, которые могут стать узким местом. Поэтому данный алгоритм рекомендуется использовать для небольшого числа задач с высоким приоритетом, поскольку для них алгоритм позволяет учесть разницу в загрузке подсистем и снизить время их решения.

### 2.2.3. Алгоритм диспетчеризации на основе миграции задач (ДМ)

В данном алгоритме реализован периодический поиск новых подсистем для задач из очереди диспетчера (рис. 2.4).

Шаг 1. Из окрестности $S(i)$ диспетчера $i$ выбирается подсистема $j^*$ с минимальным значением $F(j)$, $j \in S(i)$.

Шаг 2. Задача направляется в очередь локальной СУР подсистемы $j^*$, после чего осуществляется доставка файлов задачи на эту подсистему.

Шаг 3. Диспетчер $i$ с интервалом времени $\Delta_2$ запускает процедуру ДЛО поиска подсистемы $j'$ для задачи в очереди диспетчера $j^*$.

Шаг 4. Если для найденной подсистемы выполняется условие $F(j^*) - F(j') > \varepsilon$, то выполняется миграция задачи в очередь подси-

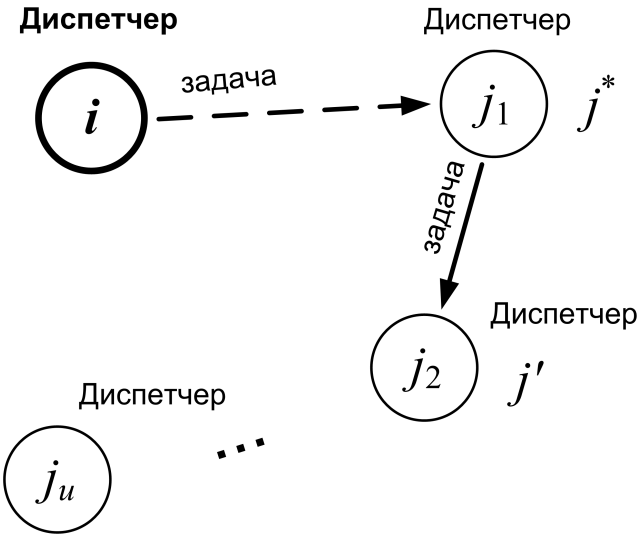



Рис. 2.4. Алгоритм диспетчеризации на основе миграции (ДМ)

стемы $j'$ (задача удаляется из очереди диспетчера $j^*$). Здесь $\varepsilon$ – критерий миграции, определяемый администратором системы.

Главная особенность алгоритма ДМ в том, что он сочетает достоинства локально-оптимальной диспетчеризации и диспетчеризации на базе репликации. Он учитывает изменения загрузки подсистем и в то же время не загружает каналы связи и файловую систему. Однако отсутствие возможности точного предсказания момента запуска задачи может вызвать нежелательную миграцию задачи.

### 2.2.4. Алгоритм диспетчеризации на основе комбинированного подхода (ДРМ)

Алгоритм основан на комбинации двух подходов – назначения на несколько подсистем и миграции из очереди (рис. 2.5). Задача ставится в очередь нескольких подсистем и затем может мигрировать между подсистемами распределённой ВС.

Важно отметить, что вычислительная сложность поиска подсистем предложенными алгоритмами ДЛО, ДР, ДМ, ДРМ не зависит от количества $H$ подсистем, так как поиск осуществляется только в пределах локальных окрест-

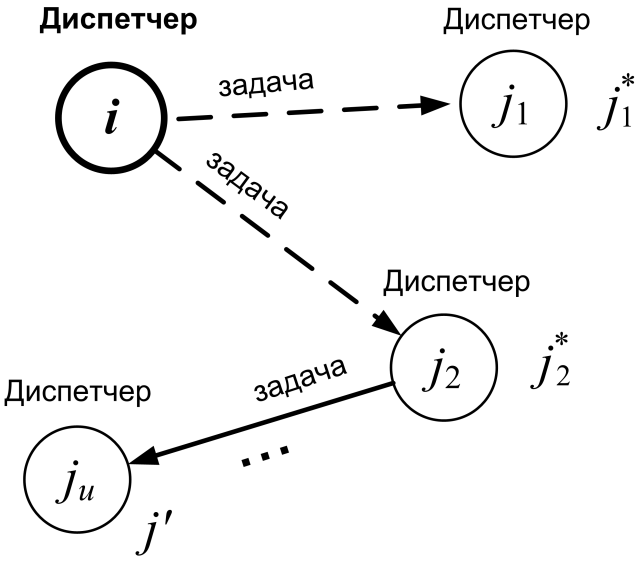



Рис. 2.5. Алгоритм диспетчеризации на основе комбинированного подхода (ДРМ)

ностей диспетчеров и имеет вычислительную сложность $T = O(|S(i)|\Delta t)$, где $\Delta t$ – время получения информации о производительности одной подсистемы и времени доставки данных. Это обеспечивает применимость алгоритмов в большемасштабных пространственно-распределённых ВС.

### 2.2.5. Примеры работы алгоритмов

Рассмотрим пример диспетчеризации задач в мультикластерной ВС согласно разработанным алгоритмам. Пусть на первую подсистему мультикластерной ВС, состоящей из $H = 7$ подсистем, поступает задача, представленная параллельной программой ранга $r = 16$ (рис. 2.6). Известны размеры входных файлов $z_1 = 1$Гб и $z_2 = 2$Гб и подсистемы $h_1 = 5$ и $h_2 = 6$, на которых они расположены. Считается, что у систем мониторинга запрошены параметры подсистем $i = 1, \ldots, 7$: $n_i$ – количество ЭМ, входящих в состав подсистемы, $c_i$ – число свободных ЭМ (не занятых решением задач), $q_i$ – число задач в очереди подсистемы, $t_i$ – оценка времени доставки входных файлов задачи на подсистему (рис. 2.6).

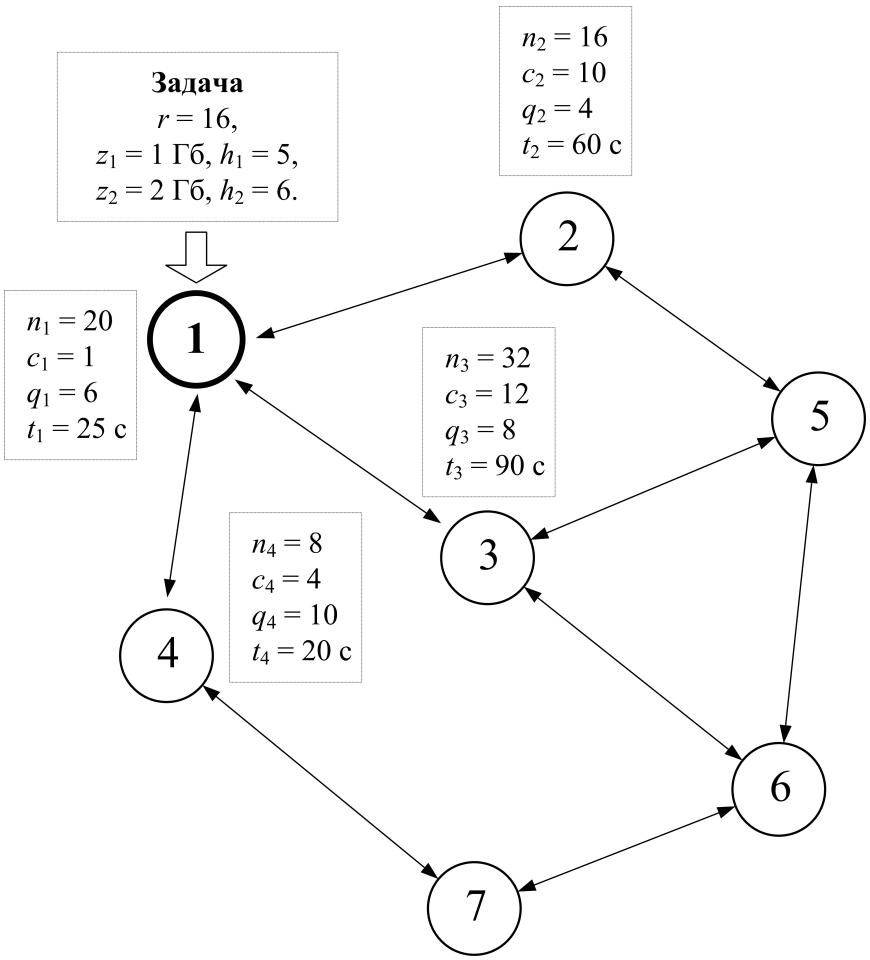



Рис. 2.6. Пример диспетчеризации задачи на мультикластерной ВС

Локальная окрестность подсистемы 1: $L(1) \cup \{1\} = \{1, 2, 3, 4\}$. Множество "подходящих" подсистем с количеством ЭМ $n_i$ не меньшим ранга параллельной программы: $S(1) = \{1, 2, 3\}$. Рассчитаем количество $w_i$ задач в очереди подсистемы $i$, приходящееся на одну ЭМ: $w_1 = 0,3$, $w_2 = 0,25$, $w_3 = 0,25$.

Рассмотрим алгоритм функционирования коллектива диспетчеров в соответствии с разработанными алгоритмами ДЛО, ДР, ДМ, ДРМ.

*Алгоритм ДЛО.* По известным параметрам подсистем определим значения целевой функции (2.2) для подсистем $j \in S(1)$: $F(1) = 15,28$, $F(2) = 2,90$, $F(3) = 2,83$ (рис. 2.7а). Задача ставится в очередь подсистемы 3 с минимальным значением целевой функции (2,83) (рис. 2.7б). Входные файлы доставляются на подсистему 3 с подсистем 5, 6, и выполняется запуск параллельной

программы. По завершении выполнения выходные файлы доставляются на указанные в паспорте задачи подсистемы.

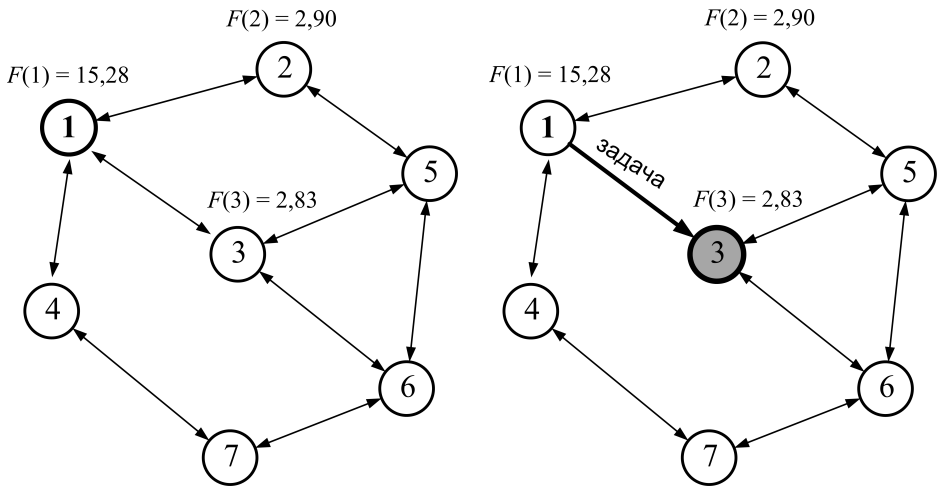



Рис. 2.7. Пример диспетчеризации задачи на мультикластерной ВС, алгоритм ДЛО

*Алгоритм ДР*. Предположим, значение параметра алгоритма ДЛО $m = 2$ (количество подсистем, в очередь которых одновременно ставится задача). Тогда после вычисления значений целевой функции для подсистем множества $S(1)$, задача ставится в очередь подсистем 2 и 3 с наименьшими значениями целевой функции (рис. 2.8а) (2,90 и 2,83, соответственно). Выполняется доставка входных файлов с подсистем 5 и 6.

Считаем, к моменту поступления задачи в очередь подсистем 2 и 3 на них нет достаточного количества свободных ЭМ. Программа находится в очередях обоих подсистем до тех пор, пока на одной из них не освободится достаточное для её реализации количество ЭМ. Через интервал времени $\Delta_1$ выполняется проверка числа доступных ЭМ на подсистемах 2 и 3. Предположим, на подсистеме 2 освободилось достаточное количество ЭМ (16). В этом случае задача удаляется из очереди подсистемы 3 и происходит её запуск на подсистеме 2 (рис. 2.8б). По выполнении параллельной программы выходные файлы доставляются на указанные в паспорте задачи подсистемы.

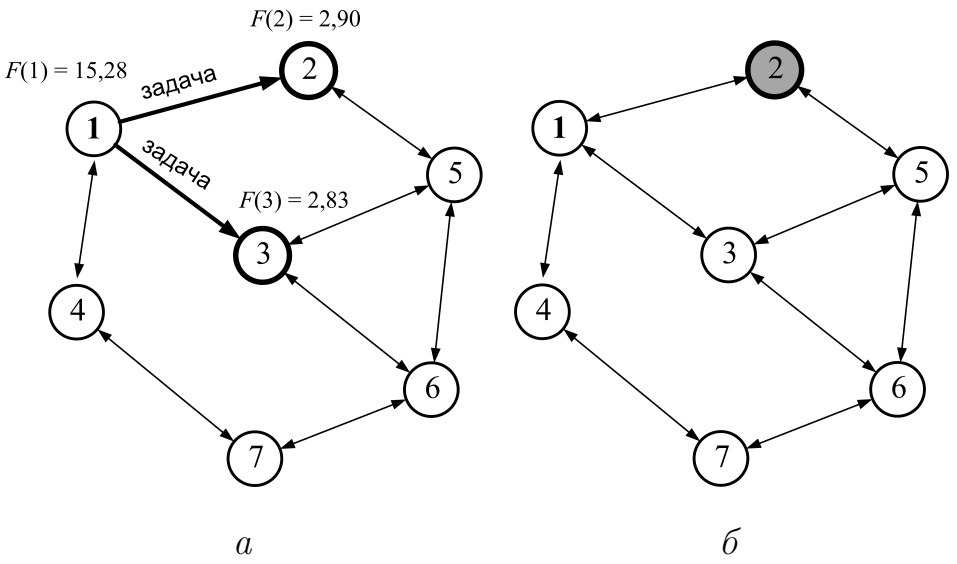



Рис. 2.8. Пример диспетчеризации задачи на мультикластерной ВС, алгоритм ДР

*Алгоритм ДМ.* Задача ставится в очередь подсистемы 3 с минимальным значением целевой функции (2,83) (рис. 2.9а). Выполняется доставка входных файлов с подсистем 5, 6. Допустим, что на подсистеме 3 нет достаточного для выполнения параллельной программы количества свободных ЭМ. Тогда через интервал времени $\Delta_2$ выполняется поиск новой подсистемы для задачи в очереди подсистемы 3 в локальной окрестности $S(3) = \{1, 3, 5, 6\}$. Предположим, значение целевой функции для подсистемы 5 меньше значения целевой функции для подсистемы 3, причём $F(3) - F(5) = 3,83 - 1,89 = 1,94 > \varepsilon = 1,0$, где $\varepsilon$ – критерий миграции, выбираемый эмпирически (рис. 2.9б). Тогда задача мигрирует на подсистему 5: задача удаляется из очереди подсистемы 3 и ставится в очередь подсистемы 5 (рис. 2.9в). Выполняется доставка входных файлов на подсистему 5. Предположим, на подсистеме 5 есть достаточное количество доступных ЭМ. В этом случае параллельная программа запускается на выполнение (рис. 2.9в). По завершении выполнения параллельной программы выходные файлы доставляются на указанные в паспорте задачи подсистемы.

*Алгоритм ДРМ.* Пусть значение параметра алгоритма $m = 2$. Тогда после вычисления значений целевой функции для подсистем множества $S(1)$,

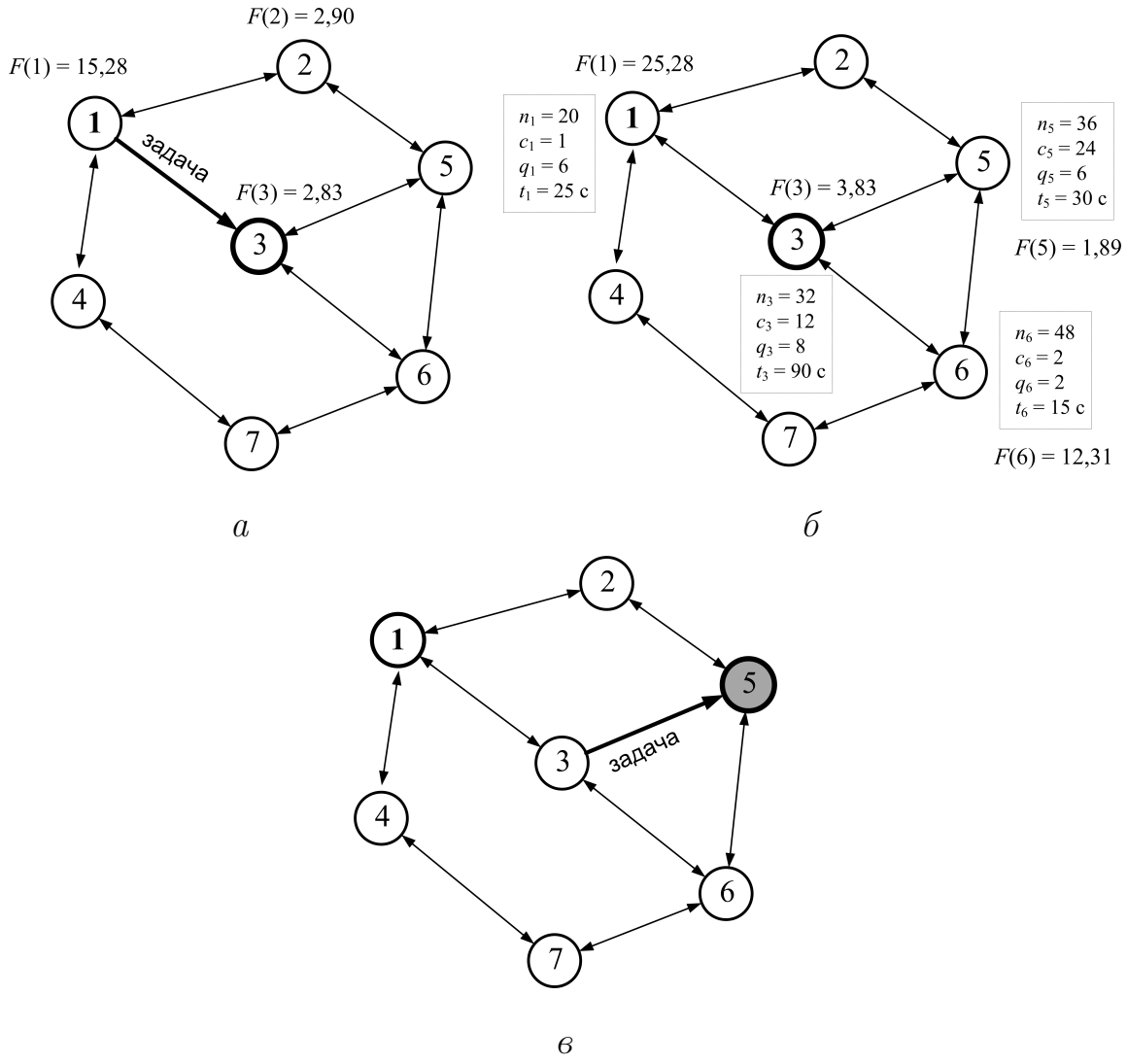



Рис. 2.9. Пример диспетчеризации задачи на мультикластерной ВС, алгоритм ДМ

задача ставится в очередь подсистем 2 и 3 с наименьшими значениями целевой функции (рис. 2.10а) (2,90 и 2,83, соответственно). Допустим, что на подсистемах 2 и 3 нет достаточного для выполнения параллельной программы количества свободных ЭМ. Тогда через интервал времени $\Delta_2$ выполняется поиск новой подсистемы для задач в очереди подсистемы 2 (локальная окрестность $S(2) = \{1, 2, 5\}$) и подсистемы 3 (локальная окрестность $S(3) = \{1, 3, 5, 6\}$). Предположим, в результате очередного выполнения процедуры поиска новой подсистемы для задачи в очереди подсистемы 3 значение целевой функции для подсистемы 5 оказалось меньше предыдущего значения, причём

$F(3) - F(5) = 3,83 - 1,89 = 1,94 > \varepsilon = 1,0$ (рис. 2.10б). В этом случае задача мигрирует на подсистему 5: задача удаляется из очереди подсистемы 3 и ставится в очередь подсистемы 5 (рис. 2.10в). Выполняется доставка входных файлов на подсистему 5. Предположим, на подсистеме 5 есть достаточное количество доступных ЭМ. Тогда на следующем шаге задача удаляется из очереди подсистемы 2 и параллельная программа запускается на выполнение на подсистеме 5 (рис. 2.10г). По завершении выполнения параллельной программы выходные файлы доставляются на указанные в паспорте задачи подсистемы.

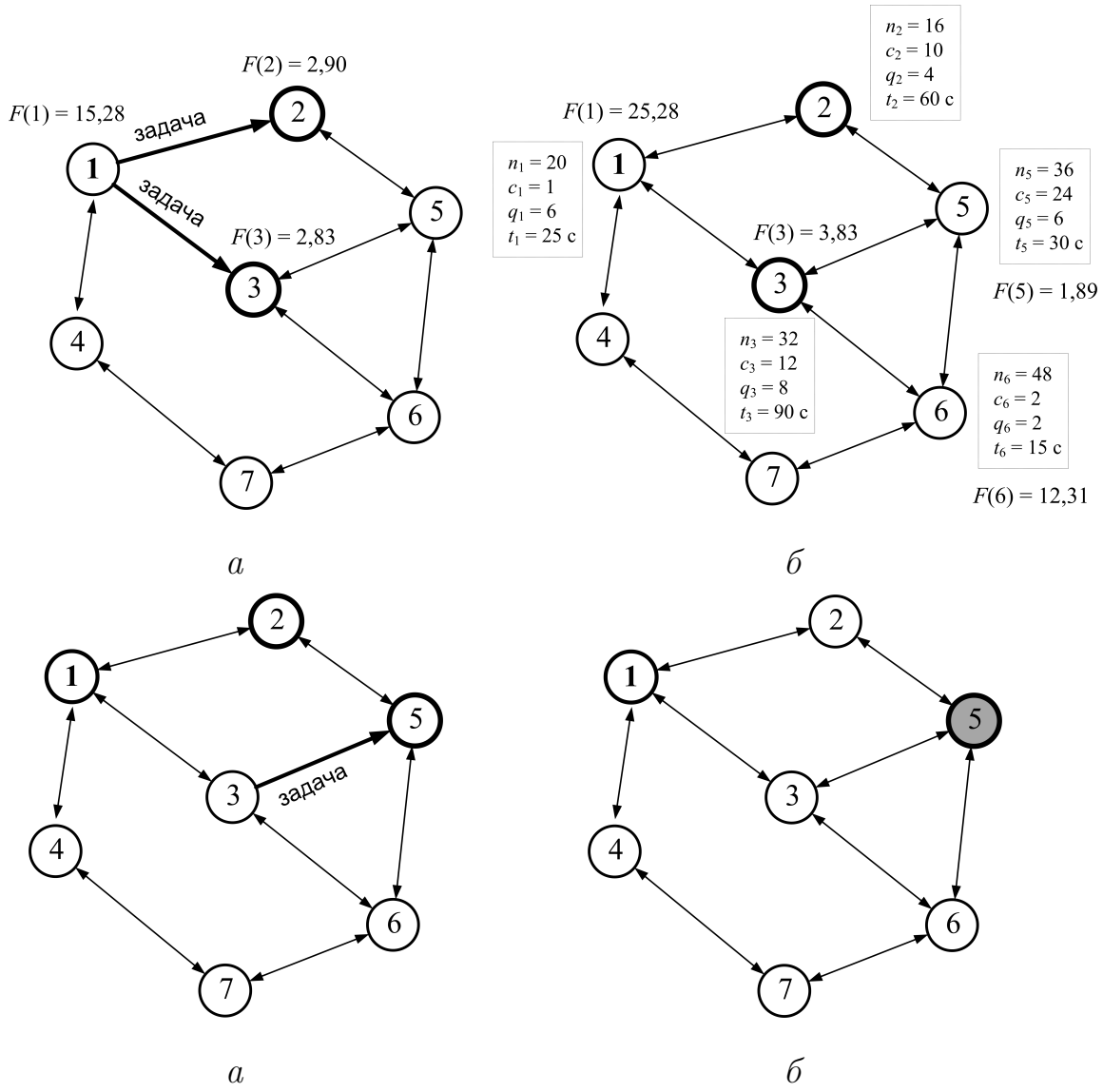



Рис. 2.10. Пример диспетчеризации задачи на мультикластерной ВС, алгоритм ДРМ

## 2.3. Моделирование алгоритмов децентрализованной диспетчеризации параллельных программ

### 2.3.1. Организация экспериментов

Исследование созданных алгоритмов проводилось на пространственно-распределённой мультикластерной ВС, созданной ЦПВТ ФГОБУ ВПО “СибГУТИ” совместно с Лабораторией вычислительных систем ИФП СО РАН (рис. 2.11). На каждом сегменте установлена операционная система GNU/Linux, локальная система управления ресурсами TORQUE 2.3.7, пакет Globus Toolkit 5.0, децентрализованный диспетчер GBroker, системы мониторинга DCSMon и NetMon. На сегменте 5 (Xeon80) настроен диспетчер GridWay 5.6.1 для управления ресурсами всех подсистем.

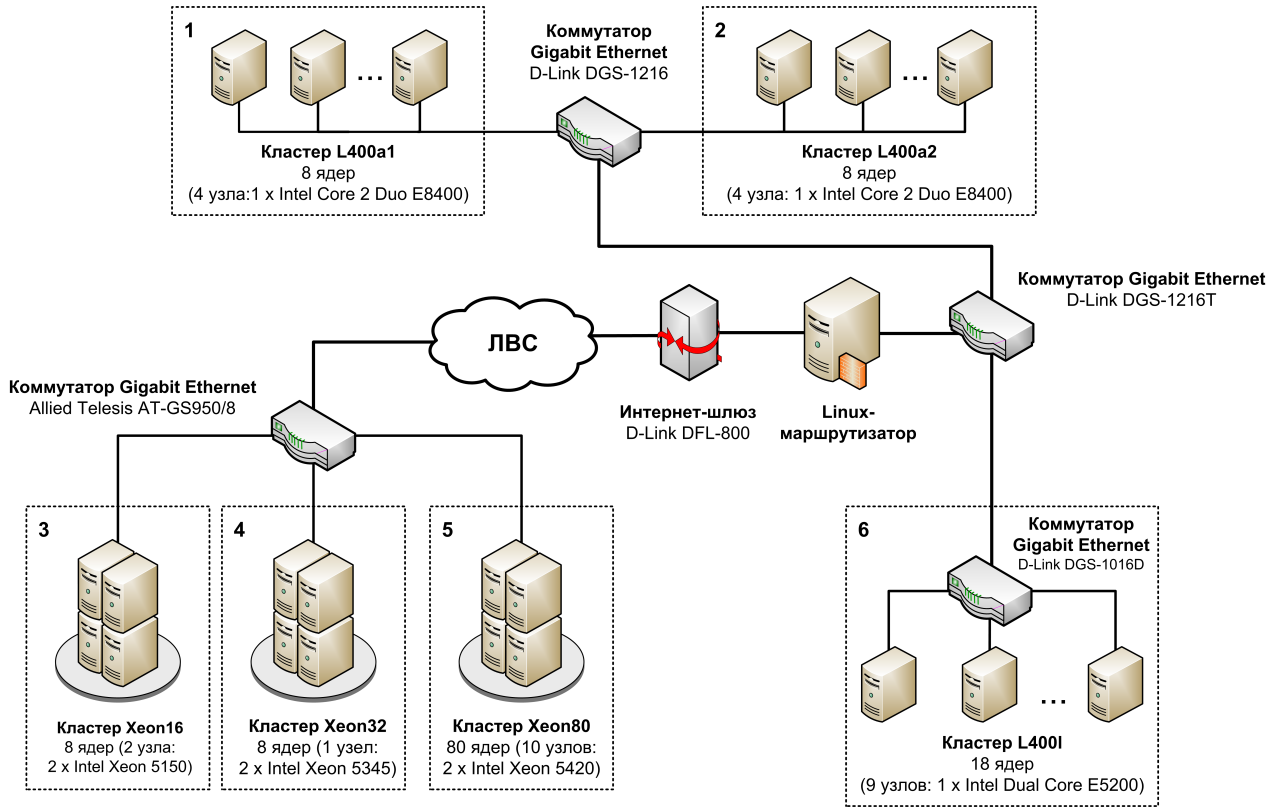



Рис. 2.11. Тестовая конфигурация мультикластерной ВС ($H = 6, N = 130$)

В качестве тестовых задач использовались MPI-программы из пакета тестов SPEC MPI 2007:

– Weather Research and Forecasting (WRF) – пакет моделирования климатических процессов;

– The Parallel Ocean Program (POP2) – пакет моделирования процессов в океане;

– LAMMPS – пакет решения задач молекулярной динамики;

– RAxML – пакет моделирования задач биоинформатики;

– Tachyon – пакет расчета графических сцен.

Входные данные для тестовых задач размещались на сегменте Xeon80; на эту же подсистему доставлялись результаты выполнения программ.

Генерировались простейшие потоки с различной интенсивностью $\lambda$ поступления задач, которые псевдослучайно с равномерным распределением выбирались из тестового набора. Каждый поток формировался из $M$ задач. Ранг $r$ каждой задачи выбирался из множества $\{1, 2, 4, 8\}$ псевдослучайно с равномерным распределением.

Обозначим через $t_k$ время поступления задачи $k \in 1, 2, \ldots, M$ на вход диспетчера, $t'_k$ – время начала решения задачи $k$, $t''_k$ – время завершения решения задачи $k$. Пусть $\tau$ – суммарное время обслуживания потока из $M$ задач.

Для оценки эффективности алгоритмов диспетчеризации использовались следующие показатели: пропускная способность $B$ системы, среднее время $T$ обслуживания задачи и среднее время $W$ пребывания задачи в очереди:

$$B = \frac{M}{\tau},\ T = \frac{1}{M}\sum_{k=1}^{M}(t''_k - t_k),\ W = \frac{1}{M}\sum_{k=1}^{M}(t'_k - t_k).$$

### 2.3.2. Сравнительный анализ алгоритмов децентрализованной диспетчеризации

На рис. 2.12 приведены результаты моделирования алгоритма ДЛО для различных целевых функций. Алгоритм ДЛО на всех целевых функциях демонстрирует аналогичные показатели среднего времени обслуживания задачи

и среднего времени ожидания задачи в очереди. Целевая функция $F$ обеспечивает незначительно большую пропускную способность по сравнению с $F'$ и $F''$ при возрастании интенсивности потока задач.

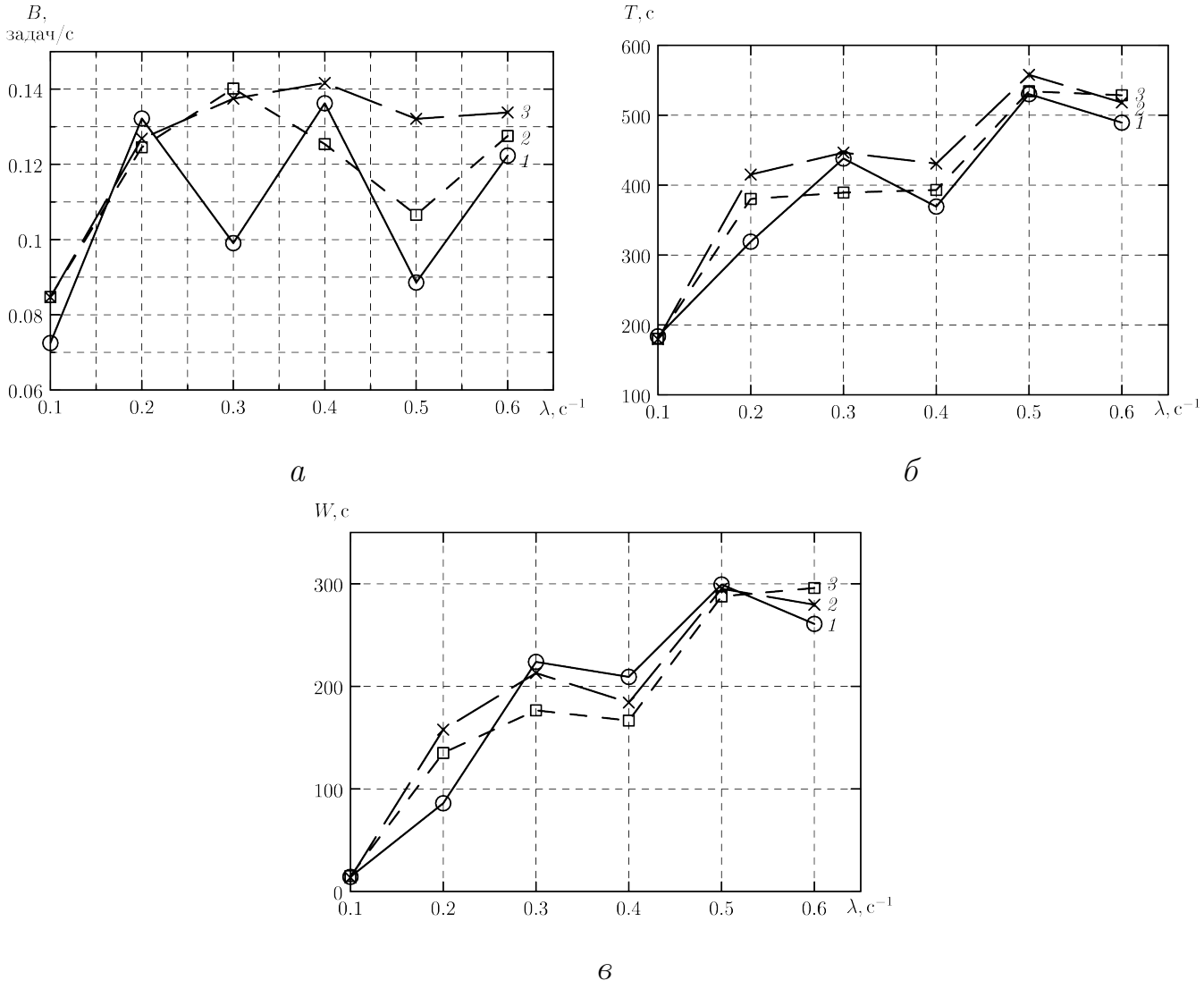



Рис. 2.12. Сравнение эффективности алгоритма ДЛО при различных целевых функциях
*1* – Функция $F''$; *2* – Функция $F'$, *3* – Функция $F$,

На рис. 2.13 и 2.14 приведены результаты сравнения эффективности алгоритмов ДЛО, ДР, ДМ и ДРМ при обслуживании потока из $M = 200$ задач. Поток задач поступал на подсистему Xeon80, локальные окрестности диспетчеров имели структуру полного графа.

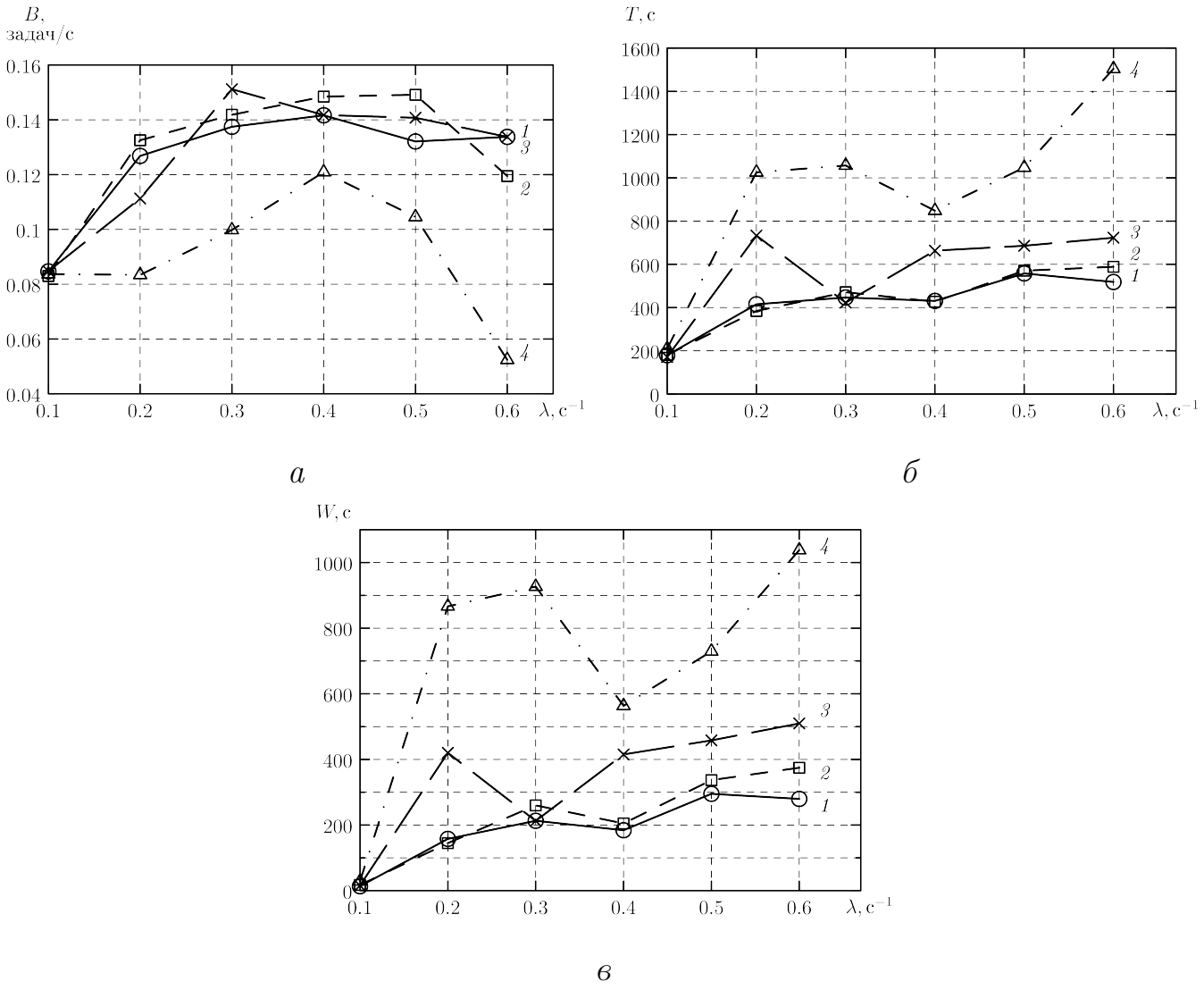



Рис. 2.13. Сравнение эффективности алгоритмов ДЛО и ДР:
*1* – Алгоритм ДЛО; *2* – Алгоритм ДР, $m = 2$; *3* – Алгоритм ДР, $m = 3$;
*4* – Алгоритм ДР, $m = 6$

Пропускная способность системы при использовании алгоритма ДР для $m \in \{2, 3\}$ выше пропускной способности при использовании алгоритма ДЛО. Деградация показателей при больших значениях $m$ связана с увеличением загрузки каналов связи при передаче входных файлов одной задачи на несколько сегментов.

В алгоритмах ДМ и ДРМ интервал поиска подсистемы $\Delta = 30$ с, условие миграции $\varepsilon = 0,2$. Наименьшие среднее время обслуживания задач и среднее время ожидания в очереди достигнуты при использовании алгоритмов ДЛО

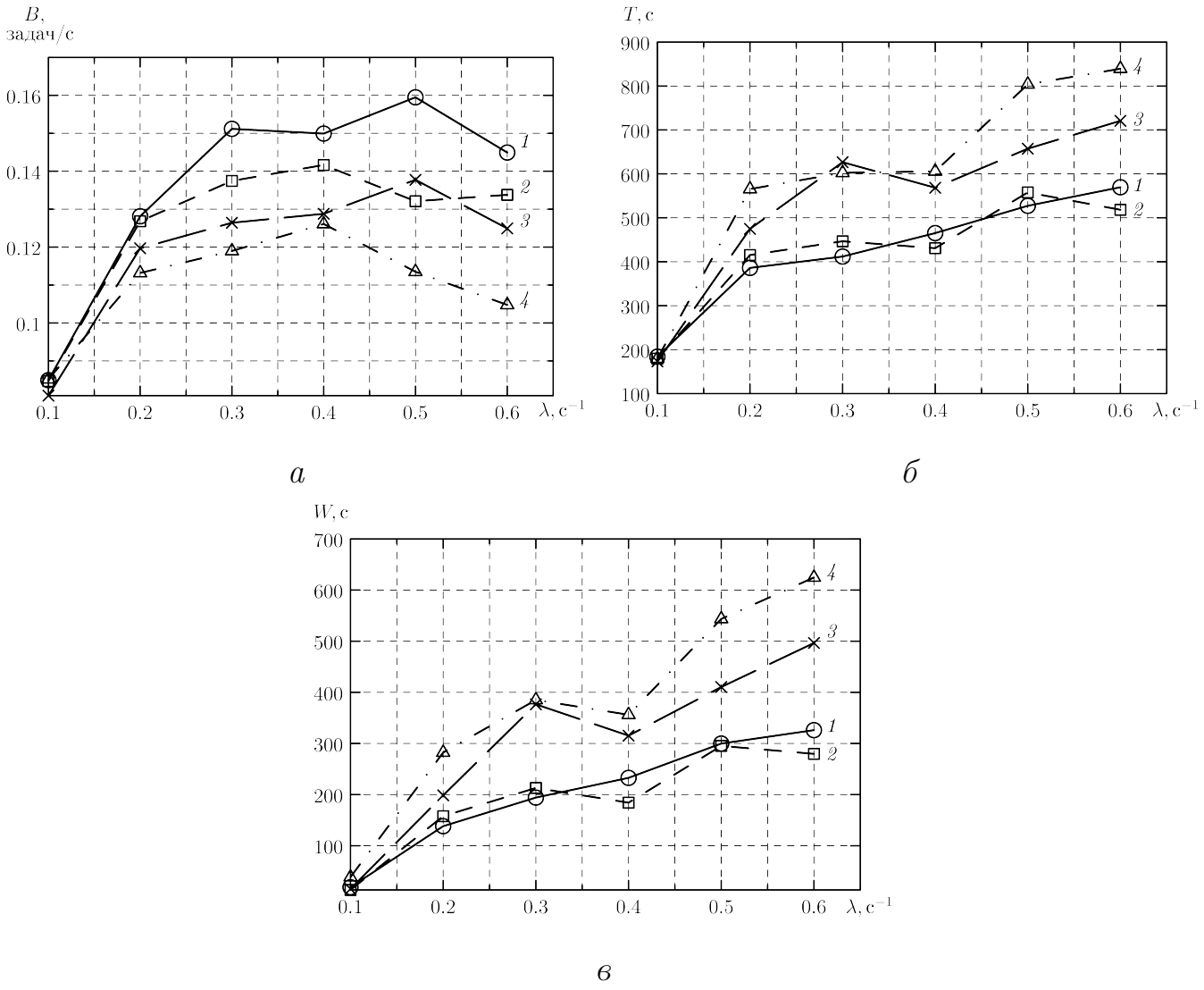



Рис. 2.14. Сравнение эффективности алгоритмов ДЛО, ДМ и ДРМ:
*1* – Алгоритм ДМ; *2* – Алгоритм ДЛО; *3* – Алгоритм ДРМ, $m = 2$;
*4* – Алгоритм ДРМ, $m = 3$

и ДМ (рис. 2.14), при этом большая пропускная способность системы получена для ДМ. Алгоритмы ДР и ДРМ рекомендуется применять в случае малой интенсивности потоков задач или небольших размеров входных данных.

При большой интенсивности потока задач значительно возрастает время доставки входных данных вследствие повышения загрузки каналов связи и сетевой файловой системы на сегментах. Это приводит к снижению пропускной способности системы и увеличению времени обслуживания задач (ожидания в очереди) для всех алгоритмов диспетчеризации.

### 2.3.3. Экспериментальное сравнение с методами централизованной диспетчеризации

Выполнено сравнение эффективности обслуживания потоков задач централизованным диспетчером GridWay и созданным децентрализованным пакетом GBroker.

Диспетчер GridWay был установлен на подсистеме Xeon80. При его конфигурации использовались параметры, рекомендованные разработчиками:

SCHEDULING_INTERVAL (интервал диспетчеризации) = 5 с;

DISCOVERY_INTERVAL (интервал обнаружения ресурсов) = 100 с;

MONITORING_INTERVAL (интервал опроса состояния ресурсов) = 2 с;

POLL_INTERVAL (интервал опроса состояния задач) = 10 с;

DISPATCH_CHUNK (число задач, для которых одновременно выполняется диспетчеризация) = 15.

Для диспетчера GBroker проведено два эксперимента. В ходе первого на подсистему Xeon80 поступал поток из $M = 300$ задач. Во втором эксперименте моделировалось распределённое обслуживание задач: одинаковые потоки из $M = 50$ задач одновременно поступали в очереди диспетчеров всех 6 подсистем. При этом в качестве структур логических связей диспетчеров использовались $2D$-тор и полный граф, а в качестве алгоритма диспетчеризации – ДМ. Пакет GridWay установлен на сегменте Xeon80 и настроен в соответствии с рекомендациями разработчиков.

На рис. 2.15 видно, что пропускная способность диспетчера GBroker при обслуживании нескольких потоков превосходит пропускную способность пакета GridWay. Среднее время обслуживания и среднее время пребывания задач в очереди близки с GridWay и незначительно возрастают при централизованном обслуживании.

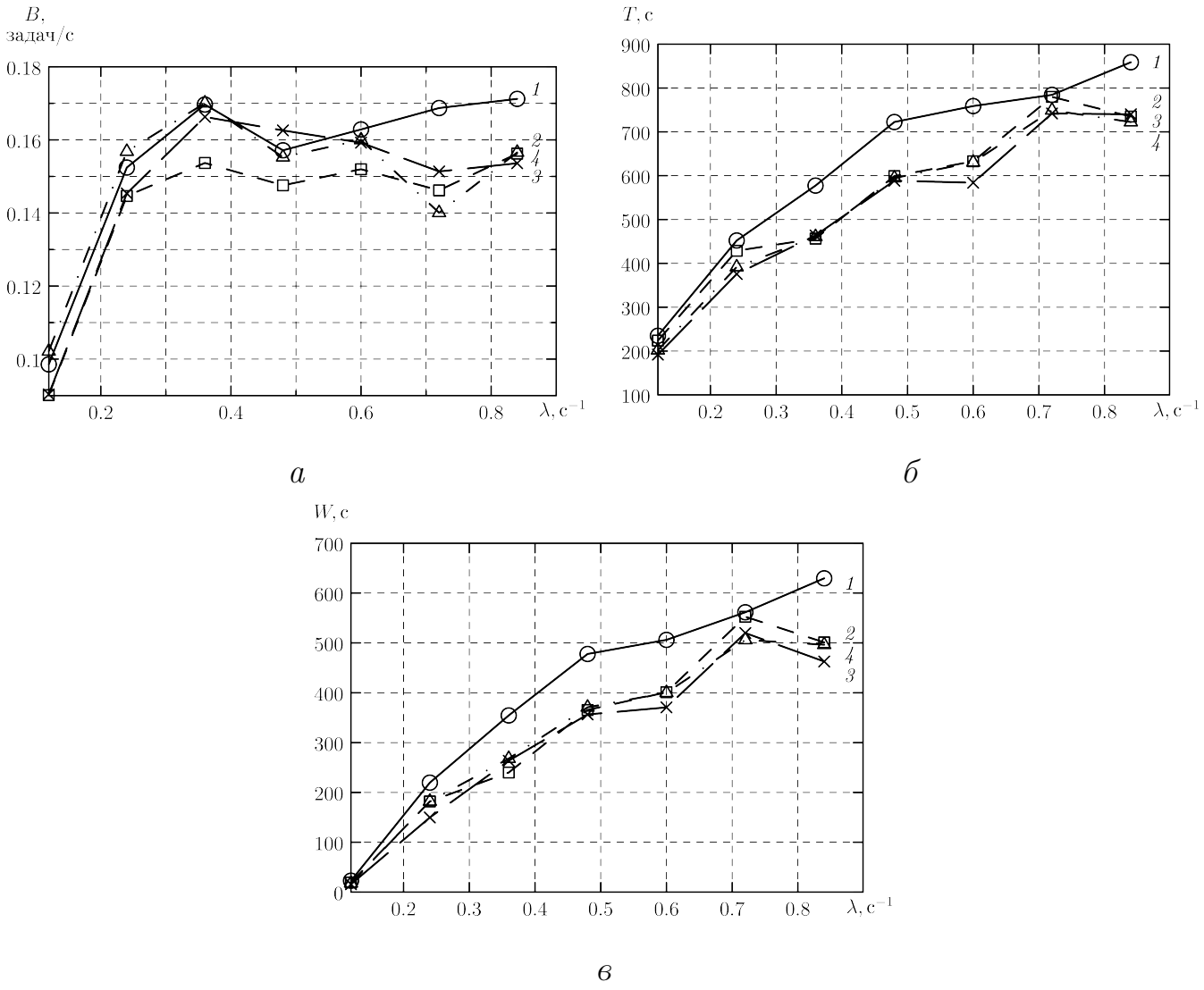



Рис. 2.15. Сравнение эффективности диспетчеров GBroker и GridWay:
*1* – GBroker; *2* – GBroker, полный граф; *3* – GBroker, $2D$-тор; *4* – GridWay

## 2.4. Выбор логических структур локальных окрестностей диспетчеров

### 2.4.1. Анализ логических структур локальных окрестностей диспетчеров

Выполнено исследование влияния структуры локальных окрестностей диспетчеров на эффективность диспетчеризации. В эксперименте на вход всех диспетчеров одновременно поступали потоки из $M = 50$ задач. Рассматривались

локальные окрестности в виде полных графов, колец, решёток, звёзд, $2D$-торов и $D_2$-графов [2] (рис. 2.16).

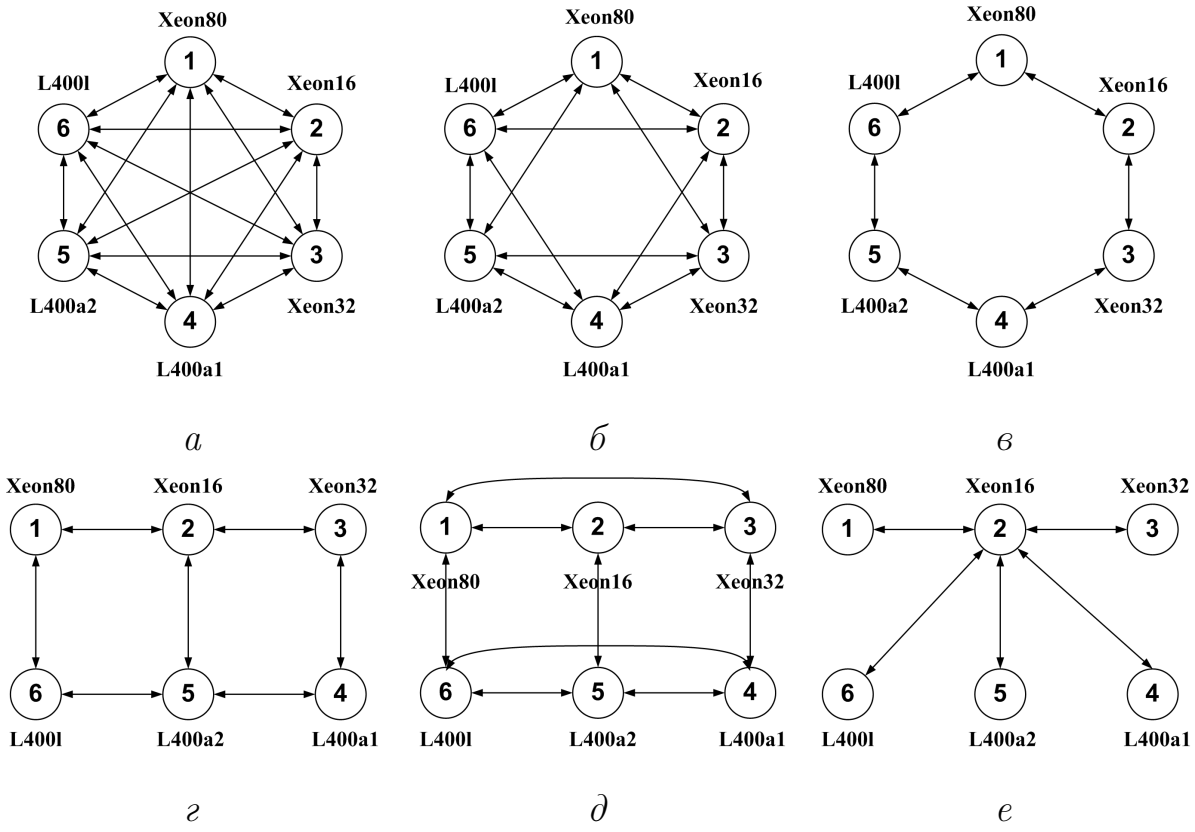



Рис. 2.16. Тестовая конфигурация мультикластерной ВС ($H = 6, N = 130$)
$а$ – полный граф, $б$ – $D2(6; 2, 3)$, $в$ – кольцо,
$г$ – решетка, $д$ – двумерный тор, $е$ – звезда

На рис. 2.17 показано влияние структуры локальных окрестностей диспетчеров на эффективность обслуживания потоков задач алгоритмом ДМ. Высокие значения пропускной способности, помимо полносвязной структуры, были получены для конфигураций на основе $2D$-тора, решётки и $D_2$-графа, при этом для двух последних были достигнуты наибольшие значения. Использование неполносвязных структур при формировании локальных окрестностей диспетчеров не приводит к значительному снижению показателей эффективности диспетчеризации. Такие структуры могут быть образованы при отсутствии прямых линий связи между отдельными подсистемами, например в случае отказов некоторых подсистем. В качестве структур локальных окрестностей мо-

гут быть рекомендованы торы или $D_n$-графы, которые обладают малым (средним) диаметром.

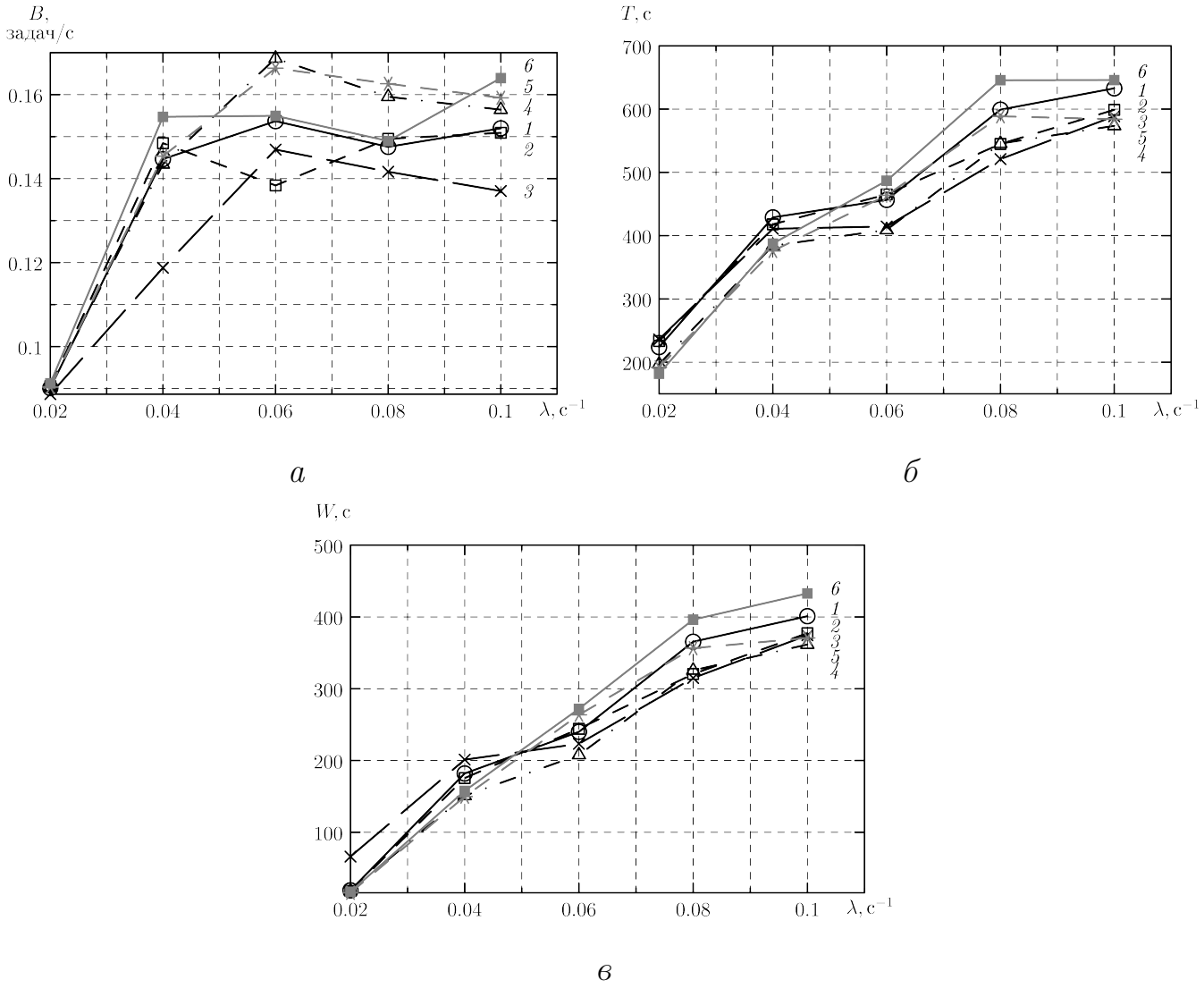



Рис. 2.17. Влияние структур локальных окрестностей диспетчеров на эффективность диспетчеризации:

*1* – полный граф; *2* – звезда; *3* – кольцо; *4* – решётка; *5* - $2D$-тор; *6* - $D_2$-граф

### 2.4.2. Алгоритмы поиска субоптимальных локальных окрестностей диспетчеров

При децентрализованной диспетчеризации каждый диспетчер взаимодействует с ограниченным числом других диспетчеров, формирующих его локальную окрестность. От выбора локальных окрестностей зависит эффективность

функционирования диспетчеров. Важной задачей организации функционирования коллектива диспетчеров является формирование логических структур локальных окрестностей, минимизирующих время обслуживания параллельных задач.

При выборе локальных окрестностей децентрализованных диспетчеров могут быть использованы известные перспективные структуры, примеры которых приведены выше. Однако главным недостатком таких структур является их однородность. Известно, что в пространственно-распределённых ВС загрузка подсистем может существенно различаться и может изменяться с течением времени. Поэтому неоднородные логические структуры локальных окрестностей могут быть более эффективны для пространственно-распределённых ВС и поэтому актуальным является создание алгоритмов поиска таких структур.

В диссертации предлагается стохастический алгоритм субоптимального формирования локальных окрестностей диспетчеров с целью минимизации функции штрафа при обслуживании потоков задач.

Рассматривается пространственно-распределённая ВС, состоящая из $H$ вычислительных подсистем, на каждой из которых функционирует диспетчер. Коллектив диспетчеров представлен в виде графа $G = (V, E)$, где вершинам соответствуют диспетчеры, а рёбрам – логические связи между ними (рис. 2.18). Наличие дуги $(i, j) \in E$ означает, что диспетчер $i$ может отправлять задачи своей очереди диспетчеру $j$. Множество диспетчеров $j$, смежных с диспетчером $i$, образует его локальную окрестность $L(i)$.

Введём обозначения: $c_i$ – стоимость использования подсистемы в единицу времени, $\lambda_i$ – интенсивность потока поступления задач на подсистему $i$, $\mu_{ij}$ – интенсивность потока миграции задач с подсистемы $i$ на подсистему $j$. Пусть $x_{ij} = \{0, 1\}$ – наличие дуги от подсистемы $i$ к $j$: $x_{ij} = 1$, если существует дуга $(i, j) \in E$, иначе $x_{ij} = 0$. Значения $x_{ij}$ представлены в виде матрицы $X = \{x_{ij} : i \in V, j \in C\}$.

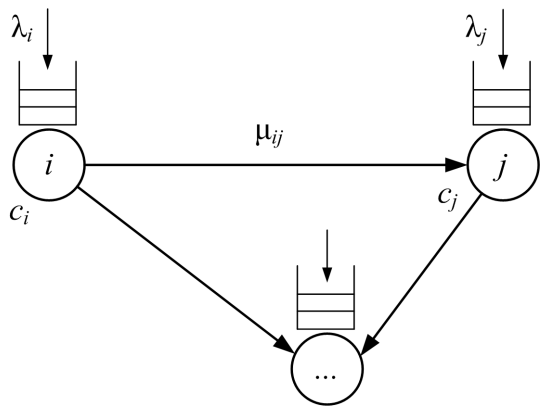



Рис. 2.18. Модель пространственно-распределённой мультикластерной ВС

Количество задач, мигрировавших с подсистемы $i$: $\sum\limits_{j=1...H} x_{ij}\mu_{ij}$. Время обслуживания мигрировавших задач складывается из времени $t_i$ обслуживания задачи на первой назначенной подсистеме и штрафа $kt_i$ за миграцию ($k$ – коэффициент штрафа за миграцию). Тогда стоимость обслуживания задач, мигрировавших с подсистемы $i$: $\sum\limits_{j=1...H} x_{ij}\mu_{ij}c_j(t_i + kt_i)$. Стоимость решения не мигрировавших задач на подсистеме $i$: $(\lambda_i - \sum\limits_{j=1...H} x_{ij}\mu_{ij})c_i t_i$.

Таким образом, суммарная стоимость обслуживания задач на подсистеме $i$:

$$(\lambda_i - \sum_{j=1...H} x_{ij}\mu_{ij})c_i t_i + \sum_{j=1...H} x_{ij}\mu_{ij}c_j(t_i + kt_i)$$

Сформулируем задачу поиска оптимальных структур локальных окрестностей диспетчеров. Необходимо минимизировать функцию штрафа за обслуживание задач на пространственно-распределённой ВС:

$$F(X) = \sum_{i=1...H} ((\lambda_i - \sum_{j=1...H} x_{ij}\mu_{ij})c_i t_i + \sum_{j=1...H} x_{ij}\mu_{ij}c_j(t_i + kt_i)) \to \min_{(x_{ij})} \qquad (2.5)$$

при ограничениях

$$x_{ij} = \{0, 1\} \qquad (2.6)$$

$$\sum_{j=1...H} x_{ij} \le l, \quad i = 1, 2, \ldots, H \qquad (2.7)$$

где $l$ – максимально допустимый размер локальной окрестности диспетчеров.

Задача (2.5)–(2.7) относится к целочисленной оптимизации. Целесообразным является разработка эвристических алгоритмов поиска субоптимальных локальных окрестностей диспетчеров.

На основе метода цепей Монте-Карло [112] предложен эвристический алгоритм $LN\_MCC$ [113] формирования субоптимальных локальных окрестностей диспетчеров. В листинге 2.1 представлен псевдокод алгоритма.

На первом шаге "жадным" алгоритмом генерируется начальное решение (процедура $init()$). Далее случайным образом генерируется решение на расстоянии $d$ от текущего лучшего решения $x^*$. Расстояние между решениями – количество различающихся переменных $x_{ij}$ в текущем и предыдущем решениях. После генерации выполняется сравнение целевых функций сгенерированного и лучшего решений. Если после $i_{\max}$ генераций не найдено очередное лучшее решение, расстояние $d$ уменьшается в два раза. Алгоритм выполняется до тех пор, пока расстояние $d$ больше минимального расстояния $d_{\min}$ и пока не найдено хотя бы одно субоптимальное решение для текущего базового.

Листинг 2.1. Псевдокод алгоритма *LN_ MCC*

*LN_ MCC*$(\lambda_i, \mu_{ij}, c_i, t_i)$

Входные данные: $\lambda_i$ – интенсивности потоков поступления задач на подсистему (для каждой подсистемы $i \in V$), $\mu_{ij}$ – интенсивности потоков миграции задачи с подсистемы $i$ на подсистему $j$ (для каждой пары $i, j \in V$), $c_i$ – стоимость использование подсистемы $i$ в единицу времени; $t_i$ – среднее время обслуживания задачи на подсистеме $i$.

Выходные данные: матрица значений $X$; $x_{ij} = 1$, если существует дуга $(i, j) \in E$, иначе $x_{ij} = 0$.

```
1   x* ← init()        /* сформировать начальное решение */
2   do
3       d ← d_0
4       while x* not found and d > d_min do
5           for i ← 0 to i_max do
6               x ← gen_sol(x*, d)          /* сгенерировать решение */
7               if f(x, λ, μ, c, t) < t(x*, λ, μ, c, t) then
8                   x* ← x
9                   break
10              end if
11          end for
12      d ← d/2             /* уменьшить расстояние */
13      end while
14  while new x* not found
```

Моделирование проводилось на подсистеме мультикластерной ВС ЦПВТ ФГОБУ ВПО “СибГУТИ”. Процессор – Intel Xeon E5420 (2.5 ГГц), 8 Гб ОЗУ. Выбраны следующие параметры алгоритма $LN\_MCC$:

– начальное расстояние между решениями $d_0 = 100$,

– количество итераций для каждого расстояния $i_{\max} = 1000$,

– максимальный размер локальной окрестности $l = 10$,

– коэффициент штрафа за миграцию задачи $k = 0,2$,

– стоимости использования подсистемы $c_i \in U(0,1;1)$ ($c_i$ – случайная величина, равномерно распределённая в интервале $(0,1;1)$),

– интенсивности потоков задач $\lambda_i \in U(0;100)$,

– интенсивности потоков миграции задач $\mu_{ij} \in U(0;20)$,

– среднее время решения задачи $t_i \in U(0;200)$.

На рис. 2.19 представлено сравнение логических структур локальных окрестностей, полученных с помощью алгоритма $LN\_MCC$, и распространённых перспективных структур. На рис. 2.21–2.24 приведены графы, полученные с помощью алгоритма $LN\_MCC$, и известные перспективные структуры, а также соответствующие графам значения целевой функции. Логическая структура полного графа (не приведена на рис. 2.19б, 2.19в, 2.19г) значительно превосходит значения функции штрафа для остальных структур. Структуры решётка, $2D$-тор и $D_2$-граф демонстрируют сопоставимые результаты. Логические структуры локальных окрестностей, сформированные алгоритмом $LN\_MCC$, характеризуются величиной штрафа в 2,5-3,0 раза меньшей по сравнению с другими логическими структурами. Это объясняется тем, что пространственно-распределённые ВС являются гетерогенными: интенсивности потоков поступления задач на подсистемы и миграции задач между подсистемами существенно различаются. Основные перспективные структуры (гиперкубы, $ND$-торы, $D_2$-графы и т.д.) являются однородными и не позволяют учитывать дисбаланс загрузки подсистем. Это ограничивает их применение в системах децентрализованной диспетчеризации пространственно-распределённых ВС.

Использование неоднородных структур, получаемых с помощью алгоритма $LN\_MCC$, позволяет снизить время выполнения параллельных программ и повысить технико-экономическую эффективность эксплуатации мультикластерных и GRID-систем. Для этого в процессе функционирования пространственно-распределённой ВС периодически запускается процедура $LN\_MCC$, которая на основе текущей информации о загрузке подсистем формирует субоптимальные локальные окрестности децентрализованных диспетчеров.

Время работы алгоритма (приведено на рис. 2.20) является приемлемым для большемасштабных систем.

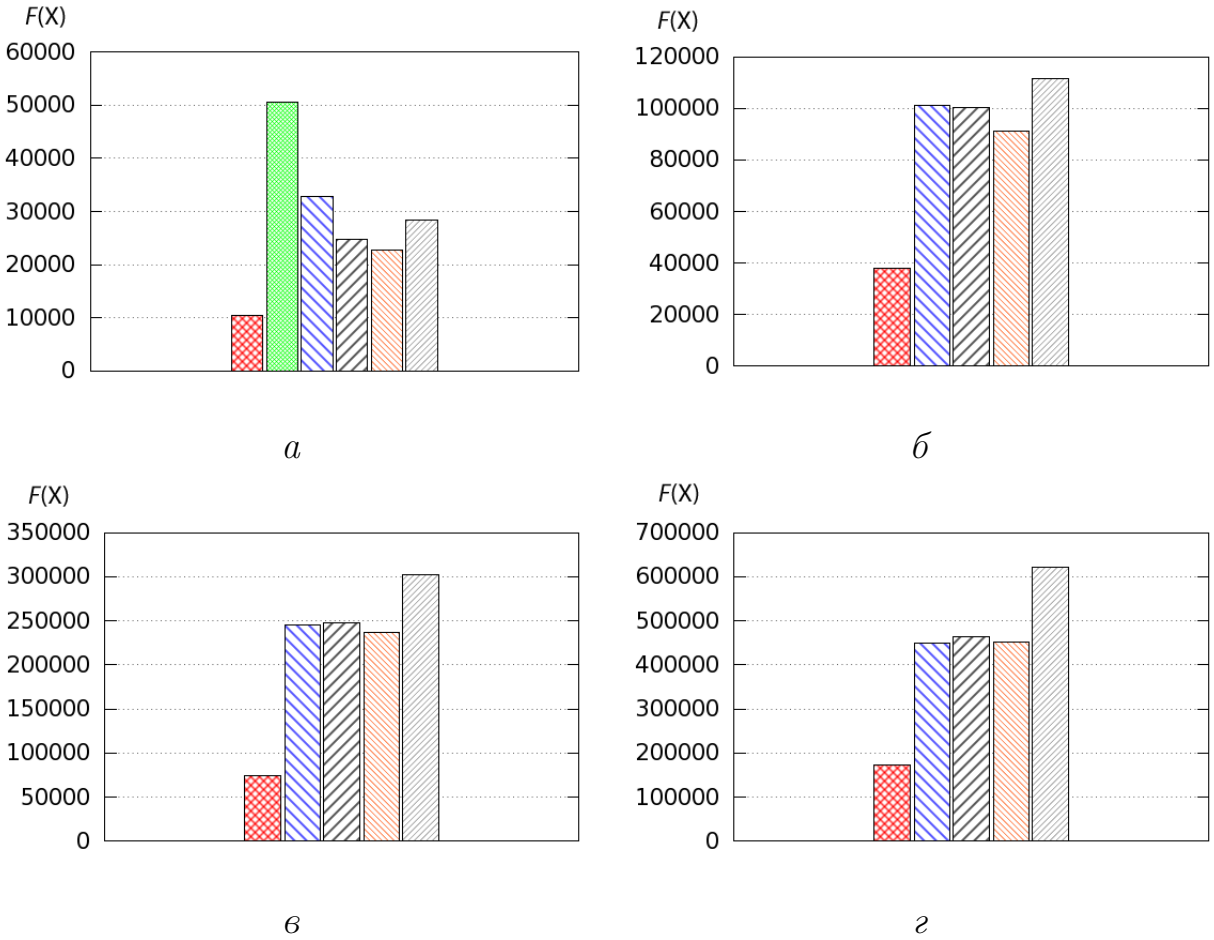



Рис. 2.19. Сравнение логических структур локальных окрестностей диспетчеров

$а$ – $H = 8$, $б$ – $H = 32$, $в$ – $H = 64$, $г$ – $H = 128$

– $LN\_MCC$, – полный граф, – $D_2$-граф,

– $2D$-тор, – решётка, – гиперкуб

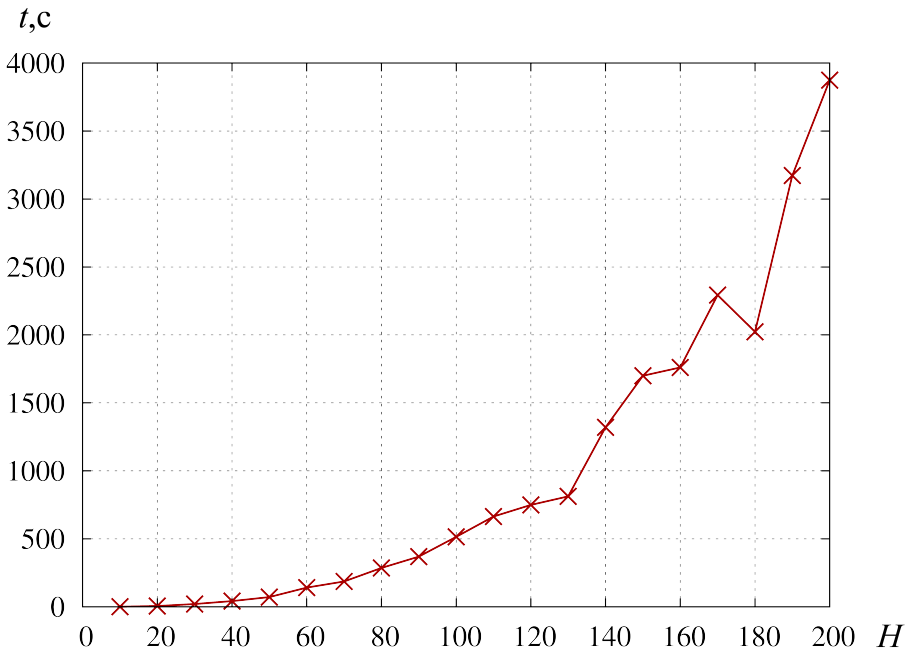


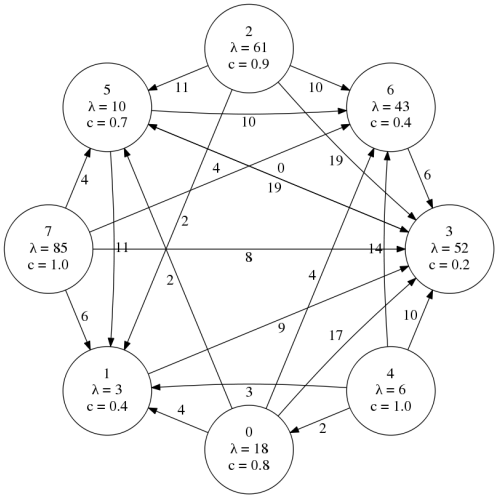

Рис. 2.20. Время работы алгоритма $LN\_MCC$
в зависимости от числа $H$ подсистем

## 2.5. Выводы

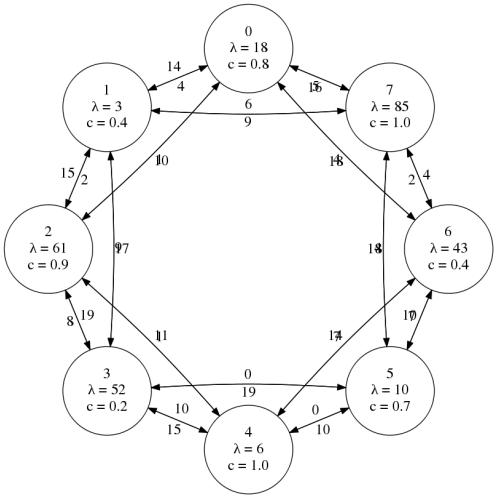
1. Централизованные системы диспетчеризации задач в большемасштабных распределённых вычислительных системах характеризуются вычислительной сложностью поиска требуемых ресурсов. Децентрализованная диспетчеризация существенно проще централизованной и позволяет повысить живучесть распределённых ВС.
2. Экспериментально показано, что среднее время обслуживания потоков задач и при децентрализованной, и при централизованной диспетчеризации сопоставимы. Время диспетчеризации достаточно мало по сравнению со временем выполнения задач.
3. Предложенные алгоритмы децентрализованной диспетчеризации позволяют учитывать динамический характер состава и загрузки ВС. Алгоритм диспетчеризации на основе миграции (ДМ) обеспечивает наименьшее среднее время обслуживания задач и наибольшую пропускную способность системы. Алгоритм на основе репликации задач (ДР) эффекти-

вен при небольшом числе подсистем, в очередь которых одновременно ставится задача.

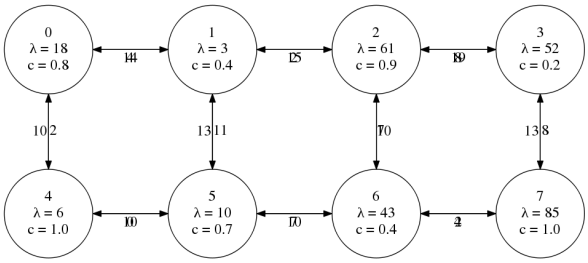
4. Предложенный эвристический алгоритм формирования субоптимальных локальных окрестностей децентрализованных диспетчеров, основанный на методе цепей Монте-Карло, обеспечивает минимизацию функции штрафа при обслуживании потока параллельных задач в 2,5-3 раза по сравнению с известными перспективными структурами.



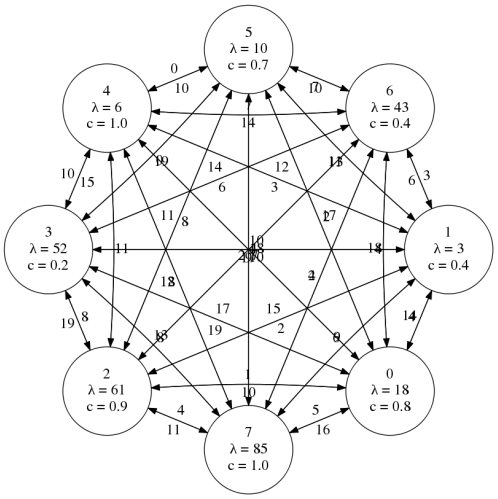

*а*



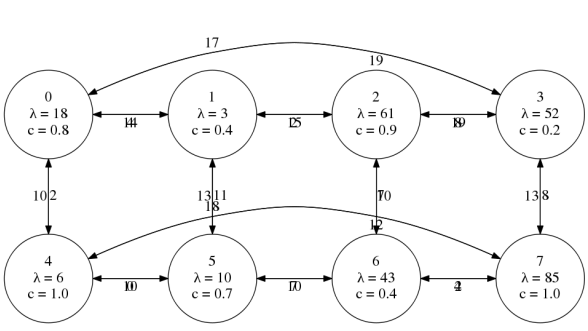

*б*



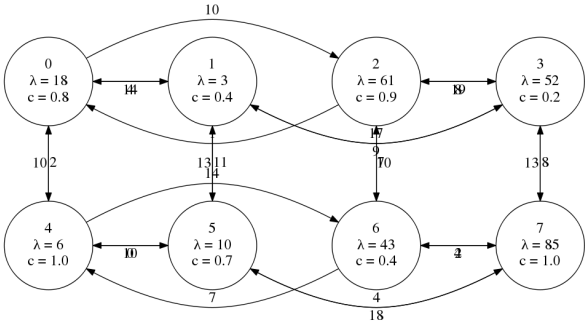

*в*




*г*



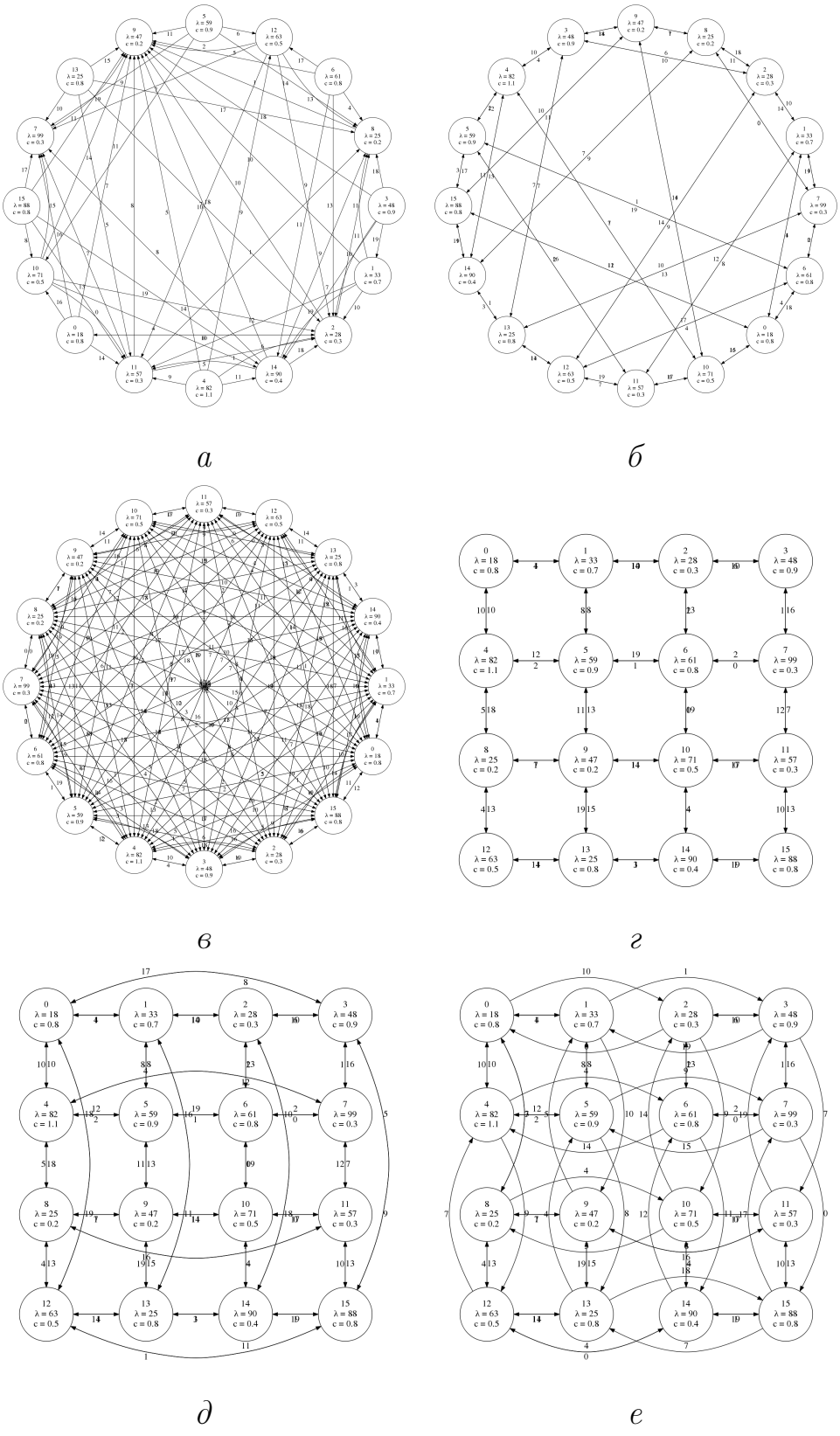

*д*




*е*

Рис. 2.21. Значения целевой функции при различных логических структурах локальных окрестностей ($H = 8$)

*а* – *LN_MCC*, 74466, *б* – $D_2\{8; 1, 2\}$, 245395, *в* – полный граф, 2777915, *г* – решетка, 236521, *д* – двумерный тор, 248186, *е* – $4D$-куб, 302453



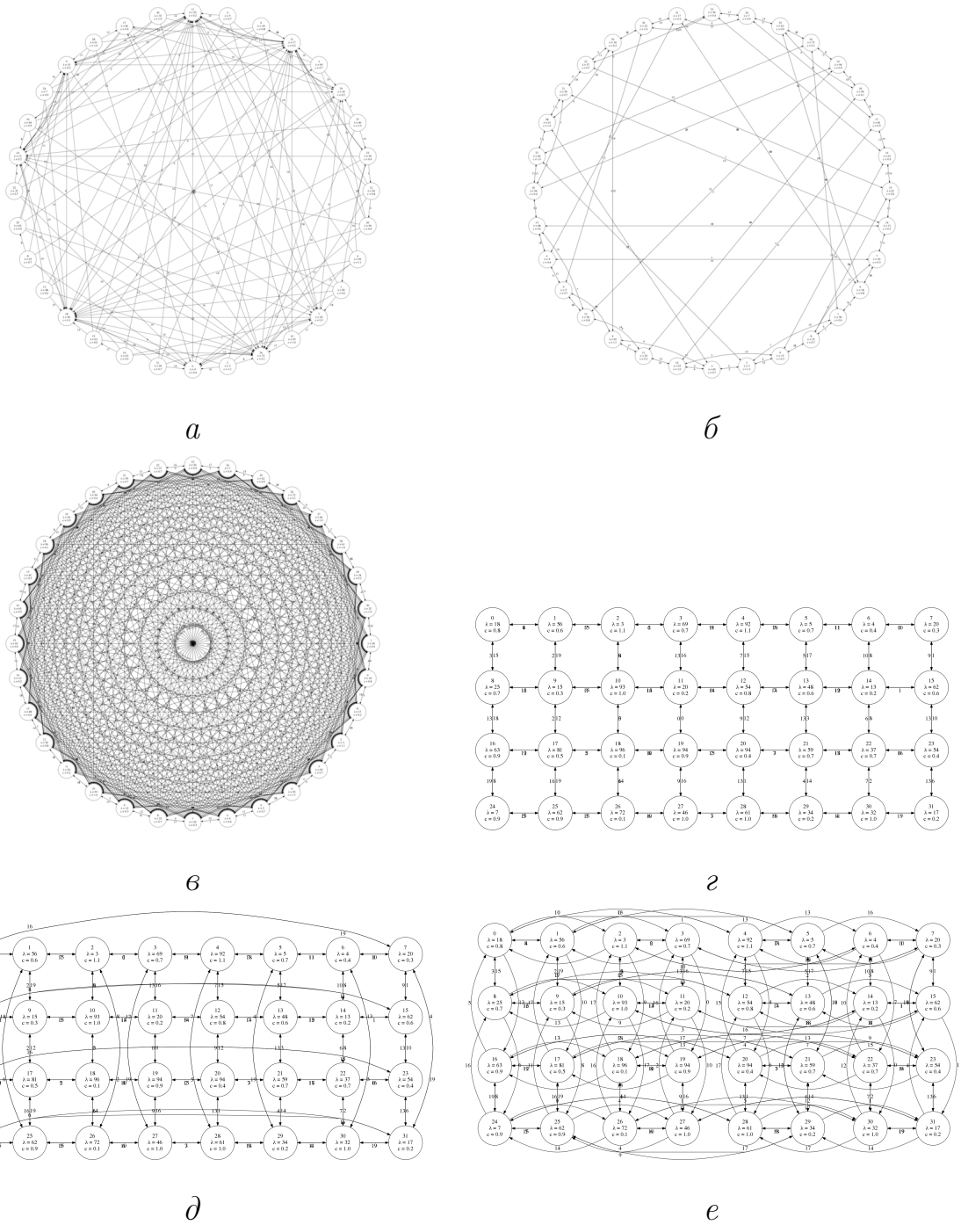
*а* *б*

*в* *г*

*д* *е*

Рис. 2.22. Значения целевой функции при различных логических структурах локальных окрестностей ($H = 16$)

*а* – *LN_MCC*, 25896, *б* – $D_2\{16; 1, 6\}$, 56617, *в* – полный граф, 160333, *г* – решетка, 48820, *д* – двумерный тор, 59260, *е* – $4D$-куб, 59896



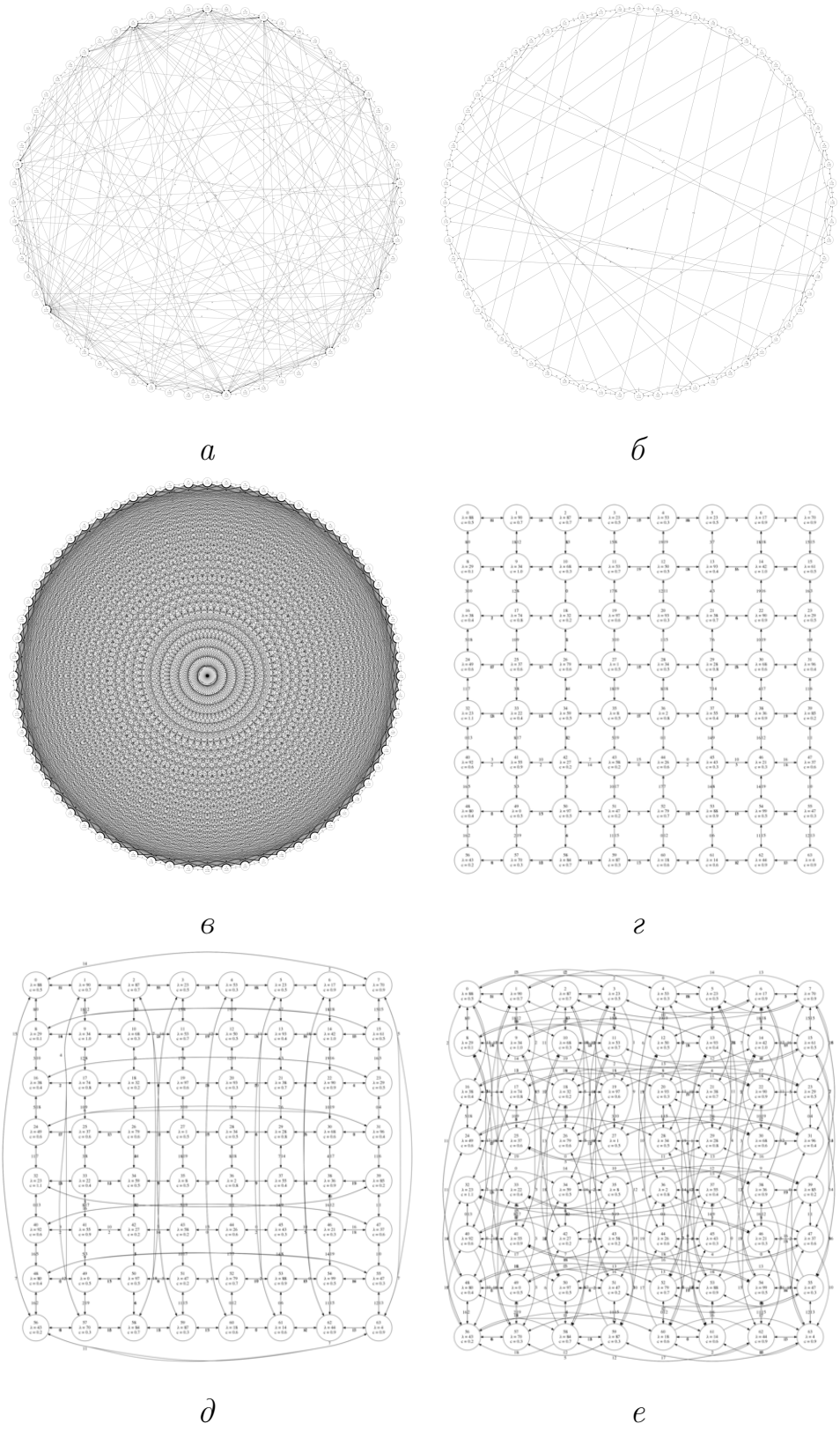
*а* *б*

*в* *г*

*д* *е*

Рис. 2.23. Значения целевой функции при различных логических структурах локальных окрестностей ($H = 32$)

*а* – $LN\_MCC$, 37857, *б* – $D_2\{32; 1, 7\}$, 101345, *в* – полный граф, 619877, *г* – решетка, 91233, *д* – двумерный тор, 100168, *е* – $5D$-куб, 111570



*а* *б*

*в* *г*

*д* *е*

Рис. 2.24. Значения целевой функции при различных логических структурах локальных окрестностей ($H = 64$)

*а* – $LN\_MCC$, 74466, *б* – $D_2\{64; 1, 14\}$, 245395, *в* – полный граф, 2777915, *г* – 236521, 91233, *д* – двумерный тор, 248186, *е* – $6D$-куб, 302453

# Глава 3. Вложение параллельных программ в иерархические пространственно-распределённые ВС

## 3.1. Задача оптимального вложения параллельных программ

### 3.1.1. Обзор алгоритмов вложения параллельных программ в пространственно-распределённые ВС

Параллельные программы для пространственно-распределённых ВС преимущественно разрабатываются в модели передачи сообщений, как правило, с использованием библиотек стандарта MPI (PACX [114], MPICH-G2 [115], GridMPI [116], MCMPI [117], Stampi [118], NumGRID [119]) или специализированных пакетов распределённых вычислений (X-Com [120], BOINC [121]).

Одной из важных проблем организации функционирования пространственно-распределённых ВС является оптимизация вложения в них параллельных программ (Task mapping, task allocation, task assignment) [122]. Под оптимальным вложением понимается такое распределение ветвей параллельной программы по процессорным ядрам ВС, при котором достигается минимум накладных расходов на межмашинные обмены информацией.

Коммуникационные среды пространственно-распределённых ВС имеют иерархическую организацию. Как правило, в них можно выделить минимум три уровня: первый уровень – сеть связи между подсистемами (кластерами); второй уровень – сеть связи между вычислительными узлами отдельной подсистемы (кластера); третий уровень – общая память вычислительных узлов. Размещение процессорных ядер в системе существенно влияет на время передачи информации между ними [123]. Время выполнения информационных

обменов между параллельными ветвями варьируется в зависимости от того, на какие процессорные ядра они назначены.

На рис. 3.1 представлен пример иерархической организации коммуникационной среды вычислительной подсистемы (кластерной ВС). Структура всей мультикластерной ВС может быть представлена в виде дерева, содержащего $L$ уровней (рис. 3.2). Каждый уровень системы образован отдельным видом структурных элементов системы (например, телекоммуникационные шкафы, вычислительные узлы и т. п.), которые объединены каналами связи своего уровня. Производительность каналов связи существенно различается. Характерной особенностью мультикластерных ВС является наличие медленных каналов связи между подсистемами.

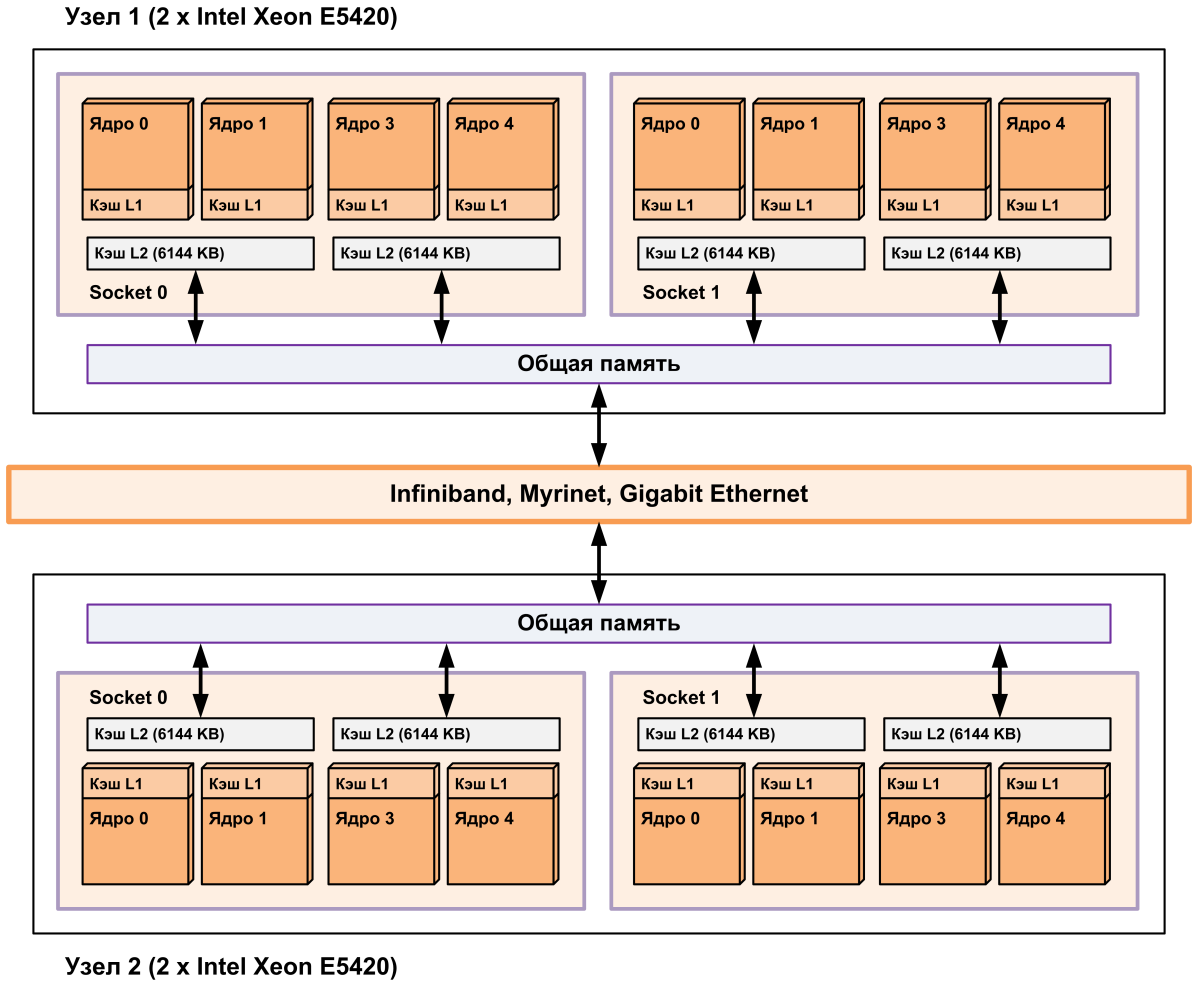



Рис. 3.1. Пример иерархической организации коммуникационной среды кластерной ВС

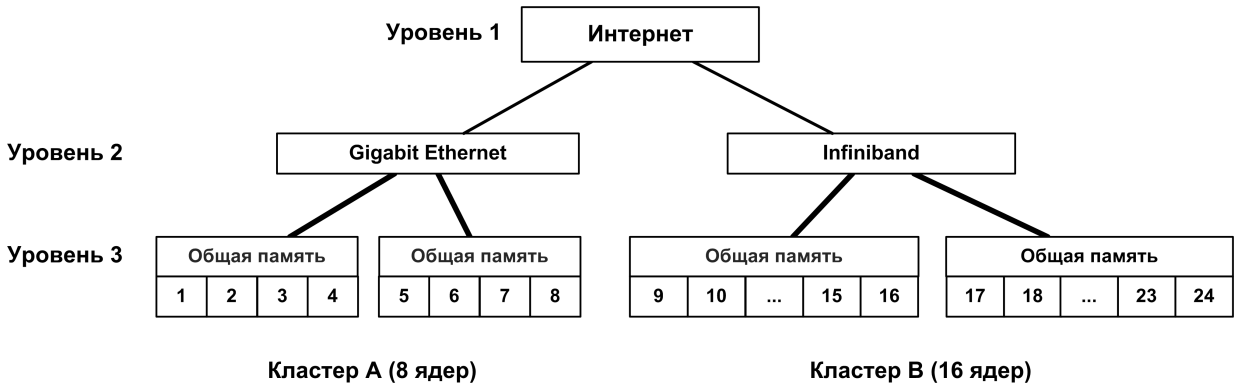



Рис. 3.2. Иерархическая организация коммуникационной среды кластерной ВС

Существующие библиотеки MPI (MPICH2, Open MPI, Intel MPI и др.) реализуют алгоритмы вложения параллельных программ с учетом только двух уровней коммуникационной сети – сети межузловых связей и общей памяти вычислительных узлов. Например, пакет hwloc [124, 125], который входит в библиотеки MPICH2 и Open MPI, использует информацию об иерархической структуре распределённых ВС и реализует вложение при помощи библиотеки Scotch [126] многоуровневого разбиения графов. Однако этот пакет ориентирован на сосредоточенные ВС и не учитывает структуры мультикластерных ВС. В алгоритме вложения, реализованном в пакете MPIPP [127], минимизируется суммарное время выполнения ветвей параллельной программы. Однако практика решения промышленных задач показывает, что время выполнения программы определяется максимальным из времён выполнения её ветвей.

Часть существующих работ направлено на вложение параллельных программ в фиксированные структуры коммуникационных сред ВС. Например, в работах [128–130] предложены алгоритмы вложения программ в системы, имеющие тороидальную структуру коммуникационной среды. В работах [131] и [132] оптимизируется вложение регулярных и нерегулярных, соответственно, информационных графов в структуры, представленные двумерными и трёхмерными решётками и торами. В статье [133] предлагается специализированный алгоритм *TreeMatch* вложения, ориентированный на ВС на базе технологии Non-Uniform Memory Access (NUMA). Усовершенствованный вариант этого алгорит-

ма [134] реализует подход к оптимизации вложения на основе изменения номеров ветвей в процессе выполнения MPI-программы. Алгоритм, учитывающий особенности реализации технологии InfiniBand, используемой для связи между узлами вычислительных кластеров, рассматривается в статье [135].

Некоторые из предложенных алгоритмов [136, 137] направлены на оптимизацию вложения MPI-программ определённого класса.

Отдельные работы направлены на оптимизацию вложения виртуальных MPI-топологий. В работах [138, 139] рассматривается задача вложения виртуальных MPI-топологий в SMP-кластеры. При этом минимизируются накладные расходы на межмашинные обмены информацией. Авторами работы [140] предлагаются подходы для повышения эффективности построения виртуальных топологий в стандарте MPI 2.2. В статье [141] предлагаются алгоритмы вложения виртуальных MPI-топологий на основе анализа информационного графа параллельной программы и структуры системы, однако в нём не учитываются особенности построения мультикластерных систем.

В работах [142, 143] исследуются универсальные эвристические алгоритмы вложения в гетерогенные системы, проводится сравнение производительности алгоритмов и их масштабируемости на большемасштабных системах. В качестве показателей эффективности используется средняя длина пути (в смысле теории графов), пройденная сообщением, и минимальное время передачи сообщения.

В большей части работ при оптимизации вложения учитываются только дифференцированные обмены. Авторами статьи [144] предлагается подход, позволяющий учитывать коллективные операции при вложении. Однако данный подход ограничен в применении, так как требует информации об алгоритмах реализации коллективных обменов. В статье [145] рассматривается подход к оптимизации вложения с целью оптимизации коллективных операций в тороидальной сети системы Blue Gene/P и Blue Gene/Q.

Немаловажным является то, как в целевой функции (критерий оптимизации) учитывается интенсивность взаимодействия параллельных ветвей. Время

выполнения программ с большим числом информационных обменов небольшого размера (это характерно для программ на языках семейства PGAS – Partition Global Address Space) существенно зависит от накладных расходов, связанных с латентностью при передаче информационных сообщений. В алгоритмах вложения таких программ целесообразно учитывать количество обменов, а не суммарный размер сообщений. Данное обстоятельство в недостаточной степени учитывается в существующих работах.

Применение известных методов вложения MPI-программ возможно и в пространственно-распределённых ВС, однако они могут не обеспечить предельной эффективности использования ресурсов ВС, так как не учитывают все иерархические уровни коммуникационной среды. В пространственно-распределённых ВС важно учитывать наличие медленных каналов связи между подсистемами. Производительность этого уровня, как правило, значительно уступает остальным и существенным образом влияет на время выполнения параллельной программы.

В диссертации предлагаются метод и эвристические алгоритмы вложения MPI-программ в пространственно-распределённые ВС. В предложенных алгоритмах учитываются все иерархические уровни коммуникационной сети ВС [146–154].

### 3.1.2. Задача оптимального вложения в иерархические пространственно-распределённые ВС

Пусть имеется пространственно-распределённая ВС, состоящая из $H$ подсистем. Коммуникационная среда системы имеет иерархическую организацию и может быть представлена в виде дерева (рис. 3.2).

Рассмотрим вложение параллельной программы в подсистему из $N$ элементарных машин (ЭМ) (рис. 3.3). Для параллельной программы системой управления ресурсами выделяется подсистема (на рис. 3.3 обозначена серым цветом), которая также также имеет иерархическую структуру. Введём обозна-

чения: $n_l$ – количество элементов на уровне $l \in \{1, 2, \ldots, L\}$; $n_{lk}$ – количество прямых дочерних узлов элемента $k \in \{1, 2, \ldots, n_l\}$, находящегося на уровне $l$; $c_{lk}$ – количество ЭМ, принадлежащих потомкам данного элемента.

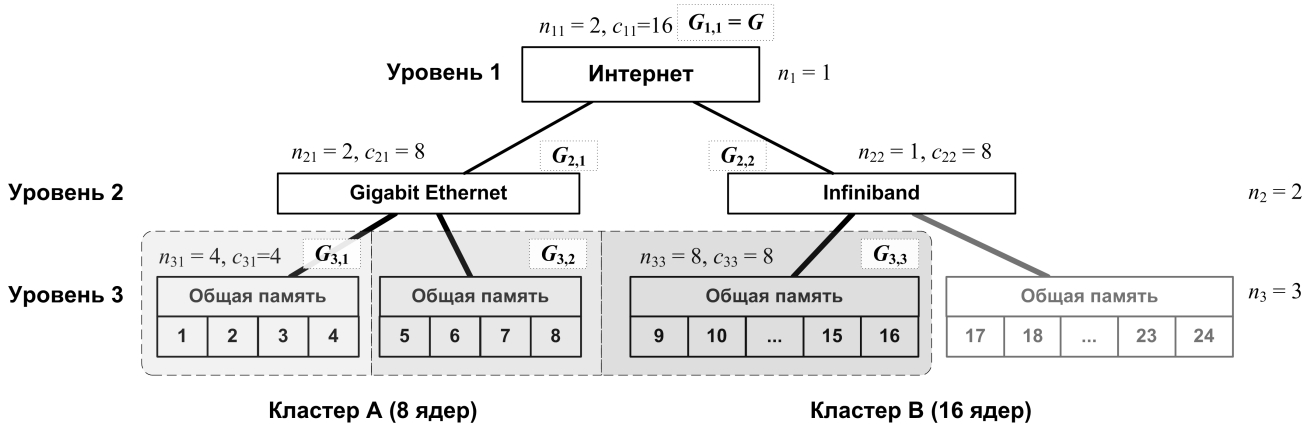



Рис. 3.3. Пример подсистемы ЭМ для решения параллельной задачи ранга $N = 16$

Параллельная программа, созданная в модели передачи сообщений, может быть представлена информационным графом $G = (V, E)$ (рис. 3.4), где $V = \{1, 2, \ldots, N\}$ – множество ветвей параллельной программы, а $E \subseteq V \times V$ – множество информационно-логических связей между ветвями. Обозначим через $d_{ij}$ вес ребра $(i, j) \in E$, отражающий интенсивность обменов данными между ветвями $i$ и $j$ при выполнении программы. В этой работе рассмотрено два способа задания весов рёбер: 1) $d_{ij}$ – суммарный объём данных, передаваемых между ветвями $i$ и $j$ за время выполнения программы ($[d_{ij}] =$ байт), 2) $d_{ij}$ – количество переданных сообщений между ветвями $i$ и $j$. Заметим, что вес $d_{ij}$ ребра может отражать как абсолютные, так и относительные объём или количество информационных обменов. Эти значения могут быть получены путём профилирования параллельной программы. Информационные графы параллельных программ часто имеют разреженную неоднородную структуру (рис. 3.5).

Вложение параллельной программы в ВС задаётся значениями переменных $x_{ij} \in \{0, 1\}$: $x_{ij} = 1$, если ветвь $i \in V$ назначена на процессорное ядро $j \in \{1, 2, \ldots, N\}$, в противном случае $x_{ij} = 0$.

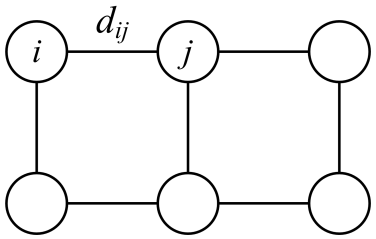
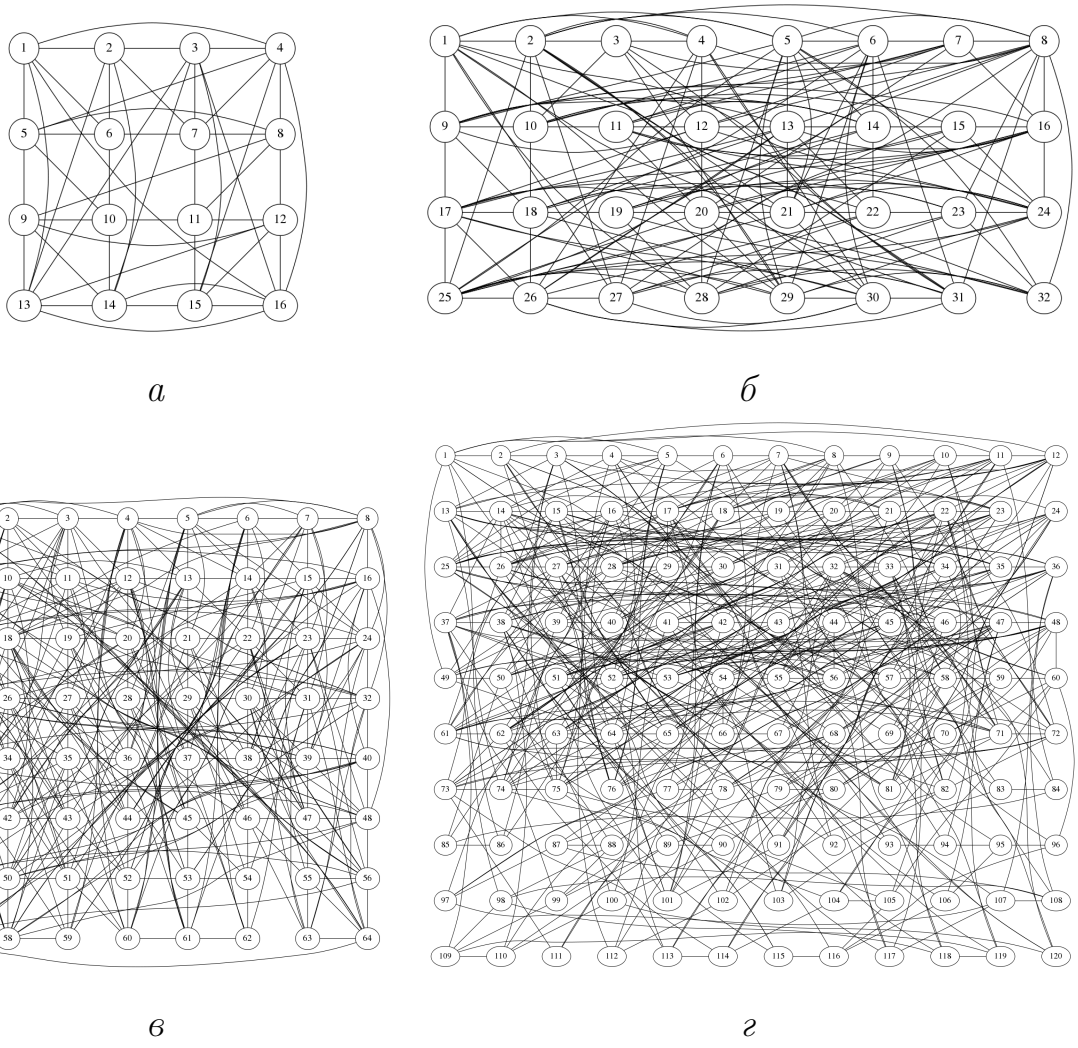



Рис. 3.4. Информационный граф $G = (V, E)$ параллельной программы



*а* *б*

*в* *г*

Рис. 3.5. Информационные графы программы Parallel Ocean Program (POP) для различного числа $N$ параллельных ветвей
*а* – $N = 16$, *б* – $N = 32$, *в* – $N = 64$, *г* – $N = 120$,

Для оценки эффективности вложения программы в структуру ВС можно использовать различные показатели: время выполнения программы, энергопотребление системы и др. В данной работе в качестве показателя эффективности вложения используется время $T$ выполнения информационных обме-

нов. Оно определяется максимальным из времён выполнения обменов ветвями программы.

Обозначим $t(i,j,p,q)$ – суммарное время взаимодействий между ветвями $i,j \in V$, назначенными на процессорные ядра $p$ и $q$ соответственно. Тогда

$$T(X) = \max_{i \in V} t_i = \max_{i \in V} \left\{ \sum_{j=1}^{N} \sum_{p=1}^{N} \sum_{q=1}^{N} x_{ip} x_{jq} t(i,j,p,q) \right\}.$$

Значение функции $t(i,j,p,q)$ может быть получено согласно различным аналитическим моделям оценки времени выполнения дифференцированных обменов (LogP, LogGP, Hockney). Например, в случае модифицированной модели Хокни (R. Hockney) [75, 155, 156] функция $t$ принимает вид $t(i,j,p,q) = d_{ij}/b(p,q)$, где $b(p,q)$ – пропускная способность канала связи между процессорными ядрами $p$ и $q$.

Сформулируем задачу оптимального вложения параллельной программы в ВС с иерархической организацией:

$$T(X) = \max_{i \in V} \left\{ \sum_{j=1}^{N} \sum_{p=1}^{N} \sum_{q=1}^{N} x_{ip} x_{jq} t(i,j,p,q) \right\} \to \min_{(x_{ij})} \tag{3.1}$$

при ограничениях

$$\sum_{j=1}^{N} x_{ij} = 1,\ i = 1,2,\ldots,N, \tag{3.2}$$

$$\sum_{i=1}^{N} x_{ij} = 1,\ j = 1,2,\ldots,N, \tag{3.3}$$

$$x_{ij} \in \{0,1\},\ i \in V, j \in \{1,2,\ldots,N\}. \tag{3.4}$$

Ограничения (3.2), (3.4) гарантируют назначение каждой ветви параллельной программы на единственную ЭМ. Ограничение (3.3) обеспечивает назначение на машину одной ветви.

Задача (3.1)-(3.4) относится к дискретной оптимизации и является трудноразрешимой [157, 158]. Рассмотрим приближённый метод её решения.

## 3.2. Алгоритмы вложения параллельных программ в иерархические пространственно-распределённые ВС

Метод вложения основан на разбиении графа задачи на подмножества интенсивно обменивающихся параллельных ветвей и вложения их в ЭМ, связанные быстрыми каналами связи. Задача разбиения графа рассмотрена ниже.

### 3.2.1. Задача оптимального разбиения графа на $k$ непересекающихся подмножеств

Пусть задан взвешенный граф $G' = (V', E')$; требуется найти разбиение вершин $V' = \{1, 2, \ldots, n\}$ графа на $k$ непересекающихся подмножеств $V'_1, V'_2, \ldots, V'_k$ с целью минимизации максимальной суммы рёбер, инцидентных любому подмножеству вершин. Число вершин в каждом подмножестве разбиения не должно превосходить заданного значения $s$. Обозначим $E'(i,j)$ – множество рёбер, соединяющих вершины из подмножеств $V'_i$ и $V'_j$, $i, j \in \{1, 2, \ldots, k\}$

$$E'(i,j) = \{(u,v) \in E' : u \in V'_i, v \in V'_j, i \neq j\};$$

$c(u,v,i,j)$ – вес ребра, инцидентного вершинам разных подмножеств $u \in V'_i$ и $v \in V'_j$,

$$c(u,v,i,j) = w(u,v)W(i,j),$$

где $w(u,v)$ – вес ребра $(u,v) \in E'$, $W(i,j)$ – коэффициент рёбер, соединяющих вершины из подмножеств $i$ и $j$.

С учётом ограничений, накладываемых на подмножества $V'_i$, сформулируем задачу оптимального разбиения графа на $k$ непересекающихся подмножеств:

$$F(V'_1, V'_2, \ldots, V'_k) = \max_{i=1,\ldots,k} \left\{ \sum_{j=1}^{k} \sum_{(u,v)\in E'(i,j)} c(u,v,i,j) \right\} \to \min_{(V'_1,V'_2,\ldots,V'_k)}, \qquad (3.5)$$

при ограничениях:

$$V'_1 \cap V'_2 \cap \cdots \cap V'_k = \varnothing, \qquad (3.6)$$

$$V'_1 \cup V'_2 \cup \cdots \cup V'_k = V', \qquad (3.7)$$

$$|V'_i| > 0, i = 1, 2, \ldots, k, \qquad (3.8)$$

$$|V'_i| \leq s, i = 1, 2, \ldots, k. \qquad (3.9)$$

На рис. 3.7 представлен пример разбиения взвешенного графа, изображённого на рис. 3.6, на четыре подмножества.

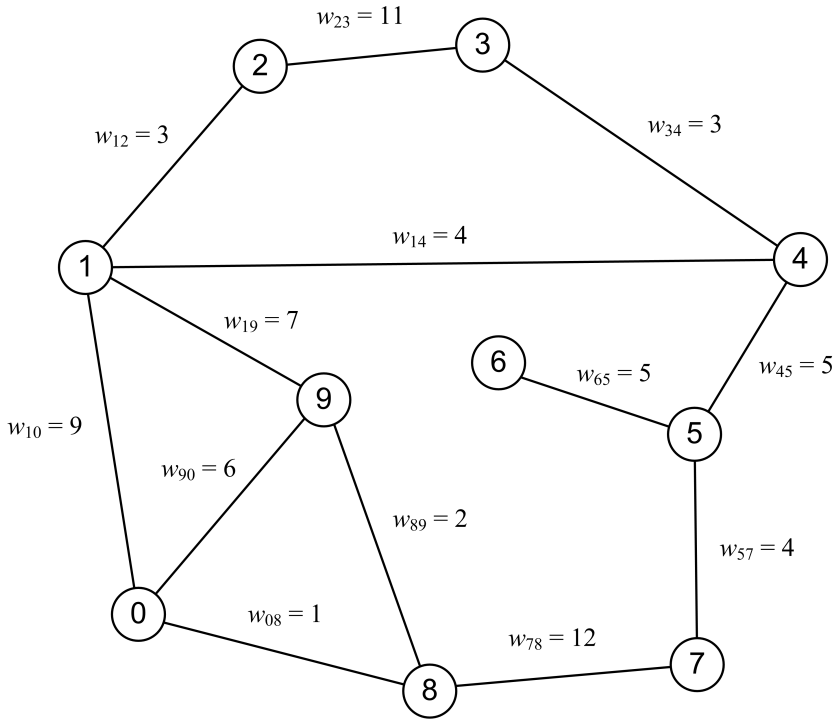



Рис. 3.6. Пример взвешенного графа

Сумма весов рёбер, инцидентных подмножеству $V'_1$, равна $w(1,2)W(1,2) + w(1,4)W(1,3) + w(8,9)W(1,4) + w(8,0)W(1,4) = 23$; для второго подмножества $V'_2$ – $w(1,2)W(1,2) + w(3,4)W(2,3) = 21$, для третьего подмножества $V'_2$ –

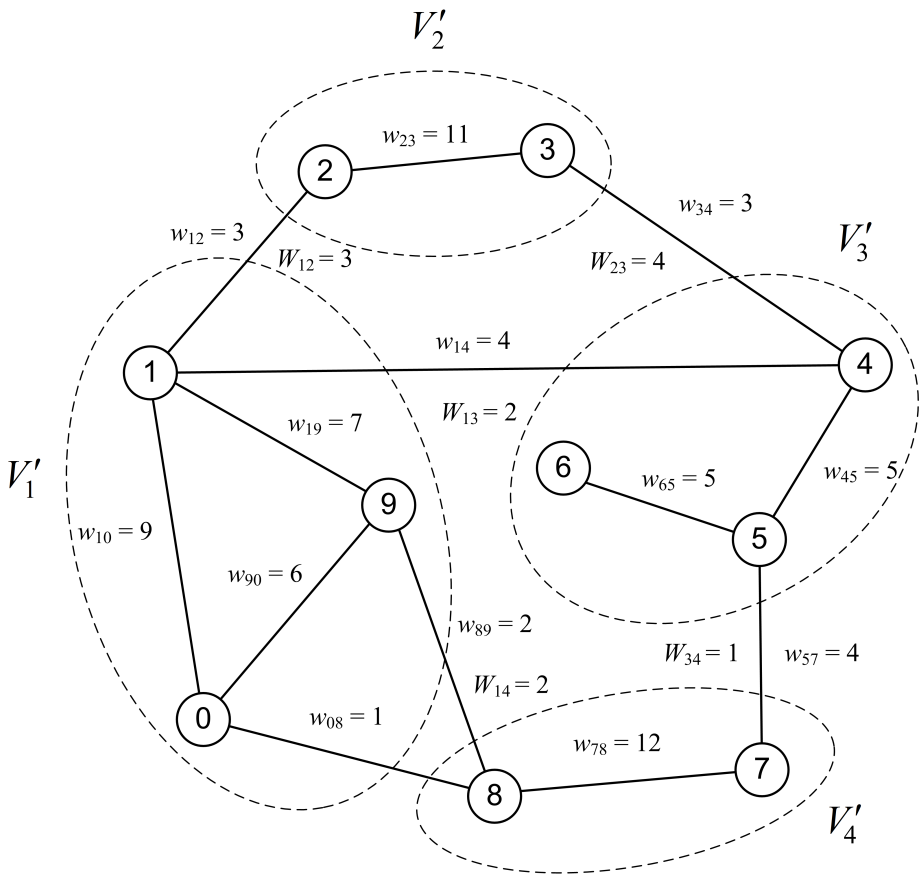



Рис. 3.7. Разбиение взвешенного графа на четыре подмножества

$$k = 4;\ s = 3;\ F(V_1', V_2', V_3', V_4') = 24$$

$w(3,4)W(2,3) + w(1,4)W(1,3) + w(5,7)W(3,4) = 24$, для четвёртого подмножества $V_2'$ – $w(0,8)W(1,4) + w(8,9)W(1,4) + w(5,7)W(3,4) = 10$. Значение целевой функции $F(V_1', V_2', V_3', V_4') = 24$.

Известно [158], что задача (3.5)–(3.9) разбиения взвешенного графа на $k$ непересекающихся подмножеств является трудноразрешимой.

### 3.2.2. Метод вложения параллельных программ в мультикластерные ВС

Предложен метод *HierarchicMap* вложения параллельных программ в мультикластерные ВС с иерархической структурой. Цель разбиения – минимизация суммы весов рёбер, инцидентных разным подмножествам разбиения. Разбиение выполняется многократно: для каждого уровня иерархии коммуникационной среды (рис. 3.9). В листинге 1 приведён псевдокод метода.

Листинг 3.1. Псевдокод метода *HierarchicMap*

*HierarchicMap*$(G_{lk}, l, k)$

Входные данные: $G$ – информационный граф параллельной программы,
$l$ – уровень коммуникационной среды,
$k$ – номер текущего элемента.

Выходные данные: $x_{ij}$ – вложение; $x_{ij} = 1$, если ветвь $i$ назначена на ЭМ $j$, иначе $x_{ij} = 0$.

```
1 if l = L then
2     return G_{L,1}, G_{L,2}, ..., G_{L,n_L}
3 else
4     (G_{l+1,1}, G_{l+1,2}, ... G_{l+1,n_lk}) ← PartGraph(G_lk; n_lk; c_{l+1,1}, c_{l+1,2}, ..., c_{l+1,n_lk})
5     for k = 1 to n_lk do
6         HierarchicMap(G_{l+1,k}, l + 1, k)
7     end for
8 end if
```

Суть метода рассмотрим на примере вложения параллельной программы в подсистему из 16 ЭМ (рис. 3.3). На первом шаге выполняется разбиение (*PartGraph*) исходного графа $G$ на $n_{11}$ подграфов ($G_{21}$ и $G_{22}$) по $c_{21}$ и $c_{22}$ вершин. Далее графы $G_{21}$ и $G_{22}$ рекурсивно разбиваются на $n_{21}$ и $n_{22}$ частей по $c_{21}$ и $c_{22}$ вершин соответственно. Полученные в результате подграфы $G_{31}$, $G_{32}$, $G_{33}$ (их вершины – ветви программы) назначаются на узел 1 (процессорные ядра $1, \ldots, 4$) и узел 2 (процессорные ядра $5, \ldots, 8$) кластера A и узел 1 (процессорные ядра $9, \ldots, 16$) кластера B.

Функция *PartGraph* возвращает список подграфов, получаемых в результате разбиения исходного графа. Задача оптимального разбиения взвешенного графа на $k$ непересекающихся подмножеств является трудноразрешимой. Для её решения существуют точные и приближённые алгоритмы различной вычислительной сложности. Интерес представляют многоуровневые

алгоритмы разбиения графов [159, 160], позволяющие получать субоптимальные решения этой задачи и характеризующиеся невысокой вычислительной сложностью. Данные алгоритмы включают этапы сжатия графа, начального разбиения и улучшения разбиения. В основе большинства алгоритмов улучшения разбиений лежит модифицированная нетрудоёмкая эвристика Кернигана-Лина (Kernighan-Lin) [161].

Вычислительная сложность метода *HierarchicMap* определяется количеством выполнений процедуры *PartGraph* разбиения графа. В случае, если используется многоуровневый алгоритм разбиения графа, её трудоёмкость составляет $T_{PG} = O(|E| \log_2 z)$, где $|E|$ – количество рёбер в графе, $z$ – число подмножеств разбиения графа. Разбиение выполняется в пределах каждого уровня $l = 1, \ldots, L-1$ для всех его элементов $k = 1, \ldots, n_l$ (графы $G_{lk}$). Время работы метода приведено на рис. 3.8.

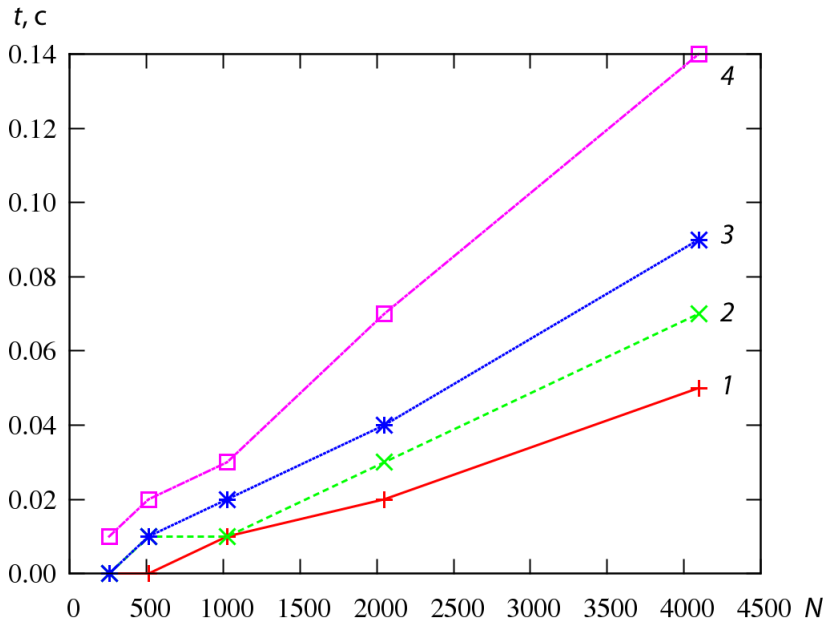



Рис. 3.8. Зависимость времени работы метода *HierarchicMap* (алгоритм *L1Map*) от количества $N$ ветвей в параллельной программе для различного числа $k$ подмножеств разбиения (процессор Intel Xeon E5420):
*1* – $k = 16$, *2* – $k = 32$, *3* – $k = 64$, *4* – $k = 128$

На основе метода *HierarchicMap* предложены алгоритмы, различающиеся между собой уровнями $l$ коммуникационной среды, которые учитываются при формировании разбиения. Назовём *L1Map* алгоритм, учитывающий только уровень $l = 1$ связи между подсистемами (кластерами) и не учитывающий уровень связи между узлами, *L2Map* – алгоритм, учитывающий только уровень $l = 2$ связи между узлами и не учитывающий уровень связи между подсистемами, *L12Map* – алгоритм, учитывающий как уровень связи между узлами, так и уровень связи между подсистемами, и т.д.

В листинге 3.2 в качестве примера приведён псевдокод алгоритма *L12Map*. Начальные условия алгоритма: $l = 1$, $k = 1$.

Листинг 3.2. Псевдокод алгоритма *L12Map*

*L12Map*$(G_{lk}, l, k)$

| | |
|---|---|
| Входные данные: | $G$ – информационный граф параллельной программы, |
| | $l$ – уровень коммуникационной среды, |
| | $k$ – номер текущего элемента, |
| Выходные данные: | $x_{ij}$ – вложение; $x_{ij} = 1$, если ветвь $i$ назначена на ЭМ $j$, иначе $x_{ij} = 0$. |

```
1  if l = 3 then
2      return G_{3,1}, G_{3,2}, ..., G_{3,n_3}
3  else
4      (G_{l+1,1}, G_{l+1,2}, ... G_{l+1,n_lk}) ← PartGraph(G_lk; n_lk; c_{l+1,1}, c_{l+1,2}, ..., c_{l+1,n_lk})
5      for k = 1 to n_lk do
6          L12Map(G_{l+1,k}, l + 1, k)
7      end for
8  end if
```

Рассмотрим пример вложения параллельной программы Parallel Ocean Problem (POP) в подсистему из $N = 32$ ЭМ алгоритмом *L12Map* (рис. 3.9). В алгоритме при разбиении учитывается два уровня коммуникационной среды

BC. Поэтому сначала выполняется разбиение (процедура *PartGraph*) для уровня $l = 1$ коммуникационной среды (сеть между подсистемами). Выполняется разбиение для потомков 1-го элемента на 1-м уровне: $n_{11} = 3$ (количество подсистем). Результатом разбиения являются графы $G_{21}$, $G_{22}$ и $G_{23}$ по 8, 16 и 8 вершин, соответственно. Количество вершин в подграфах соответствует количеству процессорных ядер в кластере A, кластере B и кластере C.

На следующем шаге выполняется разбиение для следующего уровня $l = 2$ иерархии коммуникационной среды BC (сеть связи между вычислительными узлами кластеров). На этом шаге граф $G_{21}$ разбивается на $n_{21} = 4$ (число прямых потомков элемента 1 на уровне 2) подграфов $G_{31}$, $G_{32}$, $G_{33}$, $G_{34}$ по 2 вершины в каждом (количество процессорных ядер в вычислительном узле кластера A). Результатом разбиения графа $G_{22}$ являются $n_{22} = 2$ (количество прямых потомков элемента 2 на уровне 2) подграфов $G_{35}$ и $G_{36}$ по 8 вершин (число ядер в узле кластера B). Граф $G_{23}$ разбивается на $n_{23} = 2$ (число прямых потомков элемента 3 на уровне 2) подграфа $G_{37}$ и $G_{38}$ по 4 вершины (количество ядер в узле кластера C). Полученные в результате подмножества первоначального графа вкладываются в вычислительные узлы соответствующих подсистем. Порядок следования параллельных ветвей в пределах одного узла не учитывается. Таким образом, ветви 1, 2 назначаются на процессорные ядра $1, 2$; ветви $9, 10$ – на ядра $3, 4$; ветви $17, 18$ – на ядра $5, 6$; ветви $25, 26$ – на ядра $7, 8$; ветви $3, 4, 11, 12, 19, 20, 27, 28$ – на ядра $9, \ldots, 16$; ветви $5, 6, 13, 14, 21, 22, 29, 30$ – на ядра $17, \ldots, 24$; ветви $23, 24, 31, 32$ – на ядра $25, \ldots, 28$; ветви $7, 8, 15, 16$ – на ядра $29, \ldots, 32$.

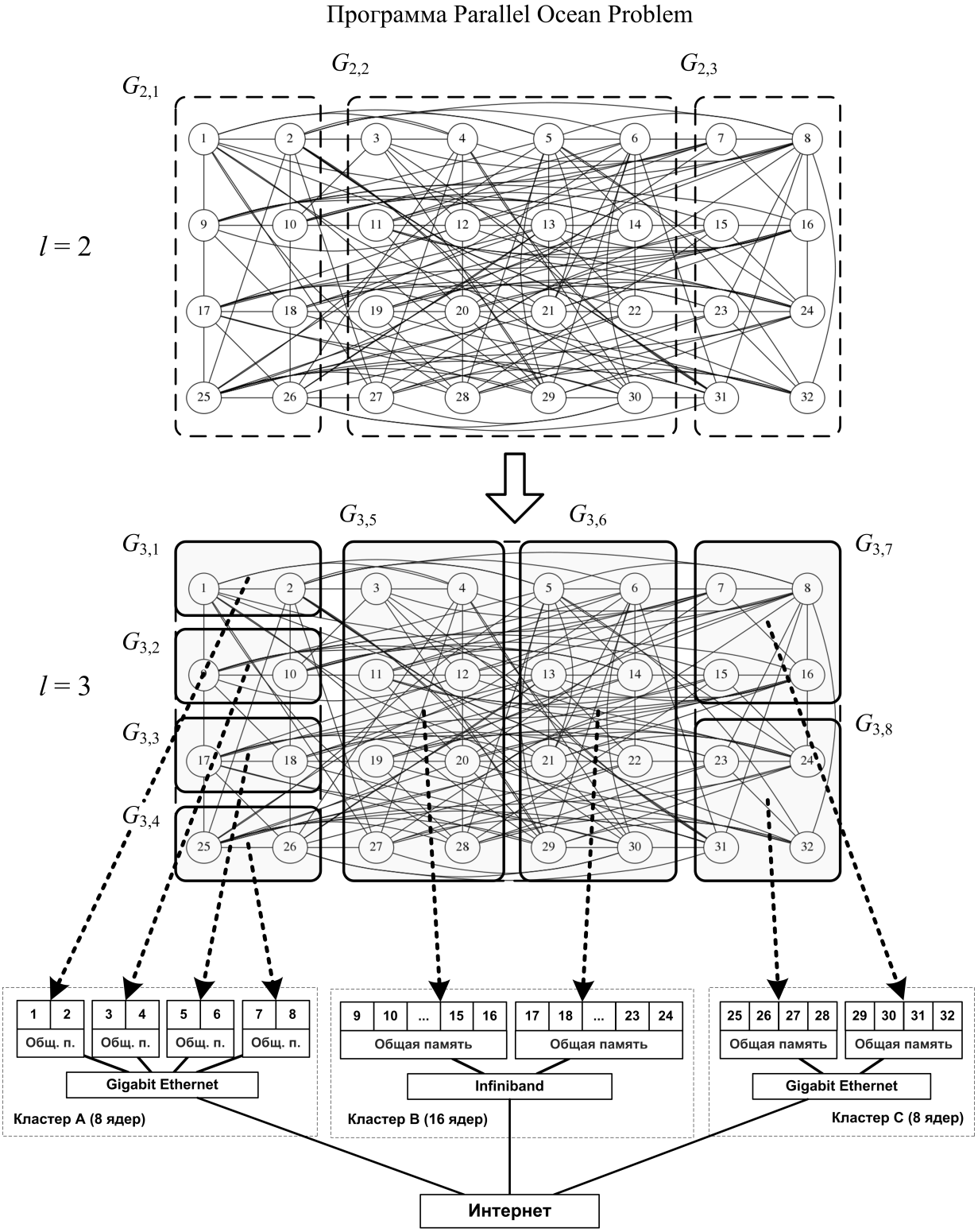



Рис. 3.9. Вложение программы Parallel Ocean Problem (POP) в подсистему из $N = 32$ ЭМ методом *HierarchicMap* (алгоритм *L12Map*)

## 3.3. Моделирование алгоритмов вложения MPI-программ в подсистемы пространственно-распределённых ВС

### 3.3.1. Организация экспериментов

Моделирование алгоритмов вложения проводилось на мультикластерной ВС Центра параллельных вычислительных технологий ФГОБУ ВПО "Сибирский государственный университет телекоммуникаций и информатики" (ЦПВТ ФГОБУ ВПО "СибГУТИ") и Лаборатории вычислительных систем Института физики полупроводников им. А.В. Ржанова СО РАН (ИФП СО РАН).

Для запуска MPI-программ на ресурсах пространственно-распределённых подсистем реализован подход на основе межсетевого протокола IPv6 (рис. 3.10). В данном протоколе используется 128-битовая адресация, которая позволят выделять всем вычислительным узлам пространственно-распределённой ВС глобальные IP-адреса. Наличие уникальных адресов обеспечивает возможность установления сетевого IP-соединения между любой парой узлов системы для запуска на них ветвей MPI-программ [162, 163]. Получение адресов и взаимодействие между узлами подсистем в ходе моделирования осуществлялось с помощью протокола 6to4 [164], который реализует передачу пакетов протокола IPv6 поверх сети IPv4. При этом сначала головным узлам, имеющим глобальный IPv4-адрес, по протоколу 6to4 выделялись IPv6-адреса, после чего средствами службы radvd IPv6-адреса от головных узлов всех подсистем раздавались вычислительным узлам.

Для запуска MPI-программ пользователь должен зайти на одну из подсистем и создать файл со списком узлов мультикластерной ВС для выполнения параллельной программы. После этого он запускает MPI-программу утилитой mpiexec (mpirun). Описанная схема запуска MPI-программ требует от библиотеки MPI поддержки протокола IPv6. В данной работе применялась библиотека Open MPI 1.4.5 [165].

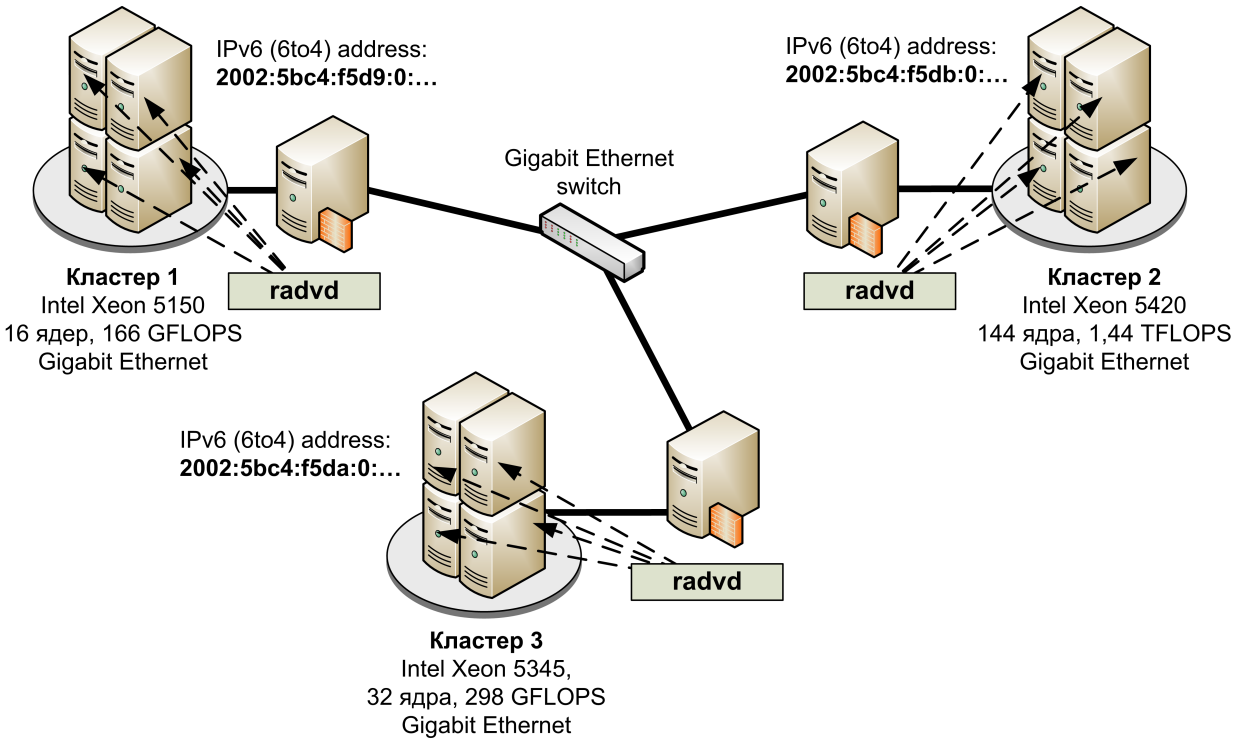



Рис. 3.10. Конфигурация тестовой подсистемы мультикластерной ВС

Натурные эксперименты по вложению тестовых MPI-программ проводились на действующей мультикластерной системе ЦПВТ ФГОБУ ВПО “СибГУТИ” и Лаборатории ВС ИФП СО РАН. Тестовая подсистема (рис. 3.10) включала в себя три вычислительных кластера:

– сегмент D: 4 узла (2 x Intel Xeon 5150, 16 процессорных ядер),

– сегмент E: 4 узла (2 x Intel Xeon 5345, 32 процессорных ядра),

– сегмент F: 18 узлов (2 x Intel Xeon 5420, 144 процессорных ядер).

Сеть связи между узлами – Gigabit Ethernet, сеть связи между сегментами ВС – Gigabit Ethernet.

На подсистемах установлена операционная система GNU/Linux. Использовались библиотека Open MPI 1.4.5 с поддержкой IPv6 и библиотека VampirTrace [166] для профилирования MPI-программ.

Опираясь на известные распространённые схемы межмашинных обменов [167], были выбраны следующие тестовые MPI-программы:

– The Parallel Ocean Program – пакет моделирования климатических процессов в мировом океане;

– SWEEP3d – программа для моделирования процессов распространения нейтронов;

– GRAPH500 – тест производительности ВС на основе обработки неструктурированных данных графового типа;

– программы LU, SP, MG, BT из пакета тестов производительности NAS Parallel Benchmarks.

При формировании вложений использовались библиотеки разбиения графов Scotch [126], METIS [168] и gpart. Все пакеты реализуют многоуровневые алгоритмы разбиения графов с использованием прямого метода разбиения на $k$ непересекающихся подмножеств. Стоит заметить, что библиотека Scotch используется в пакете hwloc [124, 125] при вложении параллельных MPI-программ в пакетах MPICH2 и Open MPI.

Ранг $r$ параллельной программы выбирался из множества $\{120, 64, 36, 32\}$ в зависимости от задачи.

Для каждой программы генерировалось два информационных графа в зависимости от способа формирования весов $d_{ij}$ рёбер. В первом случае вес ребра отражал объём данных, передаваемых между ветвями $i$ и $j$ (datasize). Во втором графе вес ребра отражал количество информационных сообщений, передаваемых между ветвями $i$ и $j$ (nops).

### 3.3.2. Результаты экспериментов

На рис. 3.11, 3.12, 3.13 представлены некоторые результаты исследования алгоритмов. В процессе моделирования измерялось время выполнения параллельных программ при вложении их в мультикластерную ВС алгоритмами *L2Map*, *L1Map* и *L12Map*. Для разбиения графов задач применялся пакет gpart, реализующий многоуровневое разбиение графов [159, 160] на основе рекурсивной бисекции с использованием модифицированной эвристики Кернигана-Лина (Kernighan-Lin) [161]. Выполнялось сравнение полученных вложений с линейным вложением, при котором ветви последовательно вкладываются в ЭМ вы-

деленной подсистемы (такое вложение реализуется по умолчанию библиотеками MPI).

Время выполнения MPI-программ при вложении их алгоритмом *L12Map* ниже по сравнению с остальными алгоритмами. Это объясняется тем, что данный алгоритм позволяет учитывать как внутрикластерную сеть связи между узлами, так и сеть связи между кластерами. Каналы связи между подсистемами и внутри подсистем существенно различаются по производительности, поэтому при вложении необходимо учитывать оба коммуникационных уровня. Различие результатов вложения алгоритмами *L1Map* и *L2Map* обусловлено схемами межмашинных обменов в конкретных параллельных программах, структурой информационных графов, рангом параллельной программы и структурой мультикластерной ВС.

Уменьшение времени выполнения параллельных программ POP (в 2,5 раз, $r = 120$), Sweep3D (на 30%, $r = 120$), GRAPH500 (до 10 раз, $r = 120$), NPB MG (в 5 раз, $r = 64$), NPB SP (на 30%, $r = 64$), NPB LU (на 10%, $r = 120$) с использованием оптимизированного вложения по сравнению с линейным вложением обусловлено разреженностью и неоднородностью графов и преобладанием в программе дифференцированных MPI-обменов. В таких графах можно выделить подмножества параллельных ветвей, интенсивно обменивающиеся данными, и распределить их по ЭМ, которые соединены быстрыми каналами связи. Эффект от использования созданных алгоритмов при вложении параллельных программ с однородными графами (NPB LU, NPB BT) и с графами недетерминированных обменов (в которых структура информационного графа для каждого запуска различна) незначителен.

Время работы алгоритмов вложения не превышает 1 с (рис. 3.8) даже для большого числа $N$ параллельных ветвей, что позволяет их использовать в большемасштабных системах.

Оптимальный выбор способа формирования информационного графа (datasize или nops) зависит от типа параллельной программы и от её ранга.

Если известно, что в графе параллельной программы преобладают частые обмены сообщениями небольшого размера (характерно для языков семейства PGAS), рекомендуется использовать тип графа nops. В остальных случаях рекомендуется использовать datasize.

Проведено моделирование с целью сравнения эффективности пакетов METIS [168], Scotch [126] и gpart разбиения графов (рис. 3.14, 3.15, 3.16). В качестве алгоритма вложения применялся *L2Map*. Использование библиотеки Scotch обеспечивает незначительное сокращение времени выполнения программы POP по сравнению с другими пакетами. На остальных тестах все библиотеки дают сопоставимые или противоречивые результаты, поэтому могут быть рекомендованы к использованию все указанные пакеты.

Созданные алгоритмы вложения эффективны для параллельных программ с разреженными и неоднородными графами. К таким программам относятся POP, NPB MG, GRAPH500. На рис. 3.17 приведено время выполнения операций диффенцированного и коллективного обменов для программ POP и NPB LU. Видно, что для программы POP (3.17а,б) оптимизация достигается за счёт уменьшения времени дифферецированных обменов. Данный тип обменов учитывается при построении информационных графов параллельных программ. Алгоритмы реализации коллективных обменов различаются в зависимости от библиотек MPI, что затрудняет их учёт в информационных графах. Несмотря на то, что в программе NPB LU преобладают дифференцированные обмены, алгоритмы вложения неэффективны из-за высокой однородности информационных графов.

## 3.4. Выводы

1. Разработан метод *HierarchicMap* и алгоритмы иерархического вложения в мультикластерные ВС параллельных MPI-программ с целью минимизации времени их выполнения. Суть метода заключается в последова-

тельном разбиении информационного графа параллельной программы в соответствии со всеми иерархическими уровнями системы, что позволяет полностью учитывать структуру коммуникационной среды пространственно-распределённых ВС. Алгоритмы могут применяться не только при вложении MPI-программ, но также в run-time системах языков семейства PGAS.

2. Созданные алгоритмы позволяют сократить время выполнения параллельных программ в среднем на 30%. Результаты моделирования вложения реальных MPI-программ показали, что на всех тестовых задачах алгоритм *L12Map*, реализующий иерархическое разбиение информационного графа в соответствии с коммуникационными уровнями системы, обеспечивает наименьшее время выполнения программ. Данный алгоритм учитывает архитектурные особенности мультикластерных и GRID-систем, в которых вычислительные узлы разных подсистем взаимодействуют через медленные каналы связи.
3. Эффективность вложения зависит от объёма и интенсивности передаваемой информации в дифференцированных обменах. Алгоритмы позволяют существенно (в 1-5 раз) сократить время выполнения параллельных программ с информационными графами, имеющими разреженную структуру с преобладанием дифференцированных обменов (например, POP, NPB MG, GRAPH500).
4. Все предложенные алгоритмы характеризуются невысокой вычислительной трудоёмкостью и позволяют вкладывать параллельные программы в большемасштабные ВС. Для разбиения информационных графов при формировании вложений могут быть рекомендованы многоуровенвые алгоритмы разбиения графов, реализованные в пакетах METIS, Scotch, gpart. С целью уменьшения времени вложения все алгоритмы могут быть эффективно распараллелены.

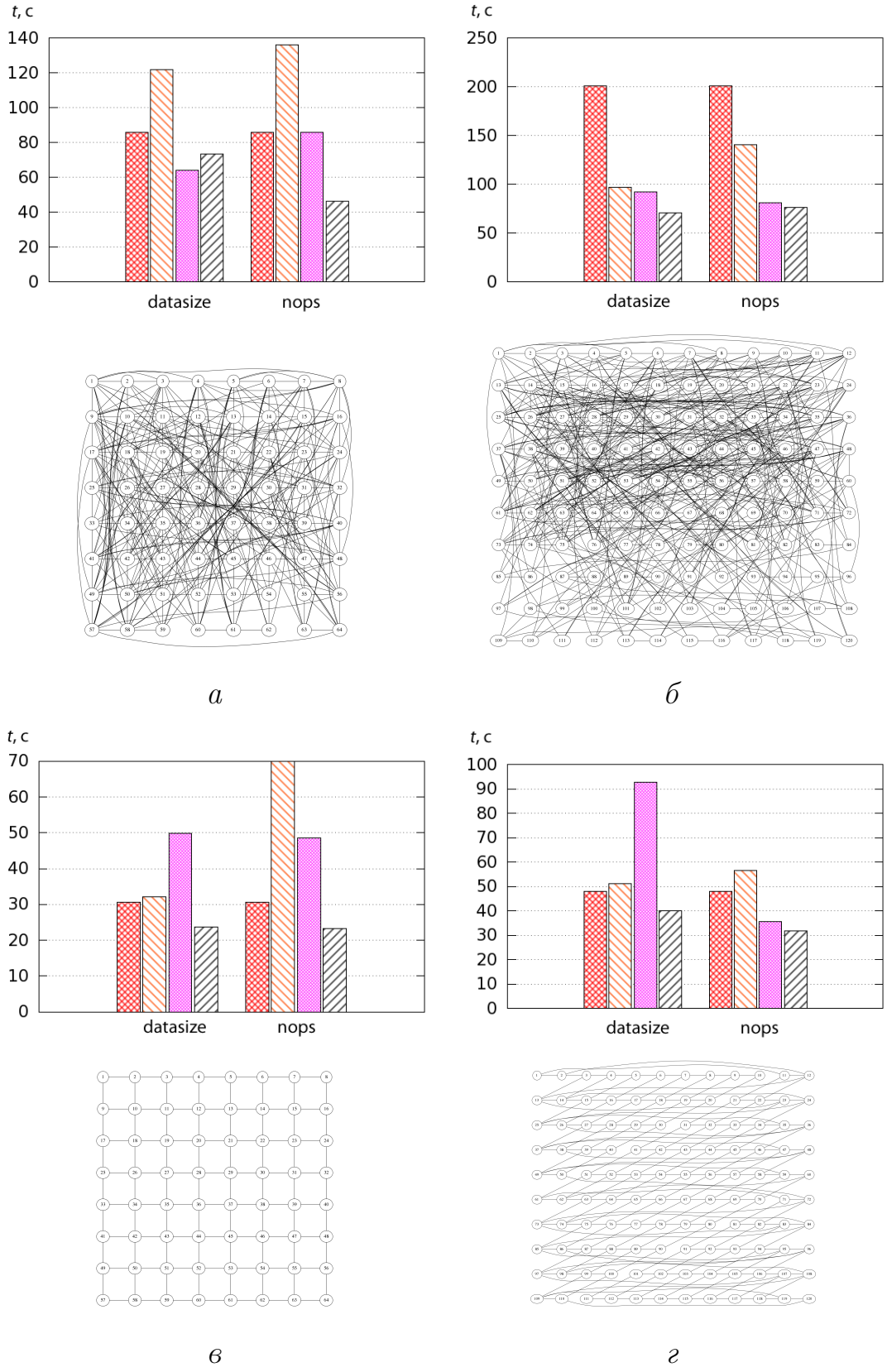
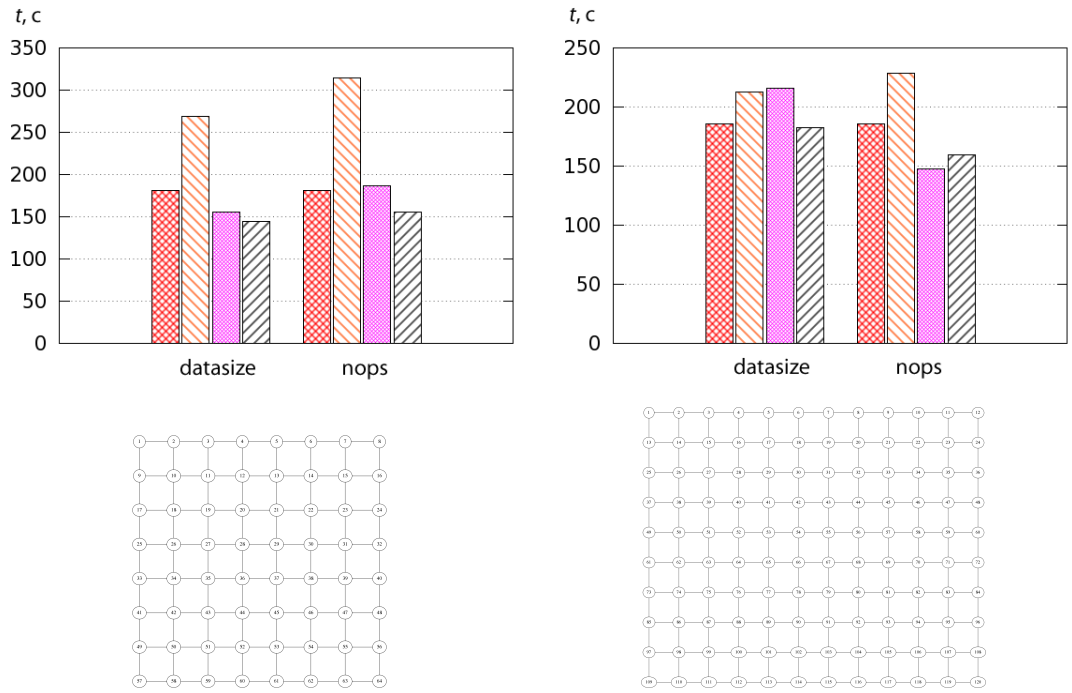



Рис. 3.11. Сравнение алгоритмов вложения. Программы POP и Sweep3D

*а* – POP, $r = 64$, *б* – POP, $r = 120$

*а* – Sweep3D, $r = 64$, *б* – Sweep3D, $r = 120$

– линейное вложение, – *L2Map*, – *L1Map*, – *L12Map*



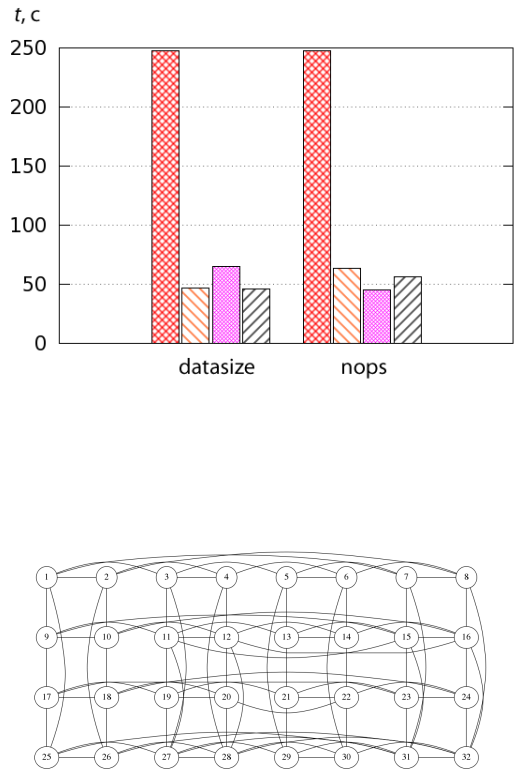



*а*



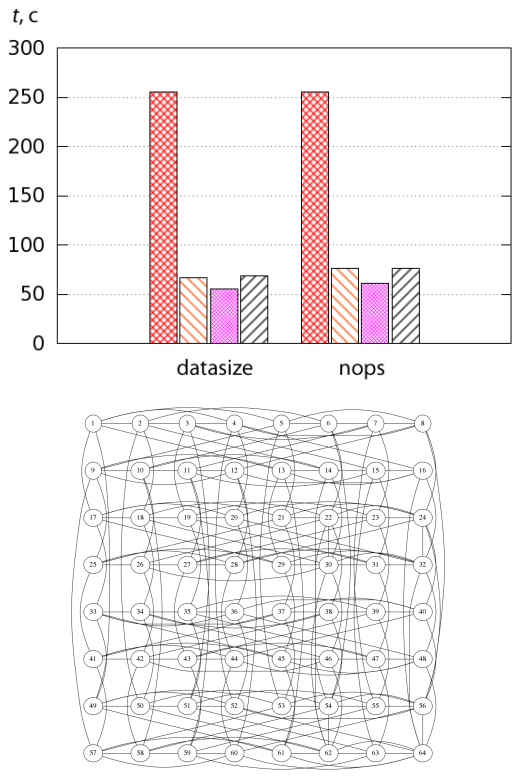



*б*






*в*





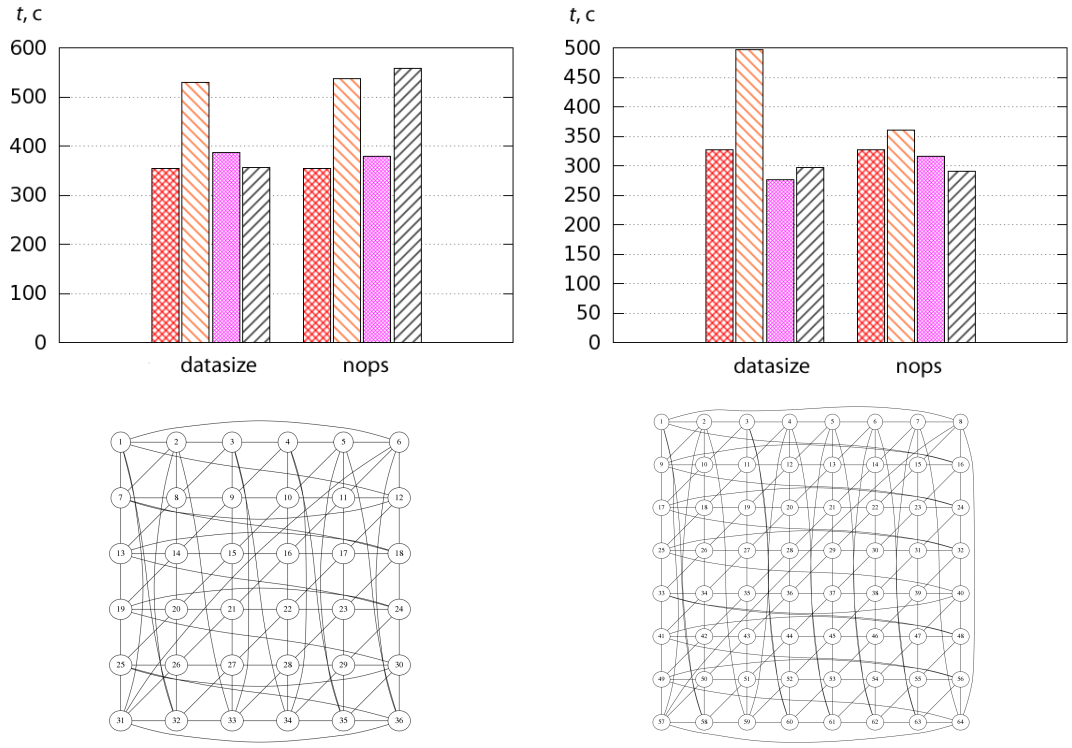

*г*

Рис. 3.12. Сравнение алгоритмов вложения. Программы NPB LU и NPB MG

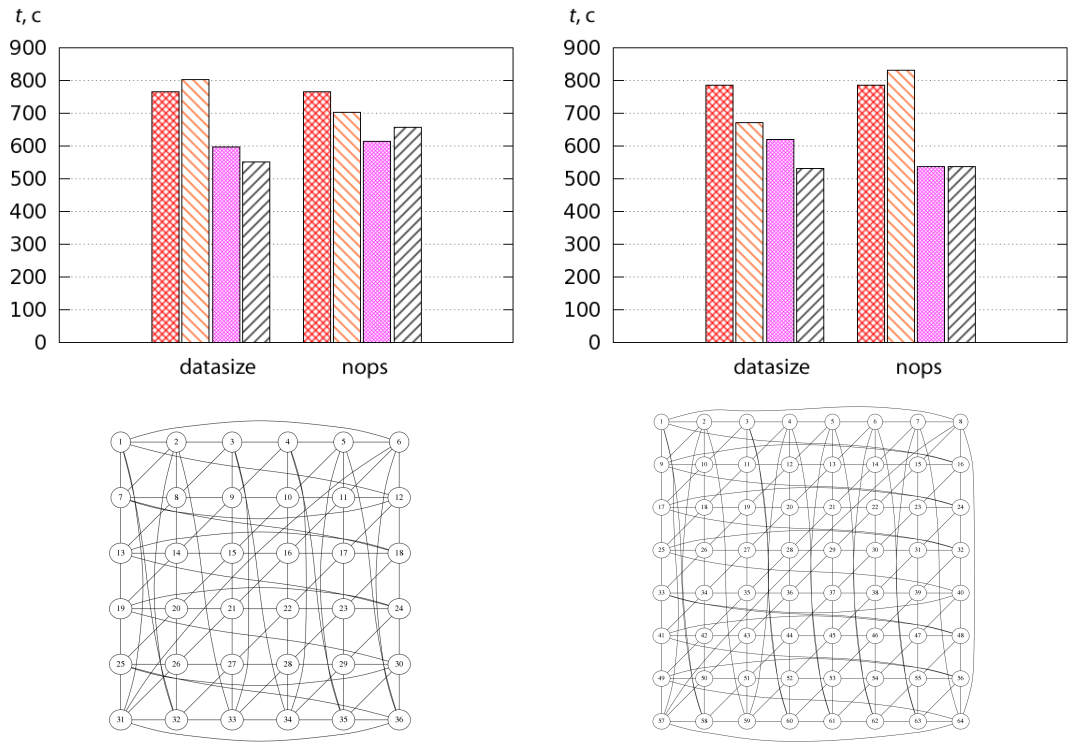
*а* – NPB LU, $r = 64$, *б* – NPB LU, $r = 120$

*а* – NPB MG, $r = 32$, *б* – NPB MG, $r = 64$

– линейное вложение, – *L2Map*, – *L1Map*, – *L12Map*




*а* *б*



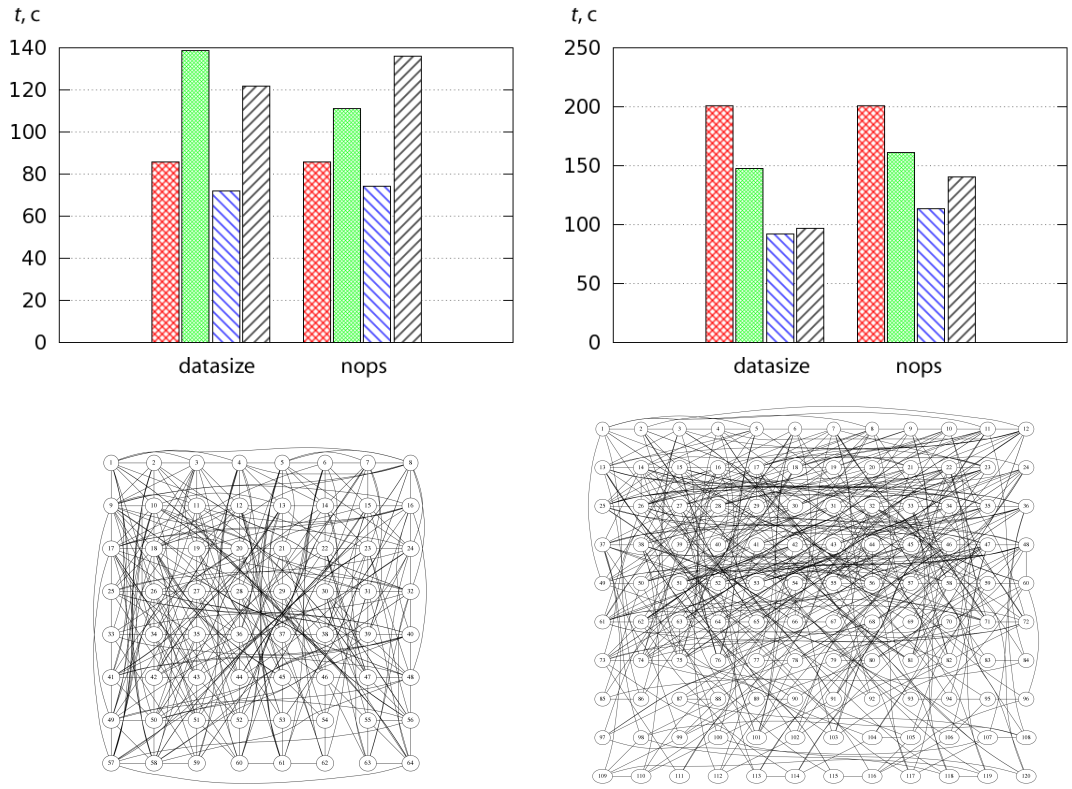

*а* *б*

Рис. 3.13. Сравнение алгоритмов вложения. Программа NPB BT и NPB SP

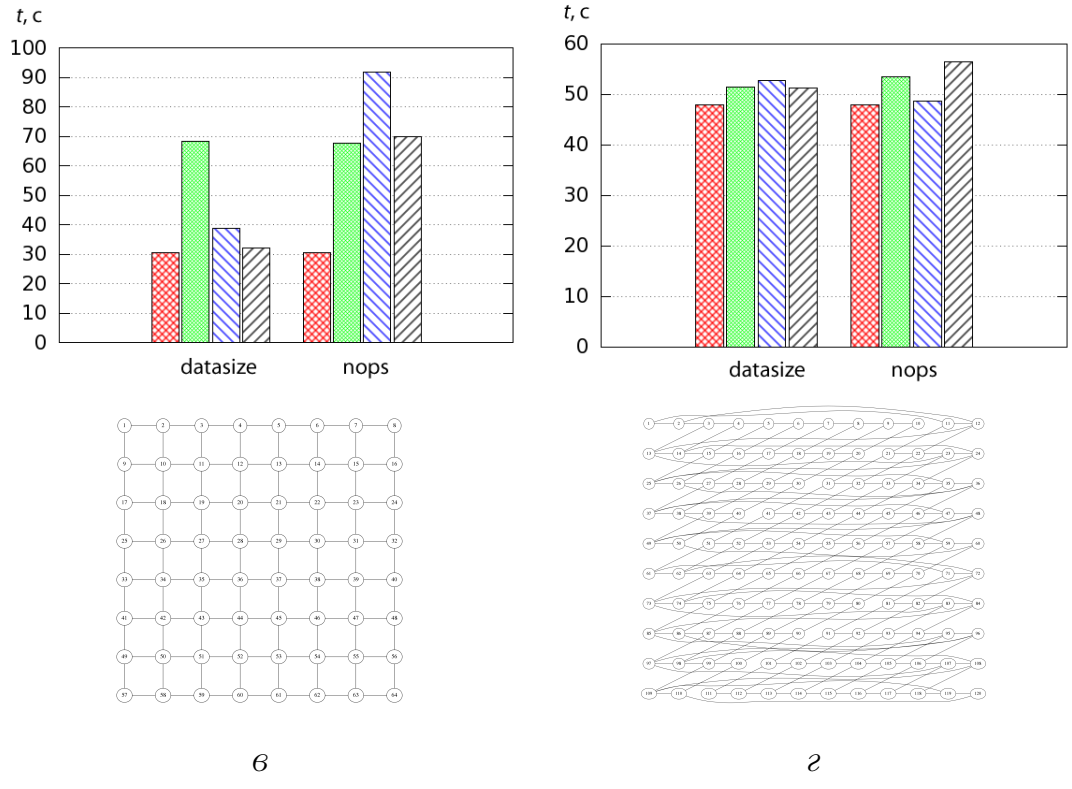
*а* – NPB BT, $r = 36$, *б* – NPB BT, $r = 64$

*а* – NPB SP, $r = 36$, *б* – NPB SP, $r = 64$

– линейное вложение, – *L2Map*, – *L1Map*, – *L12Map*




*а* *б*




*в* *г*

Рис. 3.14. Сравнение библиотек разбиения графов. Программы POP и Sweep3D

*а* – POP, $r = 64$, *б* – POP, $r = 120$

*а* – Sweep3D, $r = 64$, *б* – Sweep3D, $r = 120$

– линейное вложение, – METIS, – Scotch, – gpart

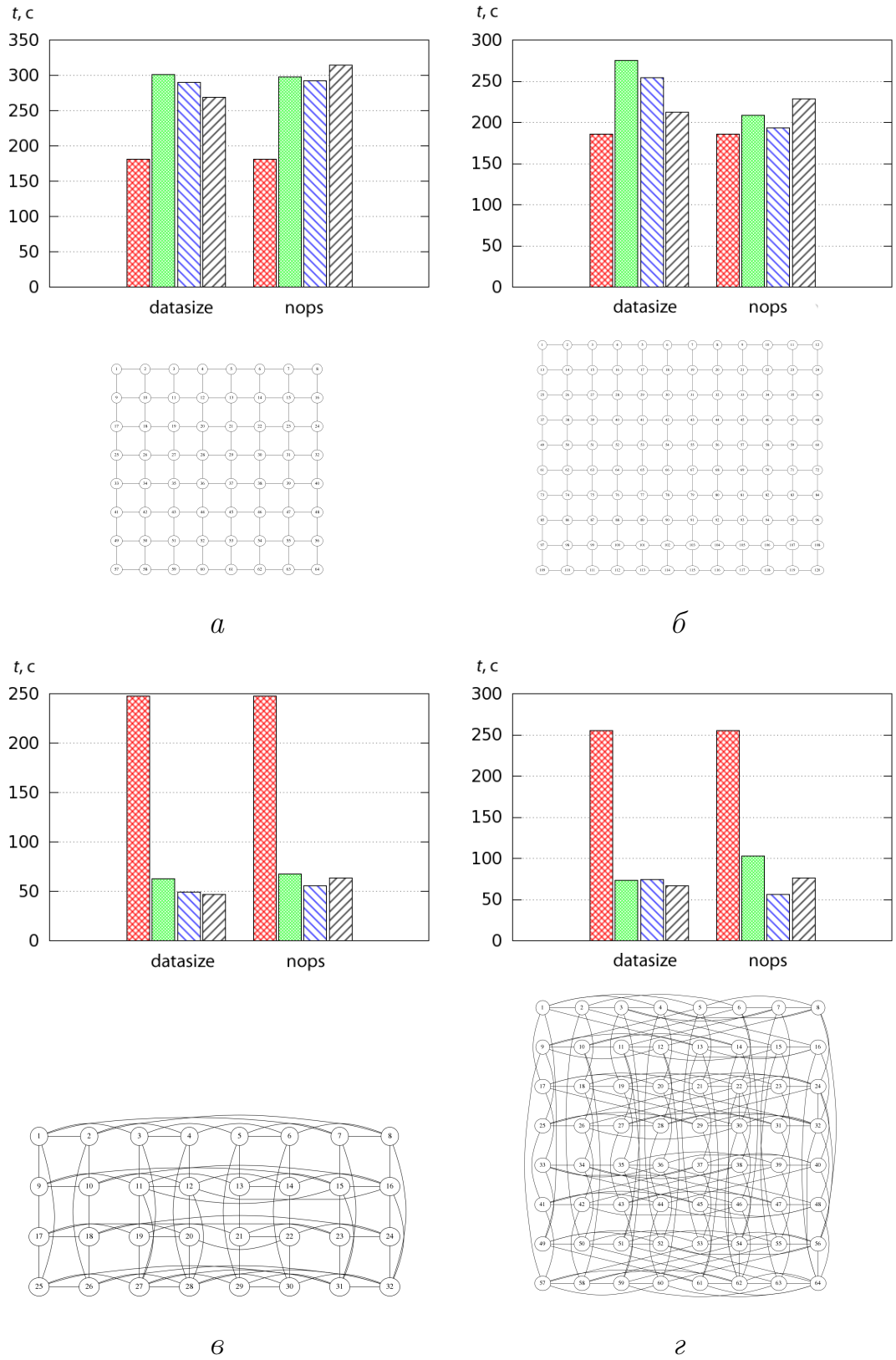



Рис. 3.15. Сравнение библиотек разбиения графов. Программы NPB LU и MG

$а$ – NPB LU, $r = 64$, $б$ – NPB LU, $r = 120$

$а$ – NPB MG, $r = 32$, $б$ – NPB MG, $r = 64$

– линейное вложение, – METIS, – Scotch, – gpart

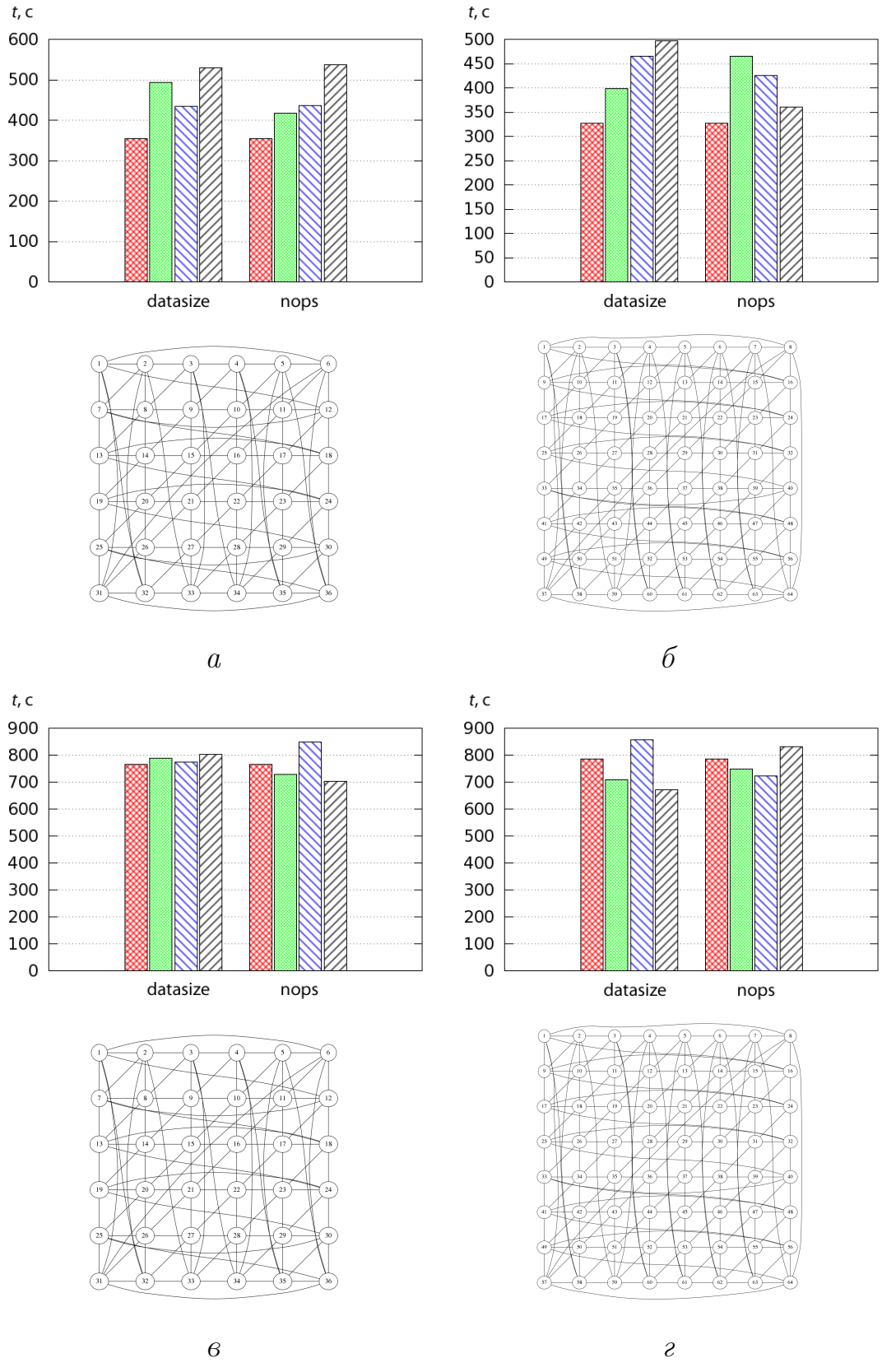
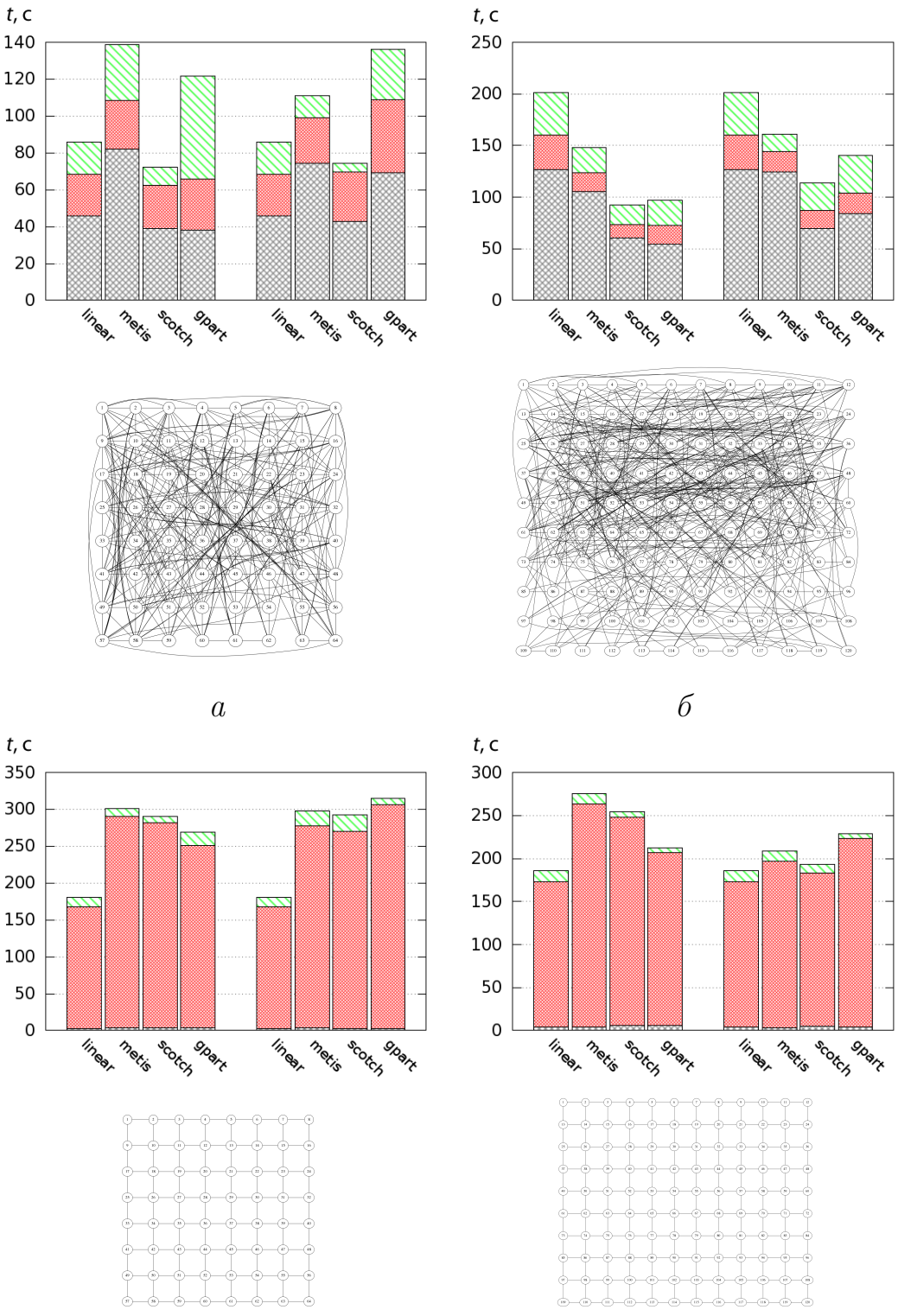



Рис. 3.16. Сравнение библиотек разбиения графов. Программа NPB BT и SP

*а* – NPB BT, $r = 36$, *б* – NPB BT, $r = 64$

*а* – NPB SP, $r = 36$, *б* – NPB SP, $r = 64$

– линейное вложение, – METIS, – Scotch, – gpart




*а* *б*

*в* *г*

Рис. 3.17. Время выполнения операций дифференцированных и коллективных обменов. Программы POP и NPB LU

*а* – POP, $r = 64$, *б* – POP, $r = 120$, *в* – NPB LU, $r = 64$, *г* – NPB LU, $r = 120$

– операции коллективных обменов, – операции дифференцированных обменов, – остальное время

# Глава 4. Пространственно-распределённая мультикластерная ВС

## 4.1. Архитектура пространственно-распределённой мультикластерной ВС

Центром параллельных вычислительных технологий ФГОБУ ВПО "СибГУТИ" совместно с Лабораторией ВС ИФП СО РАН создана и развивается пространственно-распределённая мультикластерная вычислительная система (рис. 4.1). Система включает 10 пространственно-распределённых сегментов, представленных кластерными ВС. Кластеры A-H расположены в ЦПВТ ФГОБУ ВПО "СибГУТИ", а кластеры I, J – в Лаборатории ВС ИФП СО РАН. В качестве базовых составляющих для построения кластеров A-C, J использованы стандартные персональные компьютеры на базе процессоров компании Intel. Кластер I состоит из 6 двухпроцессорных узлов на базе процессоров AMD Opteron 248, кластер D – из 4 двухпроцессорных узлов на базе Intel Xeon 5150, кластер E – из 4 двухпроцессорных узлов на базе Intel Xeon 5345. Кластер F укомплектован 18 вычислительными узлами, оснащёнными двумя процессорами Intel Xeon 5420. Кластер G включает 4 вычислительных узла, оснащённые двумя процессорами Intel Xeon 5410, кластер H – 7 двухпроцессорных узлов на базе Intel Xeon 5620.

Коммуникационная среда мультикластерной ВС построена на основе локальных сетей (Infiniband QDR, Gigabit Ethernet), и глобальной сети Internet (технология VPN). Связь осуществляется через выделенные серверы сегментов ЦПВТ ФГОБУ ВПО "СибГУТИ" и Лаборатории ВС ИФП СО РАН. Система включает более 300 процессорных ядер и имеет пиковую производительность несколько TeraFLOPS. Мультикластерная ВС допускает масштабирование путём организации взаимодействия с другими системами.

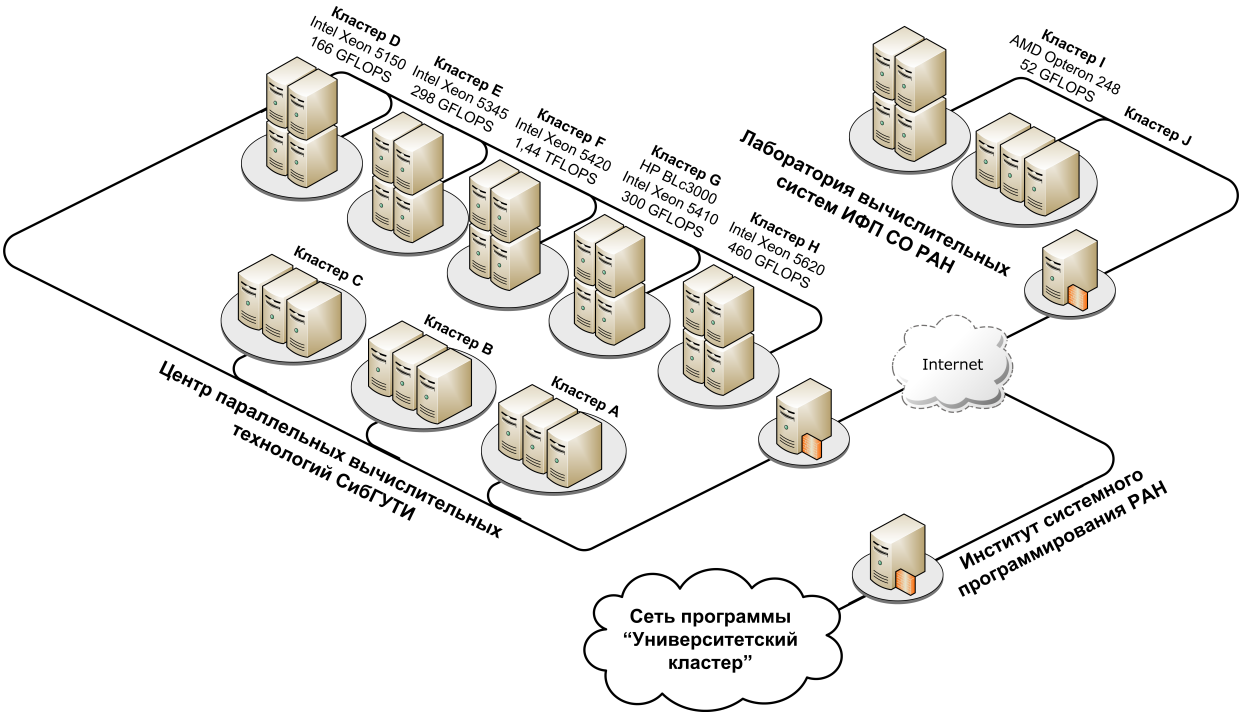



Рис. 4.1. Пространственно-распределённая мультикластерная ВС ЦПВТ ФГОБУ ВПО "СибГУТИ" и Лаборатории ВС ИФП СО РАН

## 4.2. Программное обеспечение мультикластерной ВС

### 4.2.1. Стандартные компоненты

Системное программное обеспечение мультикластерной ВС (рис. 4.2) основано на операционной системе GNU/Linux. На вычислительных узлах системы используются дистрибутивы CentOS и Fedora.

Поддерживается разработка и выполнение параллельных программ в стандартах OpenMP [169] (установлены компиляторы с языков C/C++, FORTRAN компаний Intel, Sun Microsystems и проекта GNU) и MPI (пакеты MPICH2 [170] и Open MPI [165]). Также установлены компиляторы языков семейства Partition Global Address Space (PGAS) [171]: Cray Chapel [172], IBM X10 [173] и Unified Parallel C [174].

Мультипрограммные режимы функционирования организуются системами пакетной обработки заданий TORQUE [175] и SLURM [176] и оригинальны-

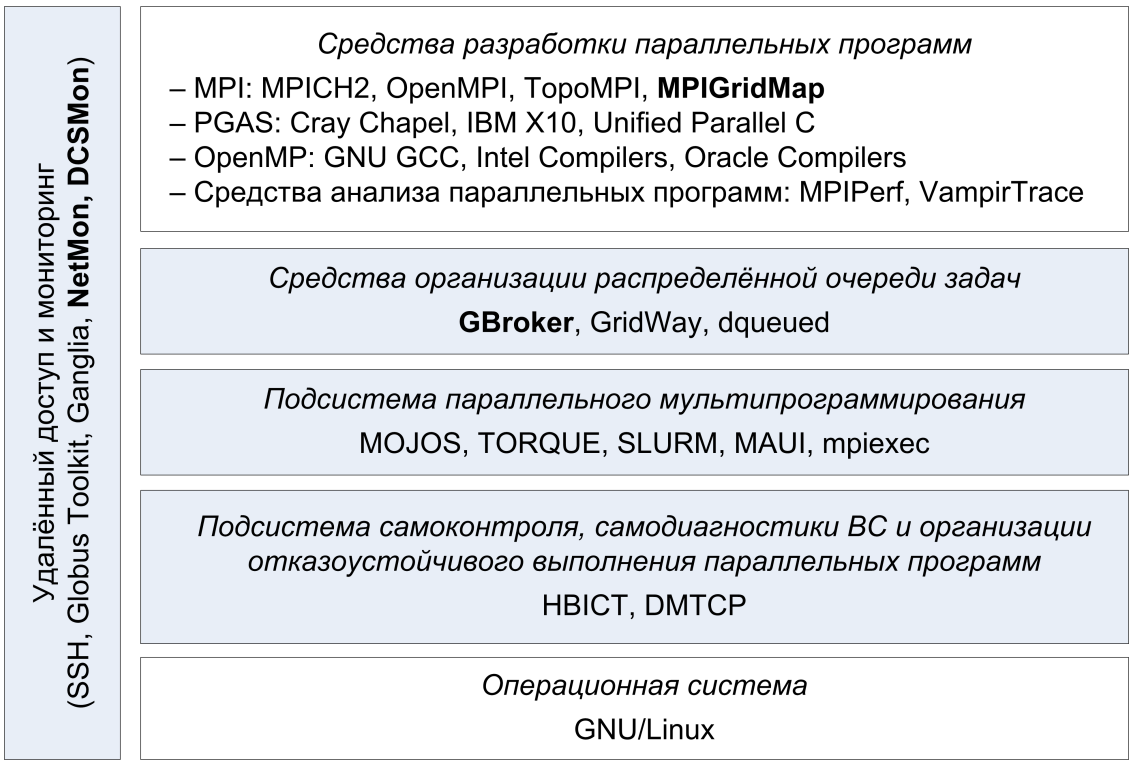



Рис. 4.2. Структура программного обеспечения пространственно-распределенной мультикластерной ВС

- стандартные компоненты;

- подсистема параллельного мультипрограммирования

ми средствами, разработанными в ЦПВТ ФГОБУ ВПО “СибГУТИ” [97, 177, 178]. К последним относятся:

– программный пакет MOJOS (MOldable JOb Scheduling) моделирования и отладки алгоритмов формирования расписаний решения масштабируемых задач на распределённых ВС;

– пакет DMTCP (Distributed MultiThreaded Checkpointing) создания контрольных точек восстановления программ и программный инструментарий HBICT сжатия контрольных точек.

Кроме того, пользователям доступна оригинальная библиотека TopoMPI коммуникационных функций, ориентированная на эффективную реализацию параллельных программ на современных мультиархитектурных распределённых ВС с иерархической структурой.

Основу промежуточного программного обеспечения мультикластерной ВС составляет программный комплекс Globus Toolkit [25, 86]. Его компоненты отвечают за авторизацию пользователей на основе сертификатов стандарта X.509, обеспечение интерфейса с локальными СУР подсистем (средствами подсистемы GRAM) и передачу файлов между подсистемами (средствами службы GridFTP). На сегменте F сконфигурирован централизованный GRID-диспетчер GridWay [30–34, 44, 90].

Для выполнения параллельной программы на мультикластерной ВС пользователь формирует ресурсный запрос (например, на языке Job Submission Description Language - JSDL). После этого пользователь отправляет задачу централизованному диспетчеру (GridWay) или одному из децентрализованных диспетчеров (GBroker), которые выполняют диспетчеризацию задачи и выделяют подсистему элементарных машин для выполнения программы. Диспетчеры взаимодействуют с СУР (TORQUE, SLURM и т.д.) и файловыми подсистемами кластерных ВС посредством промежуточного ПО Globus Toolkit 5. После выполнения программы выполняется доставка выходных файлов на подсистемы, указанные пользователем в ресурсном запросе (рис. 4.3).

### 4.2.2. Выполнение параллельных программ на мультикластерной ВС

Опишем подробнее действия пользователя при выполнении программ на мультикластерной ВС. Для того, чтобы начать пользоваться системой, пользователь должен пройти регистрацию. Для этого он отправляет запрос системному администратору. Администратор создает учётные записи пользователя на сегментах, после чего генерирует сертификат, размещает его на сервере MyProxy и отправляет сертификат и пароль по почте пользователю. Для работы в GRID-системе пользователь или устанавливает на своей машине клиентские утилиты пакета Globus Toolkit, или заходит по протоколу SSH на один из сегментов.

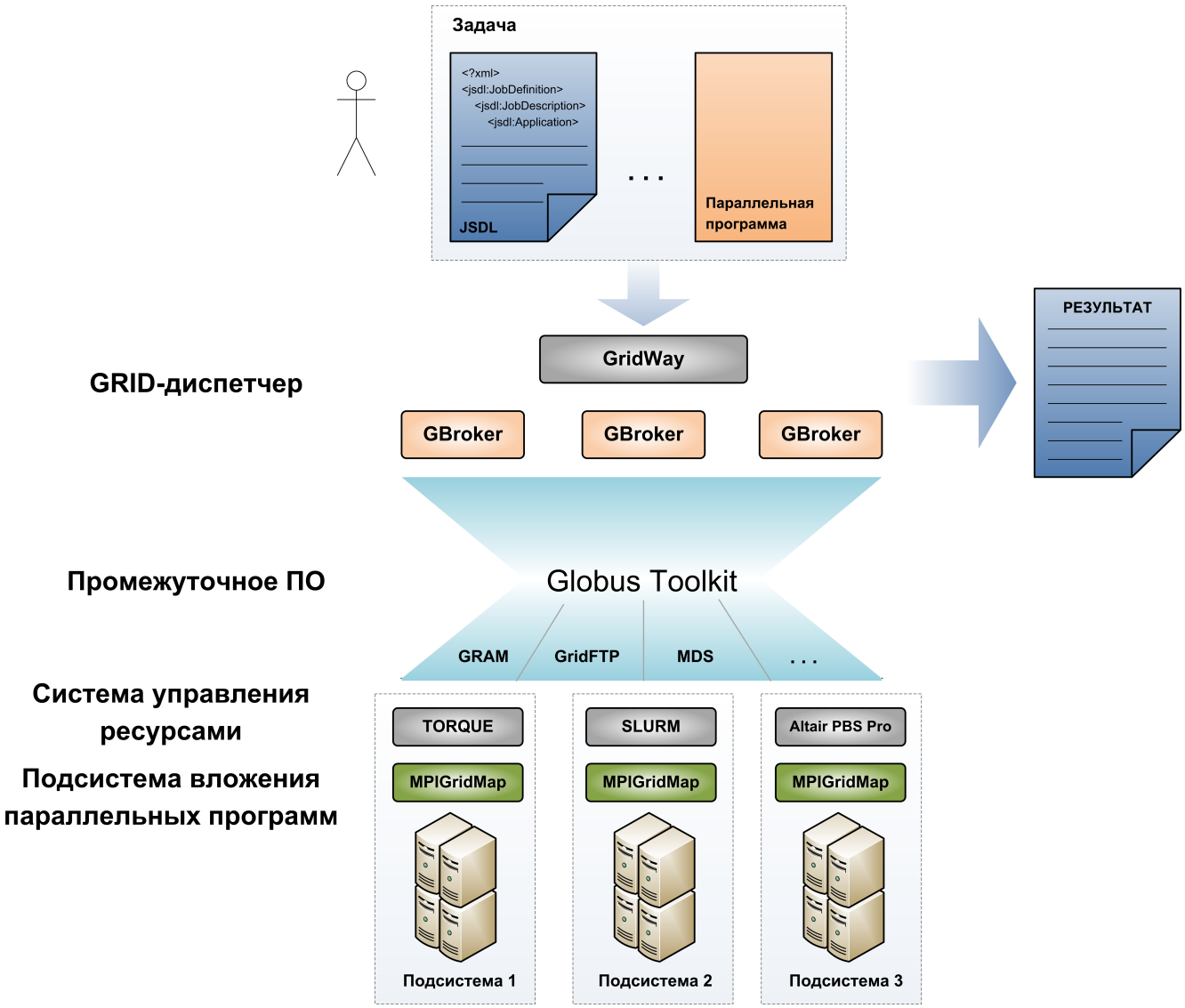



Рис. 4.3. Функциональная схема GRID-системы с расширенной конфигурацией

Сеанс работы пользователя в системе выглядит следующим образом. Пользователь начинает сеанс с выполнения аутентификации.

```
[griduser@xeon16 ~]$ myproxy-logon -s xeon80.cpct.sibsutis.ru
Enter MyProxy pass phrase:
A credential has been received for user griduser in /tmp/x509up_u565.
```

При этом ему выдается временный сертификат. После аутентификации пользователь получает беспарольный доступ к ресурсам системы. Также он может передавать данные между подсистемами

```
[griduser@xeon16 ~]$ globus-url-copy \
gsiftp://xeon80.cpct.sibsutis.ru/tmp/newdata \
gsiftp://xeon32.cpct.sibsutis.ru/tmp/inputdata
```

или между локальной машиной и любыми из подсистем.

```
[griduser@xeon16 ~]$ globus-url-copy file:///tmp/data \
gsiftp://xeon80.cpct.sibsutis.ru/tmp/newdata
```

Пакет Globus Toolkit позволяет пользователю запускать задачи на заданном сегменте мультикластерной ВС. Для этого он компилирует параллельную программу и создает паспорт задачи на языке ресурсных запросов RSL (Resource Specification Language), в котором описывает требуемые ресурсы.

```
[griduser@xeon16 ~]$ cat job.rsl
&(rsl_substitution = (GRIDFTP_XEON32 gsiftp://xeon32.cpct.sibsutis.ru)
                     (GRIDFTP_XEON16 gsiftp://xeon16.cpct.sibsutis.ru))
 (executable       = $(GRIDFTP_XEON16)/$(HOME)/mpiprog)
 (arguments        = ./file.dat)
 (job_type         = mpi)
 (count            = 8)
 (stdout           = $(HOME)/mpiprog.stdout)
 (file_stage_in    = ($(GRIDFTP_XEON16)/$(HOME)/mpiprog.data
                      $(HOME)/file.dat))
 (file_stage_out   = ($(HOME)/mpiprog.stdout
                      $(GRIDFTP_XEON16)/$(HOME)/mpiprog.res)
                     ($(HOME)/mpiprog.stdout
                      $(GRIDFTP_XEON32)/$(HOME)/mpiprog.res))
```

```
(file_clean_up    =  $(HOME)/mpiprog.stdout)
```

В данном случае, в RSL-файле указывается, что программе требуется 8 параллельных ветвей (параметр `count`). Также в ресурсном запросе указано имя исполняемого файла (`executable`), аргументы программы (`arguments`), название файла с результатами (`stdout`), входные (`file_stage_in`) и выходные `file_stage_out`) файлы.

После формирования ресурсного запроса пользователь командой `globusrun` ставит задачу в очередь на одной из доступных подсистем.

```
[griduser@xeon16 ~]$ globusrun -b -r xeon80.cpct.sibsutis.ru \
-f job.rsl
globus_gram_client_callback_allow successful
GRAM Job submission successful
https://xeon80.cpct.sibsutis.ru:44383/16073658407281/5308558837526/
GLOBUS_GRAM_PROTOCOL_JOB_STATE_STAGE_IN
GLOBUS_GRAM_PROTOCOL_JOB_STATE_PENDING
GLOBUS_GRAM_PROTOCOL_JOB_STATE_ACTIVE
```

В соответствии с ресурсным запросом, входные файлы с кластера Xeon16 должны быть доставлены на кластер Xeon80, где будет выполняться программа. После доставки входных файлов на кластер Xeon80 отправляется паспорт задачи вместе с исполняемыми файлами. Затем задача ставится в локальную очередь на подсистеме. Пользователь может получить текущий статус выполнения задачи по её идентификатору:

```
[griduser@xeon16 ~]$ globusrun -status \
https://xeon80.cpct.sibsutis.ru:44383/16073658407281/5308558837526/
[griduser@xeon16 ~]$ globusrun -status \
https://xeon80.cpct.sibsutis.ru:44383/16073658407281/5308558837526/
DONE
```

По окончании решения задачи выполняется фаза stage-out, в процессе которой выходные файлы доставляются с сегмента Xeon80 на сегмент Xeon32.

Альтернативой запуска задачи на определённой подсистеме средствами Globus Toolkit является использование GRID-диспетчера. После обзора существующих GRID-диспетчеров [30–70, 89], был сделан выбор в пользу диспетчера GridWay. Диспетчер GridWay установлен на сегменте F (Xeon80) мультикластерной ВС и поддерживает распределённую очередь GRID-задач. При поступлении задачи диспетчер на основе информации о загруженности сегментов системы принимает решение о выборе подсистемы для выполнения программы. Процедура запуска задачи через GridWay включает следующие шаги.

Пользователь заходит на любой сегмент системы по протоколу SSH и подготавливает ресурсный запрос в формате JDL (Job Description Language). Этот формат, как и RSL, поддерживает параметры входных (stage-in) и выходных (stage-out) файлов.

```
[griduser@xeon16 ~]$ cat job.jt
EXECUTABLE   =  mpiprog
ARGUMENTS    =  ./mpiprog.res
INPUT_FILES  =  gsiftp://xeon16.cpct.sibsutis.ru/.../mpiprog.data
STDOUT_FILE  =  gsiftp://xeon32.cpct.sibsutis.ru/.../mpiprog.res
TYPE         =  mpi
NP           =  8
```

После этого пользователь с помощью команды `gwsubmit` отправляет задачу в очередь диспетчера.

```
[griduser@xeon16 ~]$ gwsubmit job.jt
```

Для запуска параллельной программы диспетчер выбирает подсистему с необходимым количеством свободных ЭМ и ставит задачу в очередь этой подсистемы, выполняет процедуры доставки входных и выходных файлов. Пользователю, кроме того, доступны команды `gwps` просмотра текущего состояния всех

GRID-задач в системе и `gwhost` отображения информации о сегментах мультикластерной ВС.

### 4.2.3. Пакет GBroker децентрализованной диспетчеризации параллельных программ

В ЦПВТ ФГОБУ ВПО “СибГУТИ” и Лаборатории вычислительных систем ИФП СО РАН создан и развивается программный пакет GBroker [94–111] децентрализованной диспетчеризации параллельных задач в пространственно-распределённых ВС (рис. 4.4). Пакет разработан на языке ANSI C для операционной системы GNU/Linux. Он включает в себя диспетчер GBroker, клиентское приложение GClient и системы мониторинга NetMon и DCSMon. Модуль GBroker реализует алгоритмы децентрализованной диспетчеризации задач, взаимодействуя с локальными СУР через подсистему GRAM пакета Globus Toolkit. DCSMon – модуль мониторинга вычислительных ресурсов подсистем локальной окрестности диспетчера. NetMon – модуль мониторинга производительности каналов связи межу подсистемами.

Администратор устанавливает пакет на всех подсистемах, настраивает локальные окрестности диспетчеров и модулей DCSMon и NetMon. Для NetMon администратор также задаёт размеры тестовых сообщений и интервалы выполнения измерений времени доставки данных. Ниже приведён пример конфигурационного файла диспетчера GBroker:

```
<?xml version="1.0"?>
<gbroker-config>
    <host-list>
        <host>xeon16.vpn.local</host>
        <host>xeon32.vpn.local</host>
        <host>xeon80.vpn.local</host>
        <host>l400a1.vpn.local</host>
```

```
        <host>l400a2.vpn.local</host>
        <host>l400l1.vpn.local</host>
    </host-list>
</gbroker-config>
```

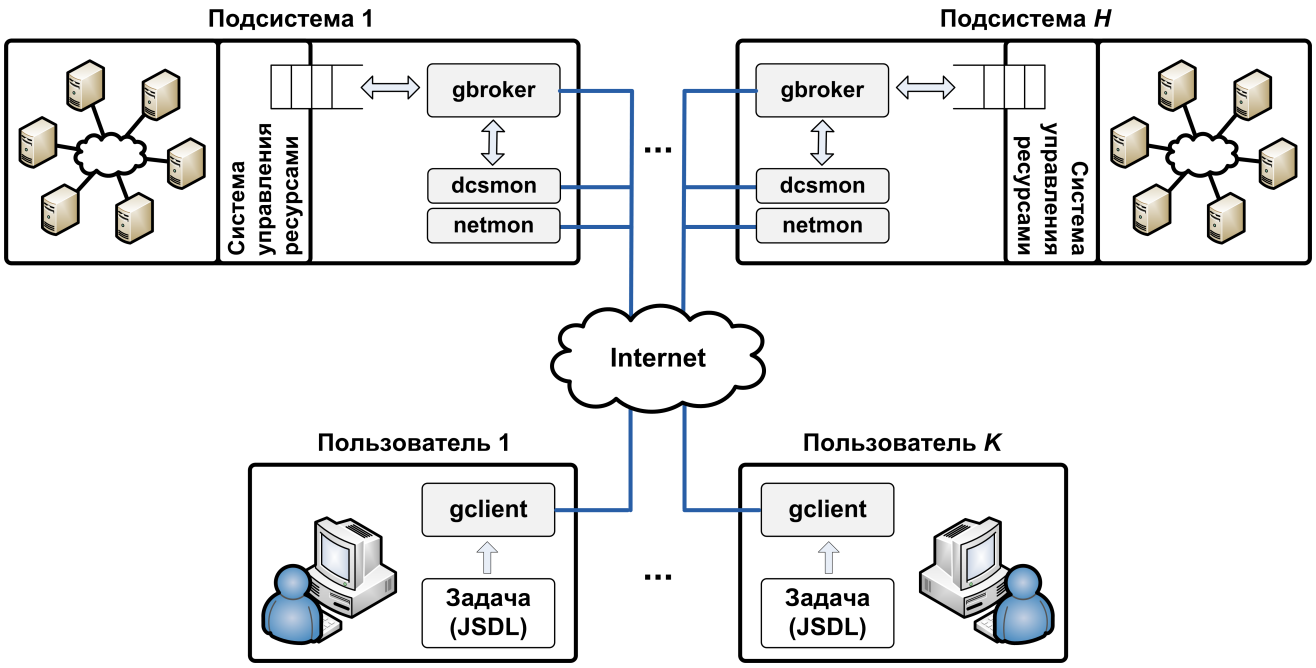
Алгоритм работы пользователя в системе следующий.

1) Пользователь выполняет аутентификацию в системе. После этого он компилирует программу

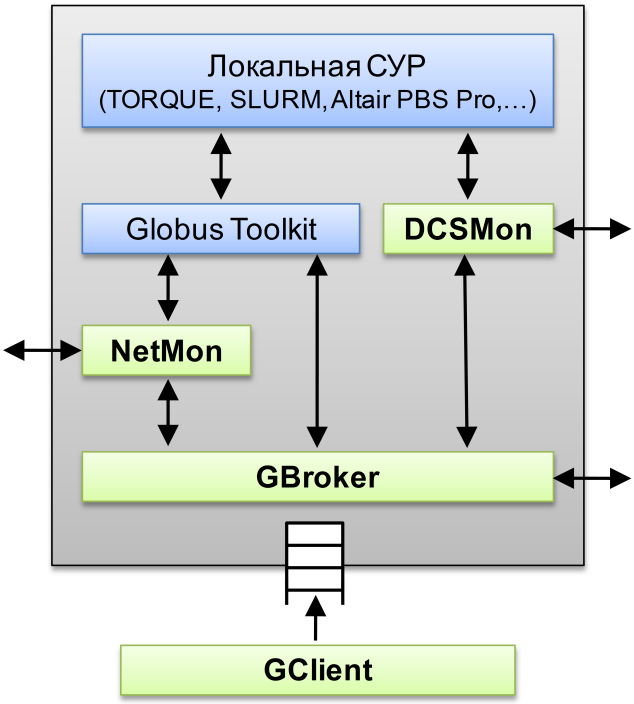
```
[griduser@xeon80 ~]$ mpicc -static mpiprog.c -o mpiprog
```

и формирует ресусный запрос на языке Job Submission Description Language (JSDL) [179]. Ниже представлен фрагмент ресурсного запроса для параллельной MPI-программы LAMMPS из 8 ветвей:

```
<?xml version="1.0" encoding="UTF-8"?>
<jsdl:JobDefinition>
  <jsdl:JobDescription>
    <jsdl:Application>
      <jsdl:ApplicationName>lammps</jsdl:ApplicationName>
      <jsdl-posix:POSIXApplication>
          <jsdl-posix:Executable>
          gsiftp://xeon80.vpn.local/.../lammps.${ARCH}
          </jsdl-posix:Executable>
          <jsdl-posix:Output>lammps.out</jsdl-posix:Output>
          <jsdl-posix:Error>lammps.err</jsdl-posix:Error>
      </jsdl-posix:POSIXApplication>
    </jsdl:Application>
    <jsdl:Resources>
      <jsdl:TotalCPUCount>
        <jsdl:Exact>8</jsdl:Exact>
```




*а*




*б*

Рис. 4.4. Функциональная структура пакета GBroker

```
      </jsdl:TotalCPUCount>
    </jsdl:Resources>
    <!-- ... -->
  </jsdl:JobDescription>
</jsdl:JobDefinition>
```

2) Пользователь отправляет задачу средствами GClient *любому* из диспетчеров GBroker распределённой ВС.

```
gclient -h xeon80.vpn.local /home/griduser/.../lammps.jsdl
```

3) После этого диспетчер получает информацию у систем мониторинга NetMon и DCSMon о состоянии подсистем его локальной окрестности и времени доставки данных. Затем, в соответствии с выбранным алгоритмом диспетчеризации, GBroker ставит задачу в очередь одной или нескольких подсистем. При необходимости, выполняется доставка входных (stage-in) и выходных (stage-out) данных. Пользователь может получить информацию о статусе выполнения задач командой `gbps` и запросить выходные данные программы по её идентификатору командой `gbroker -o`.

### 4.2.4. Пакет MPIGridMap оптимизации вложения параллельных MPI-программ в мультикластерные ВС

Разработанные алгоритмы вложения параллельных программ реализованы в программном пакете MPIGridMap, который позволяет запускать MPI-программы на пространственно-распределённых ВС с субоптимальным распределением параллельных ветвей по элементарным машинам.

Схема использования пакета выглядит следующим образом (рис. 4.5). На первом шаге программа компилируется с подключенной библиотекой профилирования VampirTrace [166] и запускается с линейным вложением (по умолчанию). Результат профилирования – протокол *prog.otf* выполнения программы в

формате Open Trace Format (OTF). Данный протокол анализируется программой OTFParse для получения информационного графа *prog.csr* параллельной задачи в формате Compressed Sparse Row (CSR). Граф отражает количество (nops) или размеры (datasize) передаваемых сообщений в дифференцированных обменах (point-to-point) при выполнении MPI-программ.

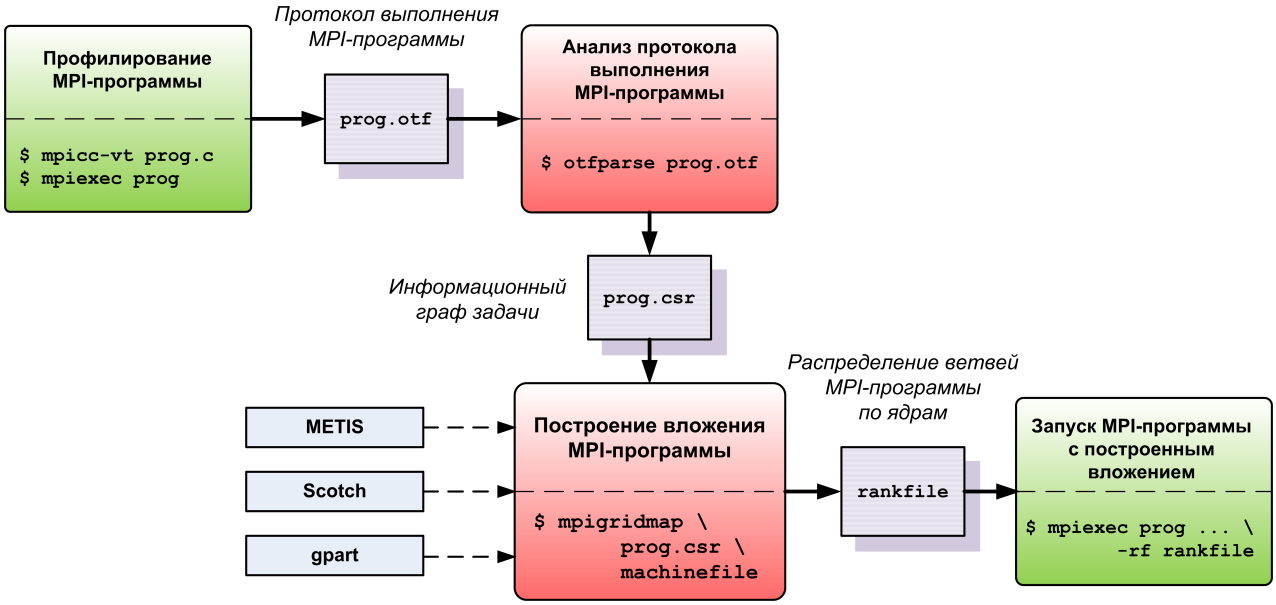



Рис. 4.5. Функциональная схема пакета MPIGridMap

Информационный граф подаётся вместе со списком *machinefile* узлов мультикластерной ВС на вход модуля MPIGridMap формирования вложения. При построении вложения могут быть использованы различные пакеты разбиения графов (METIS, Scotch, gpart). Результатом является файл *rankfile*, который содержит отображение параллельных ветвей программы на множество элементарных машин. Пользователь указывает файл *rankfile* при запуске MPI-программы. Далее приведён пример такого файла, описывающего вложение программы РОР в подсистему из 16 ЭМ.

```
rank 0=xeon16-node3-ipv6 slot=0
rank 1=xeon16-node4-ipv6 slot=0
rank 2=xeon80-node2-ipv6 slot=0
rank 3=xeon80-node2-ipv6 slot=1
```

```
rank 4=xeon16-node3-ipv6 slot=1
rank 5=xeon16-node4-ipv6 slot=1
rank 6=xeon80-node2-ipv6 slot=2
rank 7=xeon80-node2-ipv6 slot=3
rank 8=xeon80-node2-ipv6 slot=4
rank 9=xeon80-node2-ipv6 slot=5
rank 10=xeon16-node3-ipv6 slot=2
rank 11=xeon16-node4-ipv6 slot=2
rank 12=xeon16-node3-ipv6 slot=3
rank 13=xeon16-node4-ipv6 slot=3
rank 14=xeon80-node2-ipv6 slot=6
rank 15=xeon80-node2-ipv6 slot=7
```

Пакет MPIGridMap является свободно распространяемым. Он написан на языках C и C++ для операционной системы GNU/Linux.

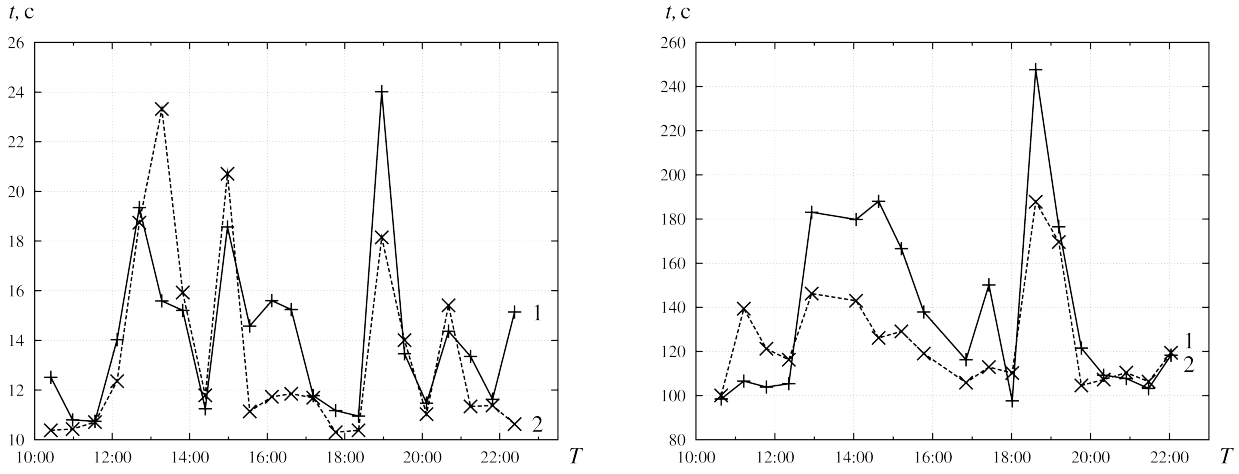
### 4.2.5. Оценка производительности каналов связи между подсистемами пространственно-распределённых ВС

При организации функционирования пространственно-распределённых мультикластерных ВС существенную роль играют средства оценки производительности каналов связи, предоставляющие информацию о времени доставки данных до подсистем. Данная информация учитывается при выборе подсистемы для выполнения параллельных программ.

В разработанный диссертантом пакет GBroker входит модуль NetMon прогнозирования производительности каналов связи. Он устанавливается на всех подсистемах и выполняет периодический сбор информации о времени передачи файлов различных размеров между подсистемами распределённой ВС. Администратор системы выполняет конфигурацию службы NetMon, задавая таблицу тестовых значений. Периодически производится измерение времени передачи

тестовых файлов средствами службы GridFTP из пакета Globus Toolkit. Прогноз времени передачи файла размера $m$ байт составляется на основе полученной таблицы путем кусочно-линейной интерполяции. Заметим, что при оценке времени передачи файлов учитываются особенности реализации протокола GridFTP (такие, как кэширование) [180, 181].

Выполнена оценка погрешности прогноза времени доставки данных. В эксперименте с интервалом 300 с в течение 12 часов выполнялись измерения при передаче файлов размерами 100 и 1000 Мбайт по протоколу GridFTP между различными подсистемами. Из графиков (рис. 4.6) видно, что относительное отклонение прогнозируемого времени передачи файлов от измеренного в среднем не превышает 35%, что является, по мнению автора, приемлемым в GRID-системах.




*а* *б*

Рис. 4.6. Прогноз времени $t$ передачи файлов между подсистемами Xeon80 и L400l (Gigabit Ethernet):
*а* – $m = 100$ Мбайт; *б* – $m = 1000$ Мбайт;
*1* – NetMon; *2* – GridFTP

## 4.3. Выводы

1. Описана функциональная структура пространственно-распределённой мультикластерной ВС, её текущая конфигурация и программное обеспечение.
2. Семейство алгоритмов децентрализованной диспетчеризации параллельных программ реализовано в программном пакете GBroker. Данный пакет включает в себя модули диспетчеризации и мониторинга. Его использование сокращает среднее время обслуживания параллельных программ, повышает пропускную способность ВС и позволяет реализовать живучее обслуживание потоков параллельных программ. Средства мониторинга позволяют получать информацию о загрузке подсистем и времени доставки файлов между подсистемами.
3. Разработан пакет MPIGridMap, выполняющий оптимизацию вложения параллельных MPI-программ в соответствии с созданными алгоритмами. Пакет обеспечивает формирование информационных графов программ и выполняет оптимизацию их вложения в мультикластерные ВС. При вложении учитываются все иерархические уровни коммуникационных сред пространственно-распределённых ВС.
4. Инструментарий децентрализованной диспетчеризации и вложения параллельных программ является неотъемлемой частью инструментария обеспечения живучести большемасштабных мультикластерных ВС.

# Заключение

Разработаны и исследованы алгоритмы и программные средства организации функционирования пространственно-распределённых мультикластерных ВС с иерархической структурой.

1. Предложены и исследованы алгоритмы децентрализованной диспетчеризации параллельных задач в мультикластерных ВС, обеспечивающие субминимальное время их обслуживания.

1.1. Построены алгоритмы децентрализованной диспетчеризации параллельных задач, минимизирующие время их обслуживания на мультикластерных ВС и увеличивающие пропускную способность систем. Разработанные алгоритмы позволяют минимизировать среднее время обслуживания параллельных программ и повысить пропускную способность системы. Выработаны рекомендации по применению предложенных алгоритмов.

1.2. Выполнено экспериментальное сравнение эффективности созданного инструментария децентрализованной диспетчеризации с методами централизованной диспетчеризации. Предложенные алгоритмы не менее эффективны, чем централизованные, и позволяют при этом обеспечить живучее обслуживание потоков параллельных задач.

1.3. Предложен стохастический алгоритм формирования субоптимальных структур локальных окрестностей децентрализованных диспетчеров. Сформированные структуры позволяют достичь субминимальных значений функции штрафа при обслуживании потоков параллельных задач.

1.4. Предложен алгоритм оценки времени доставки файлов между подсистемами мультикластерных и GRID-систем, учитывающий текущую загрузку каналов связи. Относительное отклонение прогнозируемого времени передачи файлов от измеренного в среднем не превышает 35%, что является приемлемым в пространственно-распределённых ВС.

2. Разработаны эвристические алгоритмы вложения параллельных программ в мультикластерные ВС с иерархической структурой.

2.1. Предложены алгоритмы вложения параллельных программ, учитывающие все иерархические уровни мультикластерных ВС и позволяющие сократить время выполнения параллельных программ от 1,1 до 5 раз по сравнению с линейным вложением. Алгоритмы эффективны для программ, имеющих разреженные информационные графы с преобладанием дифференцированных обменов. В основе алгоритмов лежит процедура рекурсивного разбиения информационного графа задачи.

2.2. Проведено натурное моделирование, которое показало преимущество алгоритмов вложения, учитывающих все коммуникационные уровни системы. Показано, что пакеты разбиения обеспечивают сопоставимую эффективность вложения параллельных программ. Сформулированы рекомендации по выбору пакета разбиения и способа формирования информационных графов.

3. Разработан программный инструментарий децентрализованной диспетчеризации параллельных программ в мультикластерных ВС, состоящий из клиентской программы, диспетчера и средств мониторинга каналов связи и подсистем. Созданы программные средства субоптимального вложения параллельных MPI-программ в мультикластерные ВС.

4. При участии диссертанта создана мультикластерная ВС, конфигурация которой расширена средствами децентрализованной диспетчеризации, обеспечивающими живучее обслуживание потоков параллельных задач. Система также оснащена средствами оптимизации вложения параллельных MPI-программ, позволяющими на основе информации о структуре информационных обменов и структуре ВС получать субоптимальные вложения параллельных программ. Основные результаты диссертации опубликованы в работах [94–111, 146–154].

# Литература


1. Евреинов, Э. В. Однородные вычислительные системы / Э. В. Евреинов, В. Г. Хорошевский. — Новосибирск: Наука. Сибирское отд-е, 1978. — 319 с.

2. Хорошевский, В. Г. Архитектура вычислительных систем / В. Г. Хорошевский. — М.: МГТУ им. Н. Э. Баумана, 2008. — 520 с.

3. Хорошевский, В. Г. Распределенные вычислительные системы с программируемой структурой / В. Г. Хорошевский // Вестник СибГУТИ. — 2010. — Т. 10, № 2. — С. 3–14.

4. Волков, Ю. М. Распределённая вычислительная система АСТРА / Ю. М. Волков, Ю. Ф. Ерофеев, В. И. Жиратков // В кн.: Вычислительные системы. — 1975. — Т. 63. — С. 132–139.

5. Бурцев, В. С. Параллелизм вычислительных процессов и развитие архитектур суперЭВМ / В. С. Бурцев. — М.: ИВВС РАН, 1997. — 352 с.

6. Евреинов, Э. В. Однородные универсальные вычислительные системы высокой производительности / Э. В. Евреинов. — Новосибирск: Наука, 1966. — 308 с.

7. Евреинов, Э. В. О возможности построения вычислительных систем высокой производительности / Э. В. Евреинов, Ю. Г. Косарев. — Новосибирск: изд. СО АН СССР, 1962. — 39 с.

8. Поспелов, Д. А. Введение в теорию вычислительных систем / Д. А. Поспелов. — М.: Советское радио, 1972. — 280 с.

9. Бабаян, Б. А. Многопроцессорные ЭВМ и методы их проектирования / Б. А. Бабаян, А. В. Бочаров, А. С. Волин. — М.: Высшая школа, 1990. — 143 с.

10. Бетелин, В. Б. Архитектура цифровых процессоров обработки сигналов / В. Б. Бетелин, Е. В. Грузинова, А. А. Кольцова. — М.: Рос. АН, Науч. совет по комплекс. пробл. "Кибернетика 1993. — 20 с.

11. Воеводин, В. В. Параллельные вычисления / В. В. Воеводин, В. В. Воеводин. — СПб.: БХВ-Петербург, 2002. — 608 с.

12. Дмитриев, Ю. К. Вычислительные системы из мини-ЭВМ / Ю. К. Дмитриев, В. Г. Хорошевский. — М.: Радио и связь, 1982. — 304 с.

13. Евреинов, Э. В. Однородные вычислительные системы, структуры и среды / Э. В. Евреинов. — М.: Радио и связь, 1981. — 208 с.

14. Параллельный компьютер с программируемой под структуру задачи архитектурой / А. В. Каляев, И. А. Каляев, И. И. Левин, И. М. Пономарев // Труды шестого международного семинара Распределенная обработка информации. — Новосибирск: 1998. — С. 25–29.

15. Каляев, А. В. Модульно-наращиваемые многопроцессорные системы со структурно-процедурной организацией вычислений / А. В. Каляев, И. И. Левин. — М.: Янус-К, 2003. — 380 с.

16. Каляев, И. А. Реконфигурируемые мультиконвейерные вычислительные структуры / И. А. Каляев. — Ростов-на-Дону: ЮНЦ РАН, 2003. — 320 с.

17. Корнеев, В. В. Архитектура вычислительных систем с программируемой структурой / В. В. Корнеев. — Новосибирск: Наука, 1985. — 164 с.

18. Корнеев, В. В. Вычислительные системы / В. В. Корнеев. — М.: Гелиос АРВ, 2004. — 512 с.

19. Левин, В. К. Высокопроизводительные мультипроцессорные системы / В. К. Левин // Информационные технологии и вычислительные системы. — 1995. — № 1. — С. 12–21.

20. Левин, И. И. Алгоритм коммутации элементов многопроцессорной системы со структурно-процедурной организацией вычислений / И. И. Левин, Л. М. Сластен // Материалы Всероссийской научно-технической конференции “Методы и средства обработки информации”. — 2003. — С. 119–124.

21. Миренков, Н. Н. Параллельное программирование для многомодульных вычислительных систем / Н. Н. Миренков. — М.: Радио и связь, 1989. — 319 с.

22. Прангишвили, И. В. Параллельные вычислительные системы с общим управлением / И. В. Прангишвили, С. . Виленкин, И. Л. Медведев. — М.: Энергопромиздат, 1983. — 313 с.

23. Прангишвили, И. В. Многопроцессорные и локальные сети микро-ЭВМ в распределенных системах управления / И. В. Прангишвили. — М.: Энергоатомиздат, 1985. — 272 с.

24. Г. Е. Пухов В. Ф. Евдокимов, М. В. С. Разрядно-аналоговые вычислительные системы / М. В. С. Г. Е. Пухов, В. Ф. Евдокимов. — М.: Советское радио, 1978. — 255 с.

25. Foster, I. Globus: A Metacomputing Infrastructure Toolkit / I. Foster, C. Kesselman // Intl J. Supercomputer Applications. — 1997. — Vol. 11, no. 3. — P. 115–128.

26. Foster, I. The Grid: Blueprint for a New Computing Infrastructure / I. Foster, C. Kesselman. — Morgan-Kaufmann, 1998.

27. Foster, I. The Anatomy of the Grid: Enabling Scalable Virtual Organizations / I. Foster // Proceedings of the 7th International Euro-Par Conference Manchester on Parallel Processing. — 2001. — P. 1–4.

28. Foster, I. What is the Grid? A Three Point Checklist [Электронный ресурс]. — 2002. Режим доступа: http://www-fp.mcs.anl.gov/~foster/Articles/WhatIsTheGrid.pdf, свободный (дата обращения 28.01.2013).

29. Foster, I. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration / I. Foster // Open Grid Service Infrastructure WG, Global Grid Forum. — 2002.

30. Huedo, E. A framework for adaptive execution on grids / E. Huedo, R. S. Montero, I. M. Llorente // Software – Practice and Experience (SPE). — 2004. — Vol. 34. — P. 631–651.

31. Huedo, E. An Experimental Framework For Executing Applications in Dynamic Grid Environments: Tech. Rep. 2002-43 / E. Huedo, R. S. Montero, I. M. Llorente: ICASE, 2002.

32. Huedo, E. A recursive architecture for hierarchical grid resource management / E. Huedo, R. S. Montero, I. M. Llorente // Future Generation Computer Systems. — 2009. — Vol. 25. — P. 401–405.

33. Leal, K. A decentralized model for scheduling independent tasks in Federated Grids / K. Leal, E. Huedo, I. M. Llorente // Future Generation Computer Systems. — 2009. — Vol. 25. — P. 840–852.

34. Montero, R. S. Grid Resource Selection for Opportunistic Job Migration / R. S. Montero, E. Huedo, I. M. Llorente // 9th International Euro-Par Conference. — Vol. 2790. — 2003. — P. 366–373.

35. Berman, F. Adaptive computing on the grid using AppLeS / F. Berman, R. Wolski, H. Casanova // IEEE Trans. on Parallel and Distributed Systems. — Vol. 34. — 2003. — P. 369–382.

36. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments / H. Casanova, A. Legrand, D. Zagorodnov, F. Berman // In Proc. of the 9th Heterogeneous Computing Workshop (HCW'00). — 2000. — P. 394–363.

37. New Grid Scheduling and Rescheduling Methods in the GrADS Project / K. Cooper, A. Dasgupta, C. K. K. Kennedy et al. // Proc. of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04). — Vol. 34. — 2004. — P. 199–206.

38. VGrADS: enabling e-Science workflows on grids and clouds with fault tolerance / L. Ramakrishnan, C. Koelbel, Y. Kee et al. // In: SC'09 The International Conference for High Performance Computing, Networking, Storage and Analysis. — 2009. — 12 pp.

39. Buyya, R. Nimrod/G: An architecture for a resource management and scheduling system in a global computational Grid / R. Buyya, D. Abramson, J. Giddy // Proc. of the 4th International Conference on High Performance Computing in Asia-Pacific Region. — 2000. — P. 283–289.

40. Buyya, R. An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications / R. Buyya, J. Giddy, D. Abramson // Sweep Applications, The Second Workshop on Active Middleware Services (AMS 2000), In conjunction with HPDC 2001. — 2000. — 10 pp.

41. Buyya, R. A Computational Economy for Grid Computing and its Implementation in the Nimrod-G Resource Broker / R. Buyya, D. Abramson, J. Giddy // Future Generation Computer Systems (FGCS). — 2001. — Vol. 18, no. 8. — P. 1061–1074.

42. Condor-G: A computation management agent for multi-institutional grids / J. Frey, T. Tannenbaum, M. Livny et al. // Cluster Computing. — 2001. — Vol. 5. — P. 237–246.

43. Evaluation of Meta-scheduler Architectures and Task Assignment Policies for High Throughput Computing: Tech. Rep. 5576 / E. Normale, S. Lyon, E. Caron et al.: Ecole Normale Superieure de Lyon, 2005.

44. A Comparison Between two Grid Scheduling Philosophies: EGEE WMS and GridWay / J. L. Vazquez-Poletti, E. Huedo, R. S. Montero, I. M. Llorente // Multiagent and Grid Systems. — 2007. — Vol. 3, no. 4. — P. 429–439.

45. Andreetto, P. Practical approaches to grid workload and resource management in the EGEE project / P. Andreetto, S. Borgia, A. Dorigo // In CHEP '04: Proceedings of the Conference on Computing in High Energy and Nuclear Physics. — Vol. 2. — 2004. — P. 899–902.

46. Supporting Internet-Scale Multi-Agent Systems / N. J. E. Wijngaards, B. J. Overeinder, M. van Steen, F. M. T. Brazier // Data and Knowledge Engineering. — 2002. — Vol. 41. — P. 229–245.

47. Gradwell, P. Grid scheduling with agents. — 2003.

48. A Multiagent-Based Approach to the Grid-Scheduling Problem / M. Solar, J. Rojas, M. Mendoza et al. // CLEI Electronic Journal. — 2012. — Vol. 15, no. 2. — 16 pp.

49. Altameem, T. An agent-based approach for dynamic adjustment of scheduled jobs in computational grids / T. Altameem, M. Amoon // Journal of Computer and Systems Sciences International. — 2010. — Vol. 49, no. 5. — P. 765–772.

50. Multiagent Distributed Grid Scheduler / V. Korneev, D. Semenov, A. Kiselev et al. // Proceedings of the Federated Conference on Computer Science and Information Systems. — 2011. — P. 577–580.

51. SmartGRID: A Fully Decentralized Grid Scheduling Framework Supported by Swarm Intelligence / Y. Huang, A. Brocco, P. Kuonen et al. // Proceedings of the 2008 Seventh International Conference on Grid and Cooperative Computing. — 2008. — P. 160–168.

52. Pegasus: A framework for mapping complex scientific workflows onto distributed systems / E. Deelman, G. Singh, M.-H. Su et al. // Scientific Programming. — 2005. — Vol. 13, no. 3. — P. 219–237.

53. The Pegasus portal: web based grid computing / G. Singh, E. Deelman, G. Mehta et al. // Proceedings of the 2005 ACM symposium on Applied computing. — 2005. — P. 680–686.

54. Adaptive workflow processing and execution in pegasus / K. Lee, N. W. Paton, R. Sakellariou et al. // In 3rd Intl Workshop on Workflow Management and Applications in Grid Environments (WaGe08), in Proc. 3rd Intl. Conf. on Grid and Pervasive Computing Symposia/Workshops. — 2008. — P. 99–106.

55. Laszewski, G. Work Coordination for Grid Computing [Электронный ресурс]. Режим доступа: `http://cyberaide.googlecode.com/svn/trunk/papers/anl/vonLaszewski-wo` свободный (дата обращения 28.01.2013).

56. Taverna, reloaded / P. Missier, S. Soiland-Reyes, S. Owen et al. // SSDBM 2010. — 2010.

57. Taverna: a tool for building and running workflows of services / D. Hull, K. Wolstencroft, R. Stevens et al. // Nucleic Acids Research. — 2006. — Vol. 34. — P. 729–732.

58. Taverna: lessons in creating a workflow environment for the life sciences / T. Oinn, M. Greenwood, M. Addis et al. // Concurrency and Computation: Practice and Experience. — 2006. — Vol. 18, no. 10. — P. 1067–1100.

59. Grid Enabling Applications Using Triana / I. T. Matthew, M. Shields, I. Wang, R. Philp // In Workshop on Grid Applications and Programming Tools. — 2003.

60. Programming scientific and distributed workflow with Triana services: Research Articles / C. David, G. Gombas, A. Harrison et al. // Concurrency and Computation: Practice and Experience - Workflow in Grid Systems. — 2006. — Vol. 18, no. 10. — P. 1021–1037.

61. Distributed computing with Triana on the Grid: Research Articles / I. Taylor, I. Wang, M. Shields, S. Majithia // Concurrency and Computation: Practice and Experience. — 2005. — Vol. 17, no. 9. — P. 1197–1214.

62. ASKALON: A Grid Application Development and Computing Environment / T. Fahringer, R. Prodan, R. Duan et al. // Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing. — 2005. — P. 122–131.

63. Wieczorek, M. Scheduling of scientific workflows in the askalon grid environment / M. Wieczorek, R. Prodan, T. Fahringer // ACM SIGMOD Record Journal. — 2005. — Vol. 34. — P. 56–62.

64. Workflow enactment in iceni / S. Mcgough, L. Young, A. Afzal et al. // In UK e-Science All Hands Meeting. — 2004. — P. 894–900.

65. Scheduling Architecture and Algorithms within the ICENI Grid Middleware / L. Young, S. Mcgough, S. Newhouse, J. Darlington // In UK e-Science All Hands Meeting. — 2003. — P. 5–12.

66. Yu, J. Gridbus Workflow Enactment Engine.

67. Scientific workflow management and the Kepler system. Special issue: workflow in grid systems / B. Ludascher, I. Altintas, C. Berkley et al. // Concurr. Comput.: Pract. Exp. — 2006. — P. 1039–1065.

68. Kepler: An Extensible System for Design and Execution of Scientific Workflows / I. Altintas, C. Berkley, E. Jaeger et al. // IN SSDBM. — 2004. — P. 21–23.

69. GridAnt: A Client-Controllable Grid Workflow System / K. Amin, G. Laszewski, M. Hategan et al. // In 37th Hawai'i International Conference on System Science, Island of Hawaii, Big Island. — 2004. — P. 5–8.

70. Romberg, M. The UNICORE Grid Infrastructure / M. Romberg // Scientific Programming, Special Issue on Grid Computing. — 2002. — Vol. 10. — P. 2002.

71. Flynn, M. Very high-speed computing system / M. Flynn // Proc. of the IEEE. — Vol. 54. — 1966. — P. 1901–1909.

72. Flynn, M. Some Computer Organisations and Their Effectiveness / M. Flynn // IEEE Trans. Computers. — Vol. 9 (21). — 1972. — P. 948–960.

73. Гергель, В. П. Основы параллельных вычислений для многопроцессорных вычислительных систем / В. П. Гергель, Р. Г. Стронгин. — Нижний Новгород: Изд-во ННГУ, 2003. — 184 с.

74. Таненбаум, Э. Распределенные системы: принципы и парадигмы / Э. Таненбаум. — СПб.: Питер, 2003. — 877 с.

75. Хокни, Р. Параллельные ВС. Архитектура, программирование и алгоритмы / Р. Хокни, К. Джессоуп. — М.: Радио и связь, 1986. — 392 рр.

76. Монахов, О. Г. Параллельные системы с распределенной памятью: структуры и организация взаимодействий / О. Г. Монахов, Э. А. Монахова. — Новосибирск: ИВМиМГ СО РАН, 2000. — 242 с.

77. Монахов, О. Г. Параллельные системы с распределённой памятью: управление ресурсами и заданиями / О. Г. Монахов, Э. А. Монахова. — Новосибирск: ИВМиМГ СО РАН, 2001. — 168 с.

78. Малышкин, В. Э. Параллельное программирование мультикомпьютеров / В. Э. Малышкин, В. Д. Корнеев. — Новосибирск: НГТУ, 2006. — 296 с.

79. Элементы параллельного программирования / В. А. Вальковский, В. Е. Котов, А. Г. Марчук, Н. Н. Миренков. — М.: Радио и связь, 1983. — 240 с.

80. Языки и параллельные ЭВМ : сб. ст. — 1990.

81. Яненко, Н. Н. Параллельные вычисления в задачах математической физики на вычислительных системах с программируемой структурой / Н. Н. Яненко, В. Г. Хорошевский, А. Д. Рычков // Электронное моделирование. — 1984. — Т. 6, № 1. — С. 3–8.

82. Косарев, Ю. Г. Распараллеливание по циклам / Ю. Г. Косарев // Вычислительные системы. — 1967. — Т. 24. — С. 3–20.

83. Воеводин, В. В. Математические модели и методы в параллельных процессах / В. В. Воеводин. — М.: Наука, 1986. — 296 с.

84. Топорков, В. В. Модели распределенных вычислений / В. В. Топорков. — М.: ФИЗМАТЛИТ, 2004. — 320 с.

85. И. В. Панфилов, А. М. П. Вычислительные системы / А. М. П. И. В. Панфилов. — М.: Советское радио, 1980. — 302 с.

86. Сайт проекта Globus Toolkit [Электронный ресурс]. Режим доступа: `http://www.globus.org/toolkit/`, свободный (дата обращения 28.01.2013).

87. AL-Khateeb, A. Job Type Approach for Deciding Job Scheduling in Grid Computing Systems / A. AL-Khateeb, R. Abdullah, N. A. Rashid // Journal of Computer Science. — 2009. — Vol. 5. — P. 745–750.

88. Dong, F. Scheduling Algorithms for Grid Computing: State of the Art and Open Problems: Tech. Rep. 2006-504 / F. Dong, S. Akl: School of Computing, Queen's University Kingston, Ontario, 2006.

89. I. Gaweda, C. W. Grid Brokers and Metaschedulers. Market Overview: Tech. rep. / C. W. I. Gaweda: School of Computing, Queen's University Kingston, Ontario, 2006.

90. Сайт проекта GridWay [Электронный ресурс]. Режим доступа: `http://www.gridway.org/doku.php`, свободный (дата обращения 28.01.2013).

91. Decentralized Grid Scheduling with Evolutionary Fuzzy Systems / F. Alexander, C. Grimme, J. Lepping, A. Papaspyrou // Job Scheduling Strategies for Parallel Processing. — 2009. — P. 16–36.

92. Yu, J. A Taxonomy of Workflow Management Systems for Grid Computing: Tech. rep. / J. Yu, R. Buyya: Journal of Grid Computing, 2005.

93. Корнеев, В. В. Архитектура вычислительных систем с программируемой структурой / В. В. Корнеев. — Новосибирск: Наука, 1985. — 164 с.

94. Kurnosov, M. G. Efficiency analysis of decentralized grid scheduling with job migration and replication / M. G. Kurnosov, A. A. Paznikov // Proc. of ACM International Conference on Ubiquitous Information Management and Communication (IMCOM/ICUIMC). — 2013. — 7 pp.

95. Курносов, М. Г. Децентрализованные алгоритмы диспетчеризации пространственно-распределённых вычислительных систем /

М. Г. Курносов, А. А. Пазников // Вестник ТГУ. Управление, вычислительная техника и информатика. — 2012. — Vol. 18, no. 1. — P. 133–143.

96. Курносов, М. Г. Моделирование алгоритмов децентрализованного обслуживания потоков параллельных задач в GRID-системах / М. Г. Курносов, А. А. Пазников // Проблемы информатики. — 2012. — no. 2. — P. 45–54.

97. Масштабируемый инструментарий параллельного мультипрограммирования пространственно-распределенных вычислительных систем / В. Г. Хорошевский, М. Г. Курносов, С. Н. Мамойленко et al. // Вестник СибГУТИ. — 2011. — no. 4. — P. 3–19.

98. Курносов, М. Г. Инструментарий децентрализованного обслуживания потоков параллельных MPI-задач в пространственно-распределенных мультикластерных вычислительных системах / М. Г. Курносов, А. А. Пазников // Вестник ТГУ. Управление, вычислительная техника и информатика. — 2011. — Vol. 16, no. 3. — P. 78–85.

99. Курносов, М. Г. Децентрализованные алгоритмы управления ресурсами распределенных вычислительных и GRID-систем / М. Г. Курносов, А. А. Пазников // Материалы Международной конференции "Математические и информационные технологии, MIT-2011". — 2011. — 6 pp.

100. Курносов, М. Г. Моделирование алгоритмов децентрализованного обслуживания потоков параллельных задач в GRID-системах / М. Г. Курносов, А. А. Пазников // Материалы Российской научной конференции с участием зарубежных исследователей “Моделирование систем информатики”. — 2011. — 10 pp.

101. Курносов, М. Г. Исследование алгоритмов диспетчеризации задач в пространственно-распределенных вычислительных системах / М. Г. Курносов, А. А. Пазников // Материалы Российской научно-технической конференции “Информатика и проблемы телекоммуникаций”. — Vol. 1. — Новосибирск: СибГУТИ, 2011. — P. 199–200.

102. Пазников, А. А. Анализ алгоритмов диспетчеризации параллельных программ в пространственно-распределённых вычислительных системах / А. А. Пазников // Материалы XLIX Международной научной студенческой конференции “Студент и научно-технический прогресс”. — Новосибирск: НГУ, 2011. — P. 228.

103. Пазников, А. А. Алгоритмы диспетчеризации мультикластерных вычислительных систем на основе миграции и репликации параллельных MPI-программ / А. А. Пазников // Материалы всероссийской научной конференции молодых учёных “Наука. Технологии. Инновации”. — Vol. 1. — Новосибирск: НГТУ, 2011. — P. 63–66.

104. Курносов, М. Г. Программный пакет децентрализованного обслуживания потоков параллельных задач в пространственно-распределенных вычислительных системах / М. Г. Курносов, А. А. Пазников // Вестник СибГУТИ. — 2010. — Vol. 10, no. 2. — P. 79–86.

105. Курносов, М. Г. Моделирование алгоритмов децентрализованной диспетчеризации параллельных задач в пространственно-распределенных мультикластерных вычислительных системах / М. Г. Курносов, А. А. Пазников // Материалы XIII Российской конференции с участием иностранных ученых “Распределенные информационно-вычислительные ресурсы” (DICR-2010). — Новосибирск: ИВТ СО РАН, 2010. — 7 pp.

106. Инструментарий организации эффективного выполнения параллельных программ на распределенных вычислительных системах / М. Г. Курносов, А. А. Пазников, Ю. Е. Макарова, В. В. Апостолов // Материалы Всероссийской научно-технической конференции “Научное и технические обеспечение исследований и освоения шельфа Северного Ледовитого океана”. — 2010. — P. 49–53.

107. Курносов, М. Г. Об организации функционирования пространственно-распределенных мультикластерных вычислительных систем / М. Г. Курносов, А. А. Пазников // Материалы Российской научно-технической конференции “Информатика и проблемы телекоммуникаций”. — Vol. 1. — Новосибирск: СибГУТИ, 2010. — P. 159–161.

108. Курносов, М. Г. Децентрализованная диспетчеризация параллельных программ в распределенных вычислительных системах / М. Г. Курносов, А. А. Пазников // Материалы Девятой международной конференции “Высокопроизводительные параллельные вычисления на кластерных системах”. — Владимир: ВлГУ, 2009. — P. 260–265.

109. Пазников, А. А. Алгоритмы децентрализованной диспетчеризации параллельных программ в пространственно-распределённых вычислительных системах / А. А. Пазников // Материалы Пятой сибирской конференции по параллельным вычислениям. — 2009. — P. 161–165.

110. Пазников, А. А. Средства децентрализованной диспетчеризации задач в распределенных вычислительных системах / А. А. Пазников // Материалы XLVII Международной научной студенческой конференции “Студент и научно-технический прогресс”. — Новосибирск: НГУ, 2009. — P. 210.

111. Курносов, М. Г. Диспетчеризация параллельных задач в пространственно-распределенных вычислительных системах / М. Г. Курносов, А. А. Пазников // Материалы Российской научно-технической конференции “Информатика и проблемы телекоммуникаций”. — Vol. 1. — Новосибирск: СибГУТИ, 2009. — P. 125–127.

112. Page, E. S. On Monte-Carlo Methods in Congestion on Problems: Searching for an Optimum in Discrete Situations / E. S. Page // Operation Research. — 1965. — Vol. 13, no. 2. — P. 631–651.

113. Пазников, А. А. Стохастический алгоритм формирования локальных окрестностей децентрализованных диспетчеров мультикластерных систем / А. А. Пазников // Материалы Российской научно-технической конференции “Обработка информационных сигналов и математическое моделирование”. — Новосибирск: СибГУТИ, 2012. — P. 167–168.

114. Gabriel, E. Implementing MPI with optimized algorithms for metacomputing / E. Gabriel, M. Resch, R. Ruhle // In Proc. of the third MPI Developer’s and User’s Conference. — 1999. — P. 31–41.

115. Fernandez, E. Supporting efficient execution of MPI applications across multiple sites / E. Fernandez, E. Heymann, M. A. Senar // In Proc. of Euro-Par’2006. — 2006. — P. 383–392.

116. High performance relay mechanism for MPI communication libraries run on multiple private IP address clusters / R. Takano, M. Matsuda, T. Kudoh et al. // In Proc. of 8th IEEE international symposium on cluster computing and the grid (CCGRID 2008). — 2008. — P. 401–408.

117. Saito, H. Locality-aware Connection Management and Rank Assignment for Wide-area MPI / H. Saito, K. Taura // In Proc. of the 7th IEEE International

Symposium on Cluster Computing and the Grid (CCGRID 2007). — 2007. — P. 249–256.

118. An Architecture of Stampi: MPI Library on a cluster of parallel Computers / T. Imamura, Y. Tsujita, H. Koide, H. Takemiya // In Proc. of the 7th European PVM/MPI'2000. — 2000. — P. 200–207.

119. The NumGRID metacomputing system / N. V. Malyshkin, B. Roux, D. Fougere, V. E. Malyshkin // In Bulletin of the Novosibirsk Computing Center, series Computer Science. — 2004. — Vol. 21. — P. 57–68.

120. Филамофитский, М. П. Система поддержки метакомпьютерных расчетов X-Com: архитектура и технология работы / М. П. Филамофитский // Вычислительные методы и программирование. — 2004. — no. 5. — P. 123–137.

121. Anderson, D. P. Boinc: A system for public-resource computing and storage / D. P. Anderson // 5th IEEE/ACM International Workshop on Grid Computing. — 2004. — P. 4–10.

122. Bokhari, S. H. On the Mapping Problem / S. H. Bokhari // IEEE Transactions on Computers. — 1981. — Vol. 30, no. 3. — P. 207–214.

123. Хорошевский, В. Г. Алгоритмы распределения ветвей параллельных программ по процессорным ядрам вычислительных систем / В. Г. Хорошевский, М. Г. Курносов // Автометрия. — 2008. — Т. 44, № 2. — С. 56–67.

124. hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications / F. Broquedis, J. Clet-Ortega, S. Moreaud et al. // Int. Conference on Parallel, Distributed and Network-Based Processing (PDP2010). — 2010. — P. 180–186.

125. Mercier, G. Towards an Efficient Process Placement Policy for MPI Applications in Multicore Environments / G. Mercier, J. Clet-Ortega // Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface. — 2009. — P. 104–115.

126. Pellegrini, F. Distillating knowledge about SCOTCH / F. Pellegrini // Combinatorial Scientific Computing. — Dagstuhl Seminar Proceedings no. 09061. — 2009. — P. 180–186.

127. MPIPP: an automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters / H. Chen, W. Chen, J. Huang et al. // Proceedings of the 20th annual international conference on Supercomputing. — 2006. — P. 353–360.

128. Yu, H. Topology maping for Blue Gene/L supercomputer / H. Yu, I.-H. Chung, J. Moreira // In Proc. Of SC'06. — SC '06 no. 116. — 2006.

129. Bhanot, G. Optimizing task layout on the Blue Gene/L supercomputer / G. Bhanot // IBM Journal of Research and Development. — 2005. — Vol. 49, no. 2. — P. 489–500.

130. Mapping Communication Layouts to Network Hardware Characteristics on Massive-Scale Blue Gene Systems / P. Balaji, R. Gupta, A. Vishnu, P. Beckman // Special edition of the Springer Journal of Computer Science on Research and Development (presented at the International Supercomputing Conference (ISC)). — 2011. — Vol. 26. — P. 247–256.

131. Automated mapping of regular communication graphs on mesh interconnects / A. Bhatele, G. R. Gupta, L. V. Kale, I.-H. Chung // 2010 International Conference on High Performance Computing. — 2010. — P. 1–10.

132. Bhatele, A. Heuristic-Based Techniques for Mapping Irregular Communication Graphs to Mesh Topologies / A. Bhatele, L. V. Kale // 13th IEEE International

Conference on High Performance Computing and Communication. — 2011. — P. 765–771.

133. Jeannot, E. Near-optimal placement of MPI processes on hierarchical NUMA architectures / E. Jeannot, G. Mercier // Proceedings of the 16th international Euro-Par conference on Parallel processing: Part II. — Euro-Par'10. — 2010. — P. 199–210.

134. Mercier, G. Improving MPI applications performance on multicore clusters with rank reordering / G. Mercier, E. Jeannot // Proceedings of the 18th European MPI Users' Group conference on Recent advances in the message passing interface. — 2011. — P. 39–49.

135. Design of a scalable InfiniBand topology service to enable network-topology-aware placement of processes / H. Subramoni, S. Potluri, K. Kandalla et al. // Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. — 2012. — P. 70:1–70:12.

136. Bhatele, A. Dynamic topology aware load balancing algorithms for molecular dynamics applications / A. Bhatele, L. V. Kale, S. Kumar // In Proc. of the 2009 ACM International Conference on Supercomputing (ICS'09). — 2009. — P. 110–116.

137. Mapping Dense LU Factorization on Multicore Supercomputer Nodes / J. Lifflander, P. Miller, R. Venkataraman et al. // Parallel and Distributed Processing Symposium (IPDPS), 2012 IEEE 26th International. — 2012. — P. 596–606.

138. Traff, J. L. Implementing the MPI Process Topology Mechanism / J. L. Traff // Proceedings of the ACM/IEEE conference on Supercomputing. — 2002. — P. 1–14.

139. Karlsson, C. Optimizing Process-to-Core Mappings for Application Level Multi-dimensional MPI Communications / C. Karlsson, T. Davies, Z. Chen //

Proceedings of the 2012 IEEE International Conference on Cluster Computing. — 2012. — P. 486–494.

140. The scalable process topology interface of MPI 2.2 / T. Hoefler, R. Rabenseifner, H. Ritzdorf et al. // Concurr. Comput. : Pract. Exper. — 2011. — Vol. 23, no. 4. — P. 293–310.

141. Multi-core and network aware MPI topology functions / M. J. Rashti, J. Green, P. Balaji et al. // Proceedings of the 18th European MPI Users' Group conference on Recent advances in the message passing interface. — 2011. — P. 50–60.

142. Hoefler, T. Generic Topology Mapping Strategies for Large-scale Parallel Architectures / T. Hoefler, M. Snir // In Proc. of the 2011 ACM International Conference on Supercomputing (ICS'11). — 2011. — P. 75–85.

143. Hoefler, T. Optimized routing and process mapping for arbitrary network topologies. — 2012.

144. Process Mapping for MPI Collective Communications / J. Zhang, J. Zhai, W. Chen, W. Zheng // Proceedings of the 15th International Euro-Par Conference on Parallel Processing. — 2009. — P. 81–92.

145. Mapping applications with collectives over sub-communicators on torus networks / A. Bhatele, T. Gamblin, S. H. Langer et al. // SC '12 Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. — 2012. — P. 97:1–97:11.

146. Курносов, М. Г. Эвристические алгоритмы отображения параллельных MPI-программ на мультикластерные вычислительные и GRID-системы / М. Г. Курносов, А. А. Пазников // Вычислительные методы и программирование. — 2013. — no. 14. — P. 1–10.

147. Курносов, М. Г. Вложение параллельных программ в пространственно-распределённые вычислительные системы на основе методов разбиения графов / М. Г. Курносов, А. А. Пазников // Материалы 2-й Всероссийской научно-технической конференции “Суперкомпьютерные технологии” (СКТ-2012). — Ростов-на-Дону: Издательство Южного федерального университета, 2012. — P. 135–139.

148. Курносов, М. Г. Эвристические алгоритмы вложения параллельных MPI-программ в мультикластерные вычислительные системы / М. Г. Курносов, А. А. Пазников // Материалы Девятой российской конференции с международным участием “Новые информационные технологии в исследовании сложных структур”. — Томск: НТЛ, 2012. — P. 10.

149. Пазников, А. А. Эвристические алгоритмы вложения параллельных MPI-программ в мультикластерные вычислительные системы / А. А. Пазников // Материалы 50-й Международной научной студенческой конференции “Студент и научно-технический прогресс”: Программирование и вычислительные системы. — Новосибирск: НГУ, 2012. — P. 37.

150. Курносов, М. Г. Сравнительный анализ методов вложения параллельных MPI-программ в мультикластерные вычислительные системы / М. Г. Курносов, А. А. Пазников // Материалы Российской научно-технической конференции “Обработка информационных сигналов и математическое моделирование”. — Новосибирск: СибГУТИ, 2012. — P. 160–162.

151. Курносов, М. Г. Применение многоуровневых методов разбиения графов параллельных программ для оптимизации их вложения в мультикластерные вычислительные системы / М. Г. Курносов,

А. А. Пазников // Материалы Второй Всероссийской научно-технической конференции “Научное и технические обеспечение исследований и освоения шельфа Северного Ледовитого океана”. — 2012. — P. 109–113.

152. Пазников, А. А. Комбинаторный алгоритм вложения параллельных программ в вычислительные системы / А. А. Пазников // Материалы XLVI Международной научной студенческой конференции “Студент и научно-технический прогресс”. — Новосибирск: НГУ, 2008. — P. 218–219.

153. Пазников, А. А. Точный алгоритм вложения параллельных программ в структуры вычислительных систем / А. А. Пазников // Материалы Российской научно-технической конференции “Информатика и проблемы телекоммуникаций”. — Vol. 1. — Новосибирск: СибГУТИ, 2008. — P. 152–153.

154. Курносов, М. Г. Об оптимизации распределения ветвей параллельных MPI-программ по процессорным ядрам вычислительного кластера / М. Г. Курносов, А. А. Пазников // Материалы VII Международной конференции-семинара “Высокопроизводительные вычисления на кластерных системах”. — Нижний Новгород: ННГУ, 2007. — P. 218–225.

155. Хокни, Р. Параллельные ЭВМ / Р. Хокни, К. Джессоуп. — М.: Радио и связь, 1986. — 390 pp.

156. Hockney, R. The communication challenge for MPP: Intel Paragon and Meiko CS-2 / R. Hockney // Parallel Computing. — 1994. — Vol. 20, no. 3. — P. 389–398.

157. Гэри, М. Вычислительные машины и труднорешаемые задачи / М. Гэри, Д. Джонсон. — М.: Мир, 1982. — 416 с.

158. Garey, M. Some simplified NP-complete graph problems / M. Garey, D. Johnson, L. Stockmeyer // Theoretical Computer Science. — 2006. — P. 84–96.

159. Hendrickson, B. A multilevel algorithm for partitioning graphs / B. Hendrickson, R. Leland // Proc. of the 1995 ACM/IEEE conference on Supercomputing (CDROM). — Supercomputing '95 no. 28. — 1995. — P. 1–14.

160. Karypis, G. Multilevel k-way partitioning scheme for irregular graphs / G. Karypis, V. Kumar // Journal of Parallel and Distributed computing. — 1998. — Vol. 48. — P. 96–129.

161. Fiduccia, C. M. A linear-time heuristic for improving network partitions / C. M. Fiduccia, R. M. Mattheyses // Proc. of conference "Design Automation". — DAC '82. — 1982. — P. 175–181.

162. Efficient Message Passing on Multi-Clusters: An IPv6 Extension to Open MPI / C. Kauhaus, A. Knoth, T. Peiselt, D. Fey // Proceedings of KiCC'07, Chemnitzer Informatik Berichte. — 2007.

163. Challenges of MPI over IPv6 / A. Knoth, C. Kauhaus, D. Fey et al. // Proceedings of the Fourth International Conference on Networking and Services. — 2008. — P. 242–247.

164. Savola, P. Observations of IPv6 traffic on a 6to4 relay / P. Savola // SIGCOMM Comput. Commun. Rev. — 2005. — Vol. 35, no. 1. — P. 23–28.

165. Сайт проекта SLURM [Электронный ресурс]. Режим доступа: `http://www.open-mpi.org/`, свободный (дата обращения 28.01.2013).

166. Сайт проекта VampirTrace [Электронный ресурс]. Режим доступа: `http://www.tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zih/f` свободный (дата обращения 28.01.2013).

167. The Landscape of Parallel Computing Research: A View from Berkeley: Tech. Rep. UCB/EECS-2006-183 / K. Asanovic, R. Bodik, B. C. Catanzaro et al.:

Electrical Engineering and Computer Sciences, University of California, Berkeley, 2006.

168. Abou-Rjeili, A. Multilevel Algorithms for Partitioning Power-Law Graphs / A. Abou-Rjeili, G. Karypis // IEEE International Parallel and Distributed Processing Symposium (IPDPS). — IPDPS'06. — 2006.

169. Сайт проекта OpenMP [Электронный ресурс]. Режим доступа: `http://openmp.org/wp/`, свободный (дата обращения 28.01.2013).

170. Сайт проекта MPICH [Электронный ресурс]. Режим доступа: `http://www.mpich.org/`, свободный (дата обращения 28.01.2013).

171. Productivity and performance using partitioned global address space languages / K. Yelick, D. Bonachea, W.-Y. Chen et al. // Proceedings of the 2007 international workshop on Parallel symbolic computation. — 2007. — P. 24–32.

172. Chamberlain, B. L. Parallel Programmability and the Chapel Language / B. L. Chamberlain, D. Callahan, H. P. Zima // International Journal of High Performance Computing Applications. — 2007. — Vol. 21, no. 3. — P. 291–312.

173. X10: an object-oriented approach to non-uniform cluster computing / P. Charles, C. Grothoff, V. Saraswat et al. // Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. — 2005. — P. 519–538.

174. An evaluation of global address space languages: co-array fortran and unified parallel C / C. Coarfa, Y. Dotsenko, J. Mellor-Crummey et al. // Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming. — 2005. — P. 36–47.

175. Сайт проекта TORQUE [Электронный ресурс]. Режим доступа: http://www.adaptivecomputing.com/products/open-source/torque/, свободный (дата обращения 28.01.2013).

176. Сайт проекта SLURM [Электронный ресурс]. Режим доступа: https://computing.llnl.gov/linux/slurm/, свободный (дата обращения 28.01.2013).

177. Хорошевский, В. Г. Пространственно-распределённая мультикластерная вычислительная система: архитектура и программное обеспечение / В. Г. Хорошевский, М. Г. Курносов, С. Н. Мамойленко // Вестник ТГУ. — 2011. — Vol. 14, no. 1. — P. 79–84.

178. Архитектура и программное обеспечение пространственно-распределённых вычислительных систем / В. Г. Хорошевский, М. Г. Курносов, С. Н. Мамойленко, А. Ю. Поляков // Вестник СибГУТИ. — 2010. — no. 2. — P. 112–122.

179. Job Submission Description Language (JSDL) Specification [Электронный ресурс]. Режим доступа: http://www.ogf.org/documents/GFD.136.pdf, свободный (дата обращения 28.01.2013).

180. Ohsaki, H. Performance Evaluation of Data Transfer Protocol GridFTP for Grid Computing / H. Ohsaki, M. Imase // Enformatika. — 2006. — Vol. 16. — P. 297–302.

181. The Globus Striped GridFTP Framework and Server / W. Allcock, J. Bresnahan, R. Kettimuthu et al. // Proceedings of the 2005 ACM/IEEE conference on Supercomputing. — 2005. — P. 54–64.

# Приложение А.
# Пространственно-распределённая мультикластерная ВС

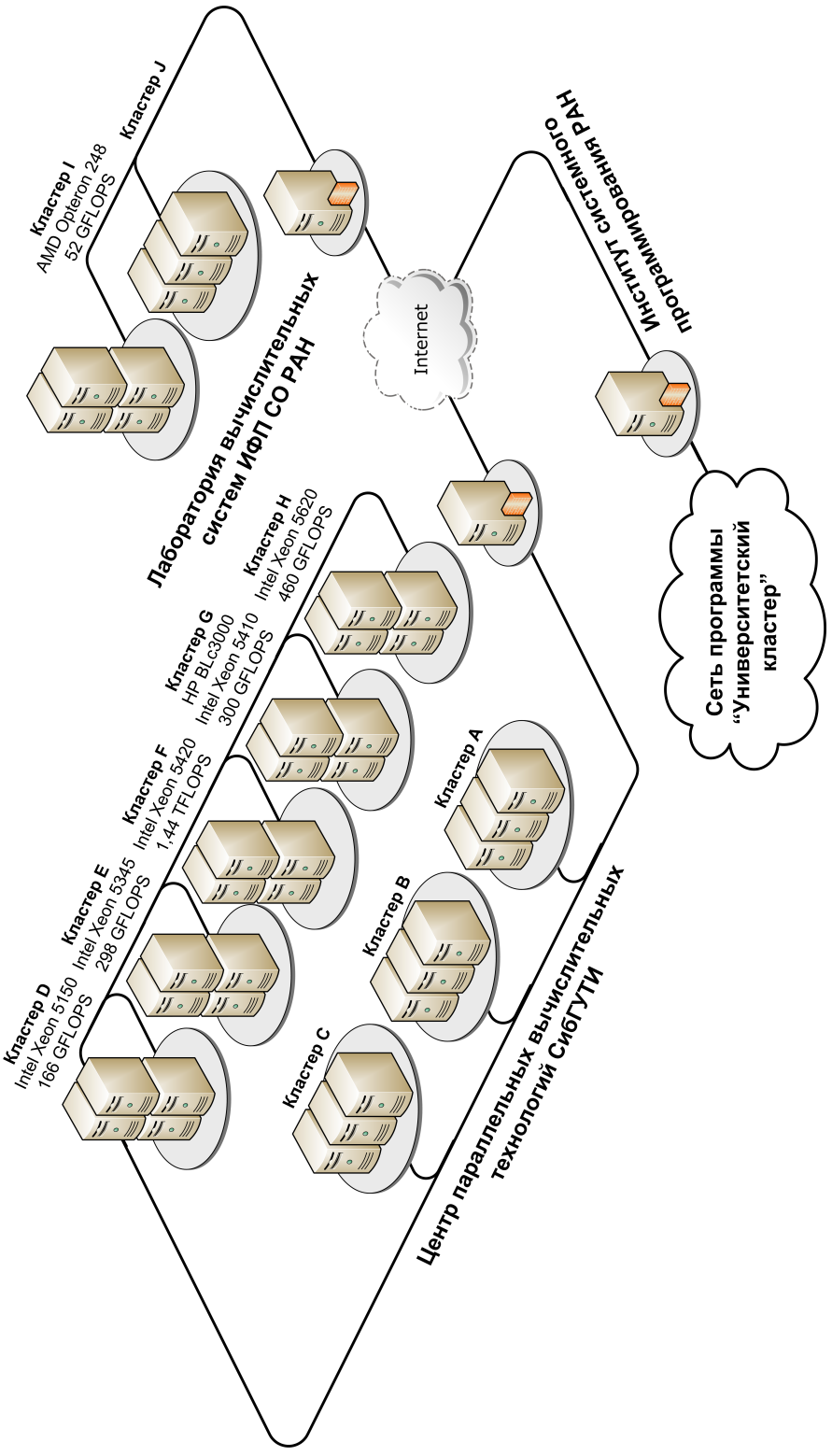



Рис. А.1. Пространственно-распределённая мультикластерная ВС ЦПВТ ФГОБУ ВПО "СибГУТИ" и Лаборатории ВС ИФП СО РАН