Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«ЮЖНО-УРАЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(национальный исследовательский университет)

На правах рукописи

ШАМАКИНА Анастасия Валерьевна

МЕТОДЫ УПРАВЛЕНИЯ РЕСУРСАМИ В ПРОБЛЕМНО-ОРИЕНТИРОВАННЫХ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛИТЕЛЬНЫХ СРЕДАХ

Специальность 05.13.11 – математическое и программное обеспечение
вычислительных машин, комплексов и компьютерных сетей

Диссертация на соискание ученой степени
кандидата физико-математических наук

Научный руководитель:
СОКОЛИНСКИЙ Леонид Борисович,
доктор физ.-мат. наук, профессор

Челябинск – 2014

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность темы

Развитие технологий распределенных вычислений в конце 1990-х годов позволило объединить географически-распределенные по всему миру гетерогенные ресурсы. Появились технические возможности для решения масштабных задач в области науки, техники и коммерции на территориально-распределенных ресурсах, принадлежащих разным владельцам. Исследования данной тематики привело к возникновению концепции *грид вычислений* (grid computing) [26, 43-45], и затем – к новой концепции *облачных вычислений* (cloud computing) [52, 64, 67, 76]. Для раскрытия всех потенциальных возможностей использования распределенных вычислительных ресурсов принципиально важно наличие результативных и эффективных алгоритмов планирования, используемых менеджерами ресурсов.

Управление ресурсами в традиционных гомогенных многопроцессорных системах (вычислительных кластерах) – хорошо изученный и проработанный вопрос. Существует большое количество менеджеров ресурсов для подобных систем [40]. Менеджеры ресурсов включаются в пакетные планировщики, в инструментарий для управления очередями заданий, в операционные системы. Эти менеджеры являются локальными, имеют полный контроль над ресурсами и реализуют механизмы и политики для эффективного использования данных изолированных ресурсов. Алгоритмы планирования для изолированных гомогенных многопроцессорных систем не могут также хорошо работать в распределенных вычислительных средах [21].

Главной задачей, которую решают технологии распределенных вычислений, является обеспечение доступа к глобально распределенным ресурсам с помощью специального инструментария. Сложность управления глобальными ресурсами заключается в том, что запуск, выполнение работы и доступ к необходимым данным могут производиться на различных компьютерах.

Глобальные распределенные вычислительные сети формируются из автономных ресурсов, конфигурация которых динамически изменяется. Кроме того, распределенные ресурсы могут принадлежать различным административным доменам, поэтому возникает проблема их администрирования, заключающаяся в согласовании различных политик. Еще одной немаловажной проблемой является гетерогенность ресурсов. Ранние работы [23, 50, 53, 63] в области управления ресурсами в распределенных вычислительных средах, фокусирующиеся на гетерогенности ресурсов, привели к созданию стандартных протоколов управления ресурсами и механизмов описания требований заданий к ресурсам. Однако практика показала, что эффективные методы и алгоритмы планирования для однородных изолированных многопроцессорных систем плохо адаптируются для распределенных гетерогенных систем [101]. Управление ресурсами в неоднородных распределенных вычислительных средах требует принципиально новых моделей вычислений и управления ресурсами.

В настоящее время перспективным является направление, связанное с применением распределенных вычислительных технологий для решения ресурсоемких научных задач в разных предметных областях: медицине, инженерном проектировании, нанотехнологиях, прогнозировании климата и др. Вычислительные задания в подобных предметных областях во многих случаях имеют *потоковую структуру* и могут быть описаны с помощью модели потока работ (workflow) [20], в соответствии с которой задание представляется в виде ориентированного ациклического графа, узлами которого являются задачи, являющиеся составными частями задания, а дуги соответствуют потокам данных, передаваемых между отдельными задачами. При этом набор задач, из которых строятся задания, является конечным и предопределенным. Проблемно-ориентированная специфика потоков работ в подобных сложных приложениях выражается в том, что в подавляющем большинстве

случаев, еще до выполнения задания, для каждой задачи могут быть получены оценки таких качественных характеристик, как время выполнения задачи на одном процессорном ядре, пределы масштабируемости и объем генерируемых данных. Использование подобных знаний о специфике задач в конкретной проблемно-ориентированной области может существенно улучшить эффективность методов управления вычислительными ресурсами. В настоящее время известно несколько программных систем, ориентированных на управление сложными приложениями с потоковой структурой в распределенных вычислительных средах. В качестве примера можно перечислить такие инструменты как Condor DAGMan [34], CoG [58], Pegasus [39], GridFlow [29] и ASKALON [91]. Существуют также алгоритмы планирования, использующие знания о проблемно-ориентированной специфике задач, составляющих вычислительное задание, такие как алгоритм Кима и Брауна [54], алгоритм DSC [96]. Однако данный класс алгоритмов применим для задач, выполняющихся на одном процессорном ядре некоторой многопроцессорной системы. В соответствие с этим актуальной является задача разработки методов и алгоритмов управления ресурсами в проблемно-ориентированных распределенных вычислительных средах, учитывающих специфику предметной области, масштабируемость отдельных задач в задании и использующих возможность параллельного выполнения независимых задач.

Цель и задачи исследования

Цель данной работы состояла в разработке методов и алгоритмов распределения ресурсов и планирования заданий, учитывающих специфику проблемно-ориентированных распределенных вычислительных сред, а также в разработке на их основе брокера ресурсов, который может быть использован в распределенных вычислительных системах. Для достижения этой цели необходимо было решить следующие задачи:

1. Разработать формальные методы представления проблемно-ориентированных распределенных вычислительных сред.
2. Разработать проблемно-ориентированный алгоритм планирования ресурсов для заданий, представляемых в виде потока работ.
3. Разработать архитектуру и принципы структурной организации брокера ресурсов для проблемно-ориентированных распределенных вычислительных сред.
4. Реализовать брокер ресурсов и провести вычислительные эксперименты для исследования эффективности предложенных подходов.

Методы исследования

В исследованиях, проводимых в диссертационной работе, использован математический аппарат, в основе которого лежит теория множеств и теория графов. Для представления заданий использована модель потока работ. При разработке брокера ресурсов применялись методы объектно-ориентированного проектирования и язык UML.

Научная новизна

1. Разработана оригинальная математическая модель проблемно-ориентированной распределенной вычислительной среды.
2. Предложен новый алгоритм планирования ресурсов, учитывающий специфику проблемно-ориентированной распределенной вычислительной среды.
3. Разработан брокер ресурсов, ориентированный на работу в проблемно-ориентированных распределенных вычислительных средах.

Теоретическая и практическая ценность

Теоретическая ценность работы состоит в том, что в ней дано формальное описание методов и алгоритмов управления ресурсами в проблемно-ориентированных распределенных вычислительных средах, включающее в

себя математические модели распределенной вычислительной системы и заданий с потоковой структурой.

Практическая ценность работы заключается в том, что на базе предложенных методов и алгоритмов разработан брокер ресурсов, позволяющий организовать эффективное использование ресурсов в проблемно-ориентированных распределенных вычислительных средах.

Структура и объем работы

Диссертация состоит из введения, четырех глав, заключения и библиографии. Объем диссертации составляет 106 страницы, объем библиографии – 107 наименований.

Содержание работы

Первая глава, «Технологии распределенных вычислений», посвящена описанию методов управления ресурсами в проблемно-ориентированных распределенных вычислительных средах. Рассматриваются современные подходы к планированию ресурсов. Дается общая классификация алгоритмов планирования, выполнен обзор наиболее популярных алгоритмов кластеризации.

Во второй главе, «Планирование в проблемно-ориентированных средах», предлагаются формальные методы для представления проблемно-ориентированных распределенных вычислительных сред, строится математическая метамодель проблемно-ориентированной распределенной вычислительной среды. Описывается новый проблемно-ориентированный алгоритм планирования ресурсов *POS* для заданий с потоковой структурой, ориентированный на распределенные вычислительные среды, формируемые на базе вычислительных кластеров с многоядерными ускорителями.

В третьей главе, «Реализация брокера ресурсов для проблемно-ориентированных сред», описывается процесс проектирования и реализа-

ции программной системы *DiVTB Broker*, представляющей собой брокер ресурсов для проблемно-ориентированных распределенных вычислительных сред. На базе модели вариантов использования специфицируются общие ключевые функции брокера. С помощью диаграмм классов и диаграмм последовательностей описываются внутренняя структура брокера ресурсов и алгоритмы, реализующие основные функции.

В четвертой главе, «Вычислительные эксперименты», приводятся результаты вычислительных экспериментов по исследованию адекватности и эффективности разработанных в диссертации моделей и методов планирования для проблемно-ориентированных вычислительных сред.

В заключении суммируются основные результаты диссертационной работы, выносимые на защиту, приводятся данные о публикациях и апробациях автора по теме диссертации, и рассматриваются направления дальнейших исследований в данной области.

В приложении приводятся основные сокращения и обозначения, используемые в диссертационной работе.

ГЛАВА 1. ТЕХНОЛОГИИ РАСПРЕДЕЛЕННЫХ ВЫЧИСЛЕНИЙ

1.1. Обзор технологий распределенных вычислений

1.1.1. Грид-вычисления

Вычислительный грид является программно-аппаратной инфраструктурой, которая обеспечивает надежный и прозрачный доступ к высокопроизводительным вычислительным ресурсам [43]. Грид представляет общую среду для развертывания инфраструктуры, ориентированной на сервисы, поддерживающей создание и совместное использование ресурсов распределенных организаций. Под ресурсами понимаются аппаратное обеспечение, инструментарий, программное обеспечение и данные, а также сервисы, подключенные посредством промежуточного слоя программного обеспечения и обеспечивающие безопасность, мониторинг, управление ресурсами и др.

При рассмотрении проблемы планирования в грид-средах для повышения уровня абстракции часто игнорируют такие компоненты инфраструктуры, как аутентификация, авторизация, обнаружение ресурсов и контроль доступа. В работе [19] приводится следующее адаптированное определение: «Грид – это тип параллельных и распределенных систем, которые обеспечивают совместное использование, выбор и динамическую агрегацию географически-распределенных автономных и гетерогенных ресурсов в зависимости от их доступности, характеристик, производительности, стоимости и требований качества обслуживания, предъявляемых пользователями».

С точки зрения функциональности можно выделить логическую архитектуру подсистемы планирования задач в грид. В работе [101] предложена общая архитектура данной подсистемы. Процесс планирования в распределенных вычислительных сетях может быть представлен тремя этапами: 1) обнаружение ресурсов и их фильтрация, 2) поиск подходящих ресурсов и

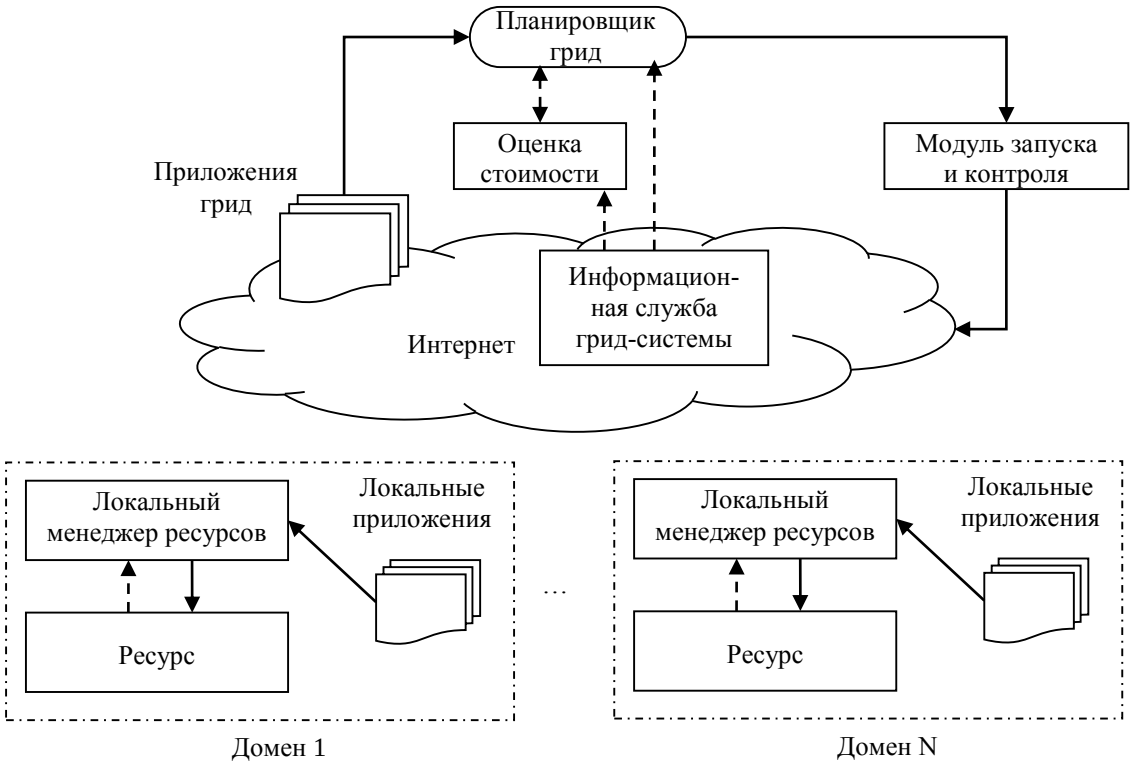

Рис. 1. Логическая архитектура системы планирования в грид.

планирование в соответствии с определенными целями, 3) выполнение задания [78]. На рис. 1 представлена модель системы планирования в распределенных вычислительных средах, в которой функциональные компоненты связывают два типа потоков данных: прерывистая линия определяет поток ресурсов/поток информации о приложении, прямая линия – поток задач/поток команд планирования задач.

Основная работа планировщика грид (GS – Grid Scheduler) заключается в том, что он принимает заявки на выполнение некоторых приложений пользователей, производит поиск подходящих ресурсов в соответствии с полученными от информационного сервиса грид-среды (GIS – Grid Information Service) данными и создает отображение приложений на ресурсы, основанное на целевых функциях и предсказании характеристик ресурсов. В отличие от планировщиков традиционных параллельных и распределенных систем,

грид-планировщики обычно не контролируют грид-ресурсы напрямую, а работают как брокеры или агенты [22]. Грид-планировщики не всегда находятся в одном домене с ресурсами, которые им доступны. На рис. 1 показан только один грид-планировщик, однако в действительности может быть развернуто несколько подобных планировщиков для формирования различных структур системы планирования (централизованной, иерархической и децентрализованной [48]) для решения проблем производительности и масштабируемости. Хотя планировщик уровня грид (иначе его называют метапланировщиком [65]) не является необходимым компонентом в инфраструктуре грид-среды (он не включен в Globus Toolkit [86]), но он имеет решающее значение для использования в грид-средах, которые имеют тенденцию быстро расширяться, добавляя в свой состав все новые и новые ресурсы от суперкомпьютеров до настольных компьютеров.

Информация о состоянии имеющихся ресурсов необходима грид-планировщику для составления приемлемого расписания работ, особенно в условиях гетерогенного и динамичного характера грид. Расписание часто представляется в виде диаграммы Ганта, горизонтальная ось которой представляет собой время, вертикальная ось – ресурсы в грид. На каждом из ресурсов прямоугольниками обозначаются задачи, ширина прямоугольника показывает длительность задачи. Роль информационного сервиса грид-среды заключается в предоставлении информации о состоянии имеющихся ресурсов планировщикам грид. Информационный сервис ответственен за сбор и предсказание информации о состоянии ресурса. Информационный сервис грид-среды также может отвечать на запросы, предоставляя информацию о ресурсе, или передавать информацию подписчикам. Примерами информационных сервисов в грид являются Globus Monitoring и Discovery System (MDS) [36].

Для составления приемлемого расписания помимо информации о ресурсе необходимо знать свойства приложений (приблизительное количество инструкций, требования к памяти и хранению, зависимости подзадач в задании) и производительность ресурсов для различных видов приложений. Информация о свойствах приложения может быть получена с помощью профилирования (AP – Application profiling), а измерение производительности ресурса для данного типа задания – с помощью компоненты тестирования (AB – Analogical Benchmarking) [53, 80]. На основе информации, полученной при профилировании приложений, и информации от компоненты тестирования, а также используемой модели производительности [21], производится оценка стоимости планирования узлов-кандидатов на выполнение приложения, из которых планировщик выбирает те, которые оптимизируют целевые функции.

Модуль запуска и контроля (LM – Launching and Monitoring) формирует окончательное расписание, предоставляет приложениям соответствующие ресурсы, поставляет входные данные и исполняемые файлы, если это необходимо, и выполняет мониторинг исполнения приложений. Модуль запуска и контроля иногда называют «компоновщиком» [35]. Примером модуля запуска и контроля является Globus GRAM (Grid Resource Allocation and Management) [37].

Локальный менеджер ресурсов (LRM – Local Resource Manager) несет основную ответственность за составление локального расписания внутри домена, где присутствуют задания не только от внешних пользователей грид, но и выполняются задания локальных пользователей домена, а также предоставляет отчетную информацию для информационного сервиса грид-среды. Внутри домена могут работать сразу несколько локальных планировщиков каждый со своей локальной политикой управления ресурсами. В качестве примеров подобных локальных планировщиков можно привести Condor [34]

и OpenPBS [68]. Локальный менеджер ресурсов собирает информацию о локальных ресурсах с помощью таких инструментальных средств, как Network Weather Service [93], Hawkeye [34] и Ganglia [73] и формирует отчет с информацией о состоянии ресурсов для информационного сервиса.

1.1.2. Облачные вычисления

Концепция предоставления вычислительных ресурсов, названная облачными вычислениями (cloud computing), сформировалась в 2008 г. В 2011 г. Национальный институт стандартов и технологий США (The National Institute of Standards and Technology, NIST) опубликовал 16-е, окончательное определение данного понятия. Согласно NIST, *облачные вычисления* – это модель обеспечения удобного повсеместного сетевого доступа по требованию к совместно используемому пулу конфигурируемых вычислительных ресурсов, которые можно быстро предоставить и внедрить с минимумом административных усилий или взаимодействия с сервис-провайдером [67].

NIST зафиксированы следующие пять обязательных характеристик облачных вычислений:

– самообслуживание по требованию;
– универсальный доступ по сети;
– объединение ресурсов;
– эластичность;
– учет потребления.

Существует три основных модели обслуживания облачных вычислений:

– программное обеспечение как услуга (SaaS – Software-as-a-Service),
– платформа как услуга (PaaS – Platform-as-a-Service),
– инфраструктура как услуга (IaaS – Infrastructure-as-a-Service),

а также появляются дополнительные модели

– аппаратное обеспечение как услуга (HaaS – Hardware as a Service),
– безопасность как сервис (SECaaS – Security as a Service),
– данные как услуга (DaaS – Data as a Service).

Модель обслуживания SaaS предоставляется возможность использования прикладного программного обеспечения провайдера, работающего в облачной инфраструктуре и доступного из различных клиентских устройств или посредством тонкого клиента, например, из браузера (например, почта Gmail) или интерфейса программы.

Модель обслуживания PaaS предоставляет потребителю возможность использования облачной инфраструктуры для размещения базового программного обеспечения для последующего размещения на нем новых или существующих приложений. В состав платформ входят инструментальные средства создания, тестирования и выполнения прикладного программного обеспечения, предоставляемые облачным провайдером. Примерами подобных платформ являются Google App Engine [76] и Windows Azure [52].

Модель обслуживания IaaS предоставляет возможность использования облачной инфраструктуры для самостоятельного управления ресурсами обработки, хранения, сетей и другими фундаментальными вычислительными ресурсами. Примером облаков, предоставляющих инфраструктуру как услугу, является Nimbus [64].

Эталонная архитектура облачных вычислений NIST содержит пять главных действующих субъектов – *актеров* (см. рис. 2). Каждый актер выступает в некоторой *роли* и выполняет *действия* и *функции*. Эталонная архитектура представляется в виде последовательности диаграмм с увеличивающимся уровнем детализации. Виды актеров облачных вычислений представлены в табл. 1.

Рис. 2. Концептуальная диаграмма эталонной архитектуры облачных вычислений.

Обобщенная облачная среда содержит три концептуальных уровня.

– *Уровень Сервиса (Service Layer)* определяет базовые сервисы, предоставляемые облачным провайдером.
– *Уровень Абстракции и Контроля ресурсов (Resource Abstraction and Control Level)* назначает/предоставляет элементы программного обеспечения, такие как гипервизор, виртуальные хранилища данных и поддерживающие программные компоненты, используемые для реализации облачной инфраструктуры, поверх которой может быть определен/установлен облачный сервис. Кроме того, данный уровень назначает/предоставляет ассоциированные функциональные модули, которые управляют абстрагированными ресурсами для обеспечения эффективного, безопасного и надежного использования.
– *Уровень Физических Ресурсов (Physical Resource Level)* включает все физические ресурсы: компьютерное оборудование и инженерную инфраструктуру.

Табл. 1. Виды узлов логического плана решения задач.

Актер	**Определение**
Облачный потребитель (Cloud Consumer)	Лицо или организация, поддерживающая бизнес-отношения и использующая услуги *Облачных провайдеров.*
Облачный провайдер (Cloud Provider)	Лицо, организация или сущность, отвечающая за доступность облачной услуги для *Облачных потребителей*.
Облачный аудитор (Cloud Auditor)	Участник, который может выполнять независимую оценку облачных услуг, обслуживания информационных систем, производительности и безопасности реализации облака.
Облачный брокер (Cloud Broker)	Сущность, управляющая использованием, производительностью и предоставлением облачных услуг, а также устанавливающая отношения между *Облачными провайдерами* и *Облачными потребителями*.
Облачный оператор связи (Cloud Carrier)	Посредник, предоставляющий услуги подключения и транспорт (услуги связи) для доставки облачных услуг от *Облачных провайдеров* к *Облачным потребителям*.

1.1.3. Платформа UNICORE

Проект UNICORE (Uniform Interface to Computing Resources — единый интерфейс к вычислительным ресурсам) появился в 1997 году, и к настоящему моменту представляет собой комплексное решение, ориентированное на обеспечение прозрачного безопасного доступа к ресурсам распределенной вычислительной среды [103].

Архитектура UNICORE 6 [104] формируется из пользовательского, сервисного и системного слоев (см. рис. 3).

Верхним слоем в архитектуре является *пользовательский слой*. В нем располагаются различные клиенты, обеспечивающие взаимодействие пользователей с распределенной вычислительной средой. На пользовательском уровне доступ организуется с использованием графического интерфейса и

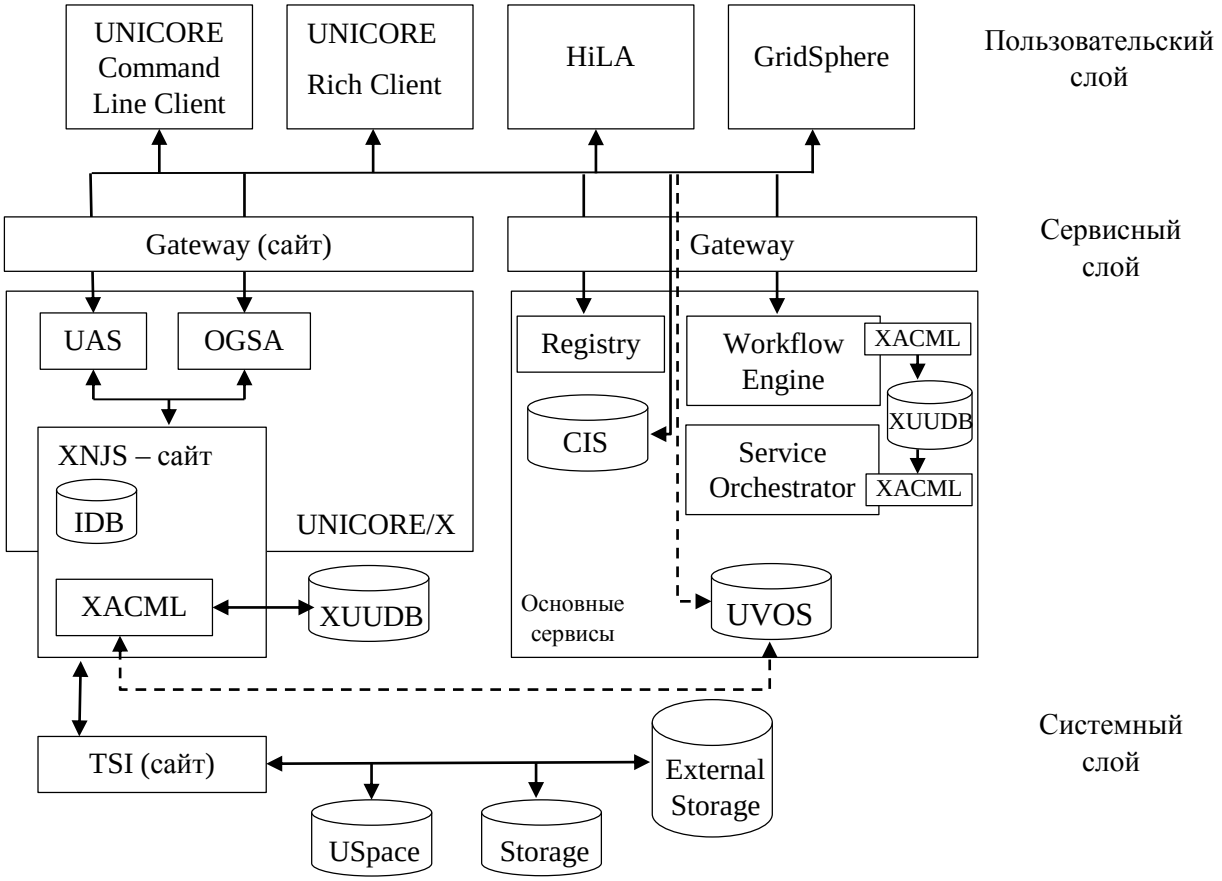

Рис. 3. Архитектура UNICORE 6.

интерфейса командой строки. Задания могут быть запущены на любой из платформ UNICORE в распределенной вычислительной среде. Пользователь может осуществлять мониторинг и управление запущенными заданиями, используя часть интерфейса, называемую *монитором заданий*. Платформа UNICORE позволяет использовать следующие клиенты: Command Line Client (UCC), Eclipse-based UNICORE Rich Client (URC), High Level API for Grid Applications (HiLA) и веб-порталы (например, GridSphere).*Средний слой или слой сервисов* включает все сервисы и компоненты сервис-ориентированной архитектуры UNICORE, основанные на WS-RF 1.2 и SOAP, а также на стандартах WS-I.

Компонент *Gateway* является точкой входа на сайт UNICORE и выполняет аутентификацию всех входящих запросов.

Сервер *UNICORE/X* является главным компонентом сайта UNICORE 6. Посредством WSRF сервер UNICORE/X обеспечивает доступ к ресурсам хранилища, сервису передачи файлов и сервисам выполнения заданиями и мониторинга.

Набор собственных интерфейсов веб-сервисов называется *UNICORE Atomic Services (UAS)*. UAS предоставляет набор базовых услуг для сервисов более высокого уровня, клиентов и пользователей.

Компонент, отвечающий за механизм выполнения заданий в UNICORE, называется *eXtended Network Job Supervisor (XNJS)*. Он обеспечивает ресурсы для хранения данных, передачу файлов и предоставляет сервисы для управления заданиями.

База данных *IDB (Incarnation DataBase)* используется для отображения описания абстрактного задания на языке JSDL (Job submission description language) в описание конкретного задания для определенного ресурса. Вся информация о доступных приложениях и характеристиках ресурсов должна быть определена в базе данных IDB.

Компонент *UNICORE User DataBase (XUUDB)* представляет собой веб-сервис и базу данных для отображения сертификатов X.509 на логины фактических пользователей и их роли. Управление доступом основывается на политике XACML.

Компонент *UNICORE VO Service (UVOS)* используется в качестве альтернативы XUUDB, а также для авторизации пользователей с помощью стандарта SAML.

Компонент *Registry* предназначен для регистрации сервисов, доступных платформе UNICORE. Единая сервисная регистрация необходима для построения распределенной инфраструктуры UNICORE и управления ей.

Common Information Service (CIS) является информационным сервисом UNICORE. CIS собирает статическую и динамическую информацию от всех компонентов XNJS, которые с ним связаны.

Компонент *Workflow Engine* обеспечивает выполнение потока работ. Поток работ создается/запускается с помощью графического интерфейса UNICORE Rich Client или из командной строки Command Line Client.

Компонент *Service Orchestrator* отвечает за выполнение отдельных задач в потоке работ и их мониторинг в распределенной вычислительной среде. В Service Orchestrator реализованы различные стратегии планирования ресурсов. Существует возможность подключения пользовательских стратегий.

В основании архитектуры платформы UNICORE находится *системный слой*. Компонент *Target System Interface (TSI)* обеспечивает взаимодействие между платформой UNICORE и отдельным ресурсом распределенной вычислительной сети. TSI обеспечивает трансляцию команд, поступающих из распределенной вычислительной среды, в команды для локальной системы.

Целевая система (Target System, TS) — это совокупность программного и аппаратного обеспечения, доступного в распределенной вычислительной среде. Описание приложений и аппаратных ресурсов хранится в файле simpleidb директории TSI.

USpace представляет собой директорию для хранения заданий пользователей. Существуют отдельные директории для каждого задания, в которых XNJS и TSI хранят исходные данные, стандартный вывод и стандартный поток ошибок. Для передачи данных между сайтами, например, для передачи данных с/на внешние хранилища, используется протокол GridFTP.

External Storage служит для передачи данных между сайтами. Для работы с внешними хранилищами компонент External Storage использует протокол GridFTP.

UNICORE 6 позволяет использовать *потоки работ* и поддерживает следующие возможности:

– представление задания в виде ориентированного графа;
– использование переменных в потоках работ;

– использование циклов и управляющих конструкций: for-each, if-else, while;
– использование в потоках работ следующих условий: код выхода; существование файла; размер файла.

Основным достоинством использования платформы UNICORE 6 для разработки распределенных вычислительных систем является наличие большого количества различных клиентов, обеспечивающих взаимодействие пользователя с ресурсами распределенной вычислительной сети, а также развитых средств обеспечения безопасности при разработке приложений в распределенных вычислительных средах.

1.2. Обзор алгоритмов планирования

1.2.1. Общая классификация алгоритмов планирования ресурсами

Существующие алгоритмы планирования ресурсов в распределенных вычислительных средах могут быть классифицированы по разным признакам: с точки зрения архитектуры компонентов, участвующих в планировании; используемых политик; целевых функций; моделей приложений; ограничений качества обслуживания (QoS – Quality of Service); стратегий, применяемых для ресурсов с динамическим поведением и др. [40].

На рис. 4 представлена иерархическая классификация алгоритмов планирования общего назначения для параллельных и распределенных вычислительных систем [31].

На верхнем уровне выделяют *статические и динамические алгоритмы* планирования ресурсов. Статическое планирование и расчет стоимостной оценки вычислений осуществляется до начала выполнения задания, когда информация относительно всех ресурсов в распределенных вычислительных средах и всех задачах задания уже доступна [25, 30, 97].

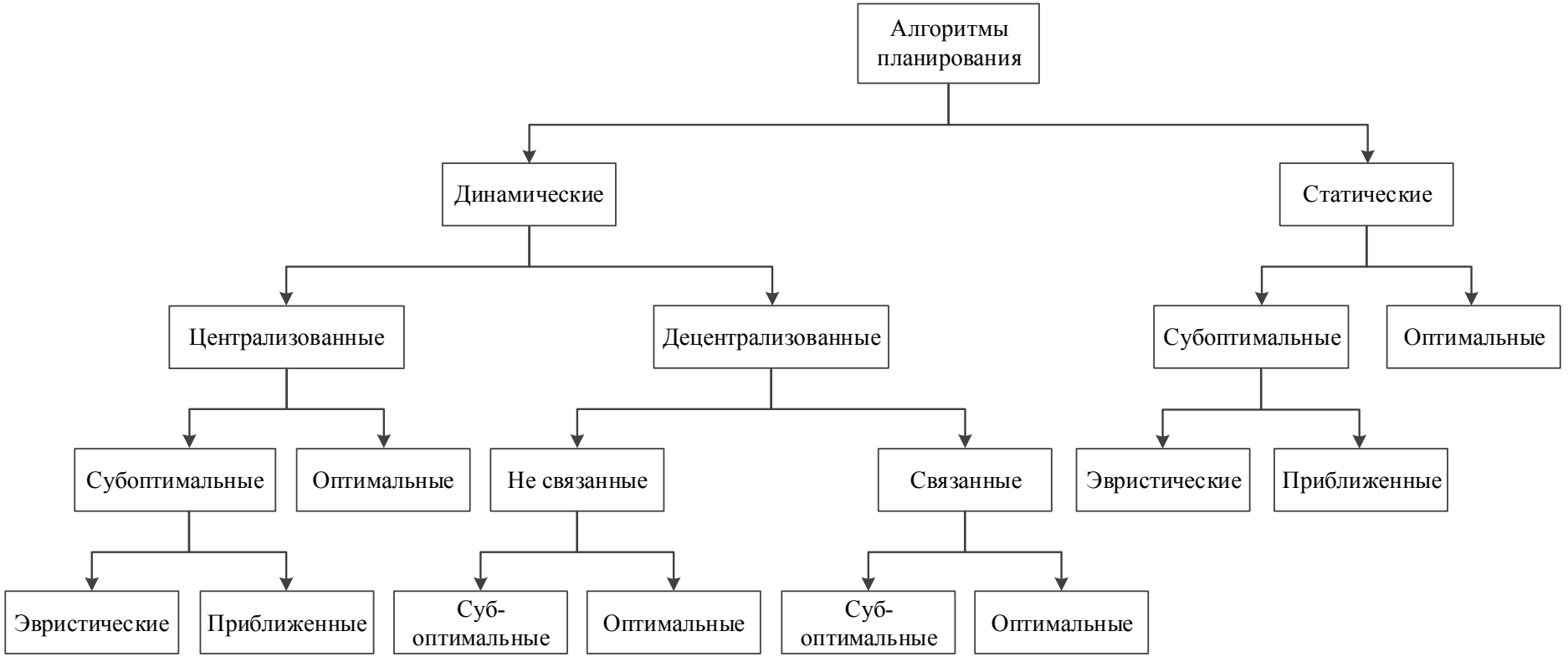

Рис. 4. Иерархическая классификация алгоритмов планирования.

Одно из главных преимуществ статической модели – это простота реализации планировщика. Однако, стоимостная оценка, основанная на статической информации, не адаптивна к ситуациям, когда один из вычислительных узлов выходит из строя, становится изолированным для системы из-за сетевого отказа или из-за высокой загрузки на систему время отклика становится более длительным, чем ожидалось. Для решения проблемы используют вспомогательные механизмы, такие как механизм перепланирования [35]. Наличие данных проблем привело к различиям между статическим и динамическим планированием [46].

Динамическое планирование обычно применяется, когда трудно оценить вычислительную стоимость приложений, поступающих на выполнение динамически в режиме online [57, 85, 33, 66]. Примером использования динамического планирования является управление очередью заданий в системах метавычислений Condor [94] и Legion [32]. Динамическое планирование задач включает в себя два важных компонента: оценка состояния системы и принятие решения о связывании задачи из очереди с выбранным ресурсом [71]. Для сохранения оптимального состояния вычислительной системы используется балансировка загрузки всех ее ресурсов. Преимущество динамической балансировки загрузки над статической состоит в том, что система не обязана знать о поведении приложения во время его выполнения до его запуска. Особенно это подход полезен в системе, где основной целью является максимизация утилизации ресурса, а не минимизация времени выполнения отдельных заданий [51].

Динамические алгоритмы онлайн-планирования, описанные в работах [14] и [65], рассматривают случай резервирования ресурсов, который популярен в распределенных вычислениях. Резервирование ресурсов используется для получения некоторой степени уверенности в производительности

ресурса. Алгоритмы в этих двух работах стремятся минимизировать время исполнения входящих заданий, которые состоят из набора задач.

В динамических сценариях планирования ответственность за принятие глобальных решений планирования ресурсов может лежать на одном централизованном планировщике или нескольких распределенных планировщиках. *Централизованная стратегия* имеет преимущество, заключающееся в простоте реализации, но она плохо масштабируется, не является отказоустойчивой и часто становится узким местом для производительности системы. Например, в работе [72] предлагается централизованный метапланировщик, который использует алгоритм обратного заполнения (Backfill) для планирования параллельных заданий на сложных гетерогенных сайтах. Аналогично, в работе [17] представлен полностью *децентрализованный, динамический и инициируемый отправителем алгоритм* планирования и балансировки загрузки для распределенных вычислительных сред. Главное свойство этого алгоритма заключается в том, что он использует интеллектуальную стратегию поиска узлов-партнеров, на которые могут быть перенесены задачи.

В том случае, когда вся информация относительно состояния ресурсов и заданий известна, *оптимальная привязка* заданий к ресурсам может быть сделана на основании некоторой целевой функции, такой как минимизация времени выполнения заданий и максимальная утилизация ресурсов.

Однако, доказать оптимальность алгоритма или сделать некоторые разумные предположения об оптимальности представляется невозможным из-за того, что общая задача планирования является NP-полной [41]. Текущие исследования пытаются найти *субоптимальные решения*, которые могут быть далее разделены на следующие основные категории. *Приближенные алгоритмы* используют формальные вычислительные модели, вместо поиска всего пространства решений и выбора из него оптимального решения. Дан-

ные алгоритмы осуществляют поиск приемлемого решения, близкого к оптимальному. В случае, когда существует метрика пригодная для оценки решения, этот метод может использоваться для уменьшения времени, потраченного на поиск приемлемого расписания.

Эвристики – представляют собой класс алгоритмов, которые делают наиболее реалистичные предположения об априорном знании относительно характеристик загрузки системы и выполнения. Оценка этого вида решения обычно основана на экспериментах в реальном мире или на моделировании. Наиболее популярными в настоящее время являются экономические подходы [26, 27, 28, 99, 90, 42, 102, 98] и эвристики, основанные на природных явлениях: генетический алгоритм (GA – Genetic Algorithm) [25, 56, 15, 82, 83, 91], моделируемый отжиг (SA – Simulated Annealing) [25, 61, 98], запрещенный поиск (TS – Tabu Search) [25] и комбинированная эвристика [13].

Распределенные алгоритмы планирования можно разделить на *связанные или несвязанные*, в зависимости от того, как работают узлы, использующиеся при планировании заданий, совместно или независимо (несовместно). В несовместном случае локальные планировщики действуют как автономные сущности и принимают решения, с учетом их собственных целевых функций. В совместном случае каждый планировщик в распределенной вычислительной среде несет ответственность за выполнение его собственной части задачи планирования, но при этом все планировщики работают с одной общей целью в масштабе всей системы [79].

1.2.2. Классификация на основе зависимости задач в задании

При рассмотрении отношений между задачами в задании обычно используют следующую дихотомию: есть ли между ними зависимость или же задачи независимы [40]. Зависимость означает, что у задач есть приоритеты,

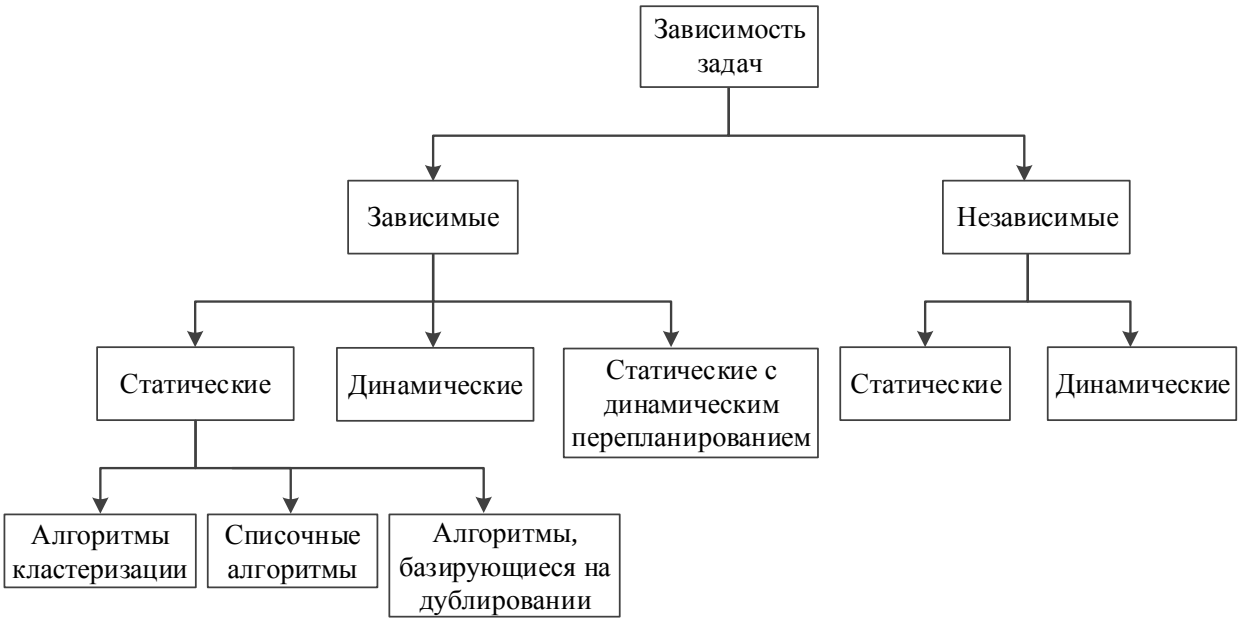

Рис. 5. Классификация зависимых и независимых задач в алгоритмах планирования ресурсами.

то есть, задача не может запуститься до того, как были выполнены все ее предшественники. Зависимость оказывает решающее влияние на разработку алгоритмов планирования. На рис. 5 представлена классификация алгоритмов планирования, основанная на наличии/отсутствии связей между задачами.

Главная *стратегия планирования независимых задач* заключается в назначении независимых задач на ресурсы в зависимости от их загрузки с целью обеспечения высокой пропускной способности вычислительной системы. Примерами статических алгоритмов с оценкой производительности являются: алгоритм минимального времени выполнения (MET — Minimum Execution Time), алгоритм минимального времени завершения (MCT — Minimum Completion Time), эвристики Min-Min [49, 95] и Max-Min [63], Suffrage [63] и XSuffrage [30]. Данные алгоритмы обычно используются для планирования заданий, которые состоят из множества независимых задач с большими модулями и интенсивными вычислениями. Muthuvelu и др. [66] пред-

лагают динамический алгоритм планирования сгруппированных задач, который позволяет уменьшить среднее значение издержек при планировании и запуске задания и увеличить использование ресурса.

В гетерогенных средах на производительность вышеупомянутых алгоритмов также влияет уровень неоднородности задач и ресурсов. Исследование в работе [63] показало, что не существует ни одного алгоритма, который бы имел преимущества во всех аспектах. Для достижения максимальной высокой производительности в вычислительном грид-планировщик должен уметь адаптироваться под неоднородные приложения/ресурсы.

В работах [84] и [81] предложены два алгоритма, основывающиеся на идее дублирования, которое реализуемо в распределенной вычислительной среде, где вычислительных ресурсов много, но не все из них постоянно доступны.

В случае *планирования задач, имеющих зависимости*, задание обычно представляется в виде ориентированного ациклического графа, в котором каждая вершина представляет собой задачу, ориентированное ребро обозначает порядок приоритета между двумя вершинами. В некоторых случаях к вершинам и ребрам могут быть добавлены веса, показывающие вычислительную стоимость и коммуникационную стоимость соответственно.

Важнейшей проблемой при планировании заданий с потоковой структурой является нахождение компромисса между использованием максимального параллелизма задач в задании и минимизации коммуникационных задержек. Для решения данной проблемы (также известной как проблема максимина [41]) в гетерогенных вычислительных системах были предложены три вида эвристических алгоритмов: эвристики списка; алгоритмы, базирующиеся на дублировании; алгоритмы кластеризации (рис. 5).

Планирование списком – это класс эвристик планирования, в котором задачам присваиваются приоритеты, задачи помещаются в список, упорядоченный по мере уменьшения величины приоритета [69, 75]. Решение о выборе задачи из списка для ее выполнения осуществляется на основе приоритета. Сначала осуществляется привязка к ресурсам задач с высоким приоритетом [41]. Классическими примерами эвристик списка являются HEFT (Heterogeneous Earliest-Finish-Time) [87] и FCP (Fast Critical Path) [69].

Критической проблемой в планировании списком для ориентированного ациклического графа является вычисление ранга узла. Собственные значения, используемые для принятия решения об упорядочивании, могут быть усредненными значениями (как в оригинальном алгоритме HEFT в [87]), средним значением [50], худшем значением, оптимальным значением и так далее. Zhao и др. [100] показали, что различные варианты могут влиять на производительность эвристики списка, которые существенны для HEFT (время исполнения может изменяться на 47.2 % для определенного графа). Sakellariou и др. [74] предложили гибридный алгоритм, который менее чувствителен к разным подходам для ранжирования узлов.

Списочный алгоритм планирования, предложенный в работе [97], подобен HEFT алгоритму, но модифицирует его для вычисления уровня узла задачи и учитывает входящую коммуникационную стоимость его родительских задач. В работе [62] Ма и др. предлагают новый списочный алгоритм для распределенных вычислительных сред под названием «расширенный динамический предельный путь» (xDCP – Extended Dynamic Critical Path), который представляет собой модификацию алгоритма DCP для однородной среды.

Алгоритмы, базирующиеся на дублировании, отличаются стратегиями выбора задач для дублирования. Первоначально, алгоритмы этой группы применялись для неограниченного числа идентичных процессоров, таких как

многопроцессорные системы с распределенной памятью. Также они имеют более высокую сложность, чем алгоритмы, обсуждавшиеся выше. Например, Darbha и др. [38] представляют алгоритм под названием «алгоритм планирования, основанный на дублировании задач» (TDS – Task Duplication-based Scheduling Algorithm) для машин с распределенной памятью, имеющий сложность $O(v^2)$ для гомогенных сред.

Для применения дублирования в гетерогенных средах был предложен новый алгоритм под названием «алгоритм планирования, основанный на дублировании задач, для гетерогенных систем» (TANH – Task duplication-based scheduling Algorithm for Network of Heterogeneous systems), который представлен в работах [70] и [18]. Дублирование уже получило некоторое признание [84] и [81], но на сегодняшний день алгоритмы, основанные на дублировании, в распределенных вычислительных средах имеют дело только с независимыми заданиями.

Применение *алгоритмов кластеризации* в параллельных и распределенных системах – это эффективный способ уменьшить коммуникационную задержку в ориентированном ациклическом графе. Основная идея данных алгоритмов заключается в кластеризации взаимосвязанных задач в маркированные группы для дальнейшего их присвоения некоторой группе ресурсов. Примерами алгоритмов кластеризации являются DSC (Dominant Sequence Clustering) [47, 96], CASS-II [60]. Экспериментальные исследования алгоритмов кластеризации приводятся в работах [59, 60].

Рассмотрим более подробно следующие алгоритмы кластеризации: алгоритм KB/L, алгоритм Саркара и алгоритм DSC.

Алгоритм KB/L

В работах [54, 55] рассматривается алгоритм линейной кластеризации Кима и Брауна, также называемый алгоритмом KB/L. Алгоритм KB/L предназначен для кластеризации графа задания и предполагает, что количество вычислительных узлов неограниченно.

Первоначально все дуги графа задания маркируются как «нерассмотренные». На первом шаге выполнения алгоритма KB/L происходит поиск критического пути графа задания, который включает только «нерассмотренные» дуги, с помощью стоимостной функции веса (1). Все вершины найденного критического пути объединяются в один кластер, коммуникационные стоимости дуг обнуляются. На втором шаге дуги, инцидентные вершинам критического пути, маркируются как «рассмотренные». Описанные выше шаги алгоритма KB/L повторятся до тех пор, пока все дуги не будут рассмотрены.

Ким в работе [54] использует стоимостную функцию

$$Cost\ function = w_1 * \sum \tau_i + (1 - w_1)\left(w_2 * \sum c_{ij} + (1 - w_2) * \sum c_{ij}^{adj}\right) \quad (1)$$

для определения длины критического пути графа задания, где w_1 и w_2 – нормализующие множители, $\sum \tau_i$ – сумма вычислительных стоимостей всех вершин критического пути, c_{ij}^{adj} – коммуникационная стоимость дуг между вершиной критического пути и всеми его смежными вершинами, не входящими в критический путь. Ким не приводит систематического способа для определения нормализующих множителей. Положим, $w_1 = \frac{1}{2}$ и $w_2 = \frac{1}{2}$ для стоимостной функции, уменьшающей длину критического пути. Целевой функцией алгоритма KB/L является минимизация параллельного времени графа.

Рассмотрим алгоритм KB/L на примере графа задания на рис. 6а. Результат кластеризации с помощью алгоритма KB/L с параллельным временем 12.5 приведен на рис. 6б.

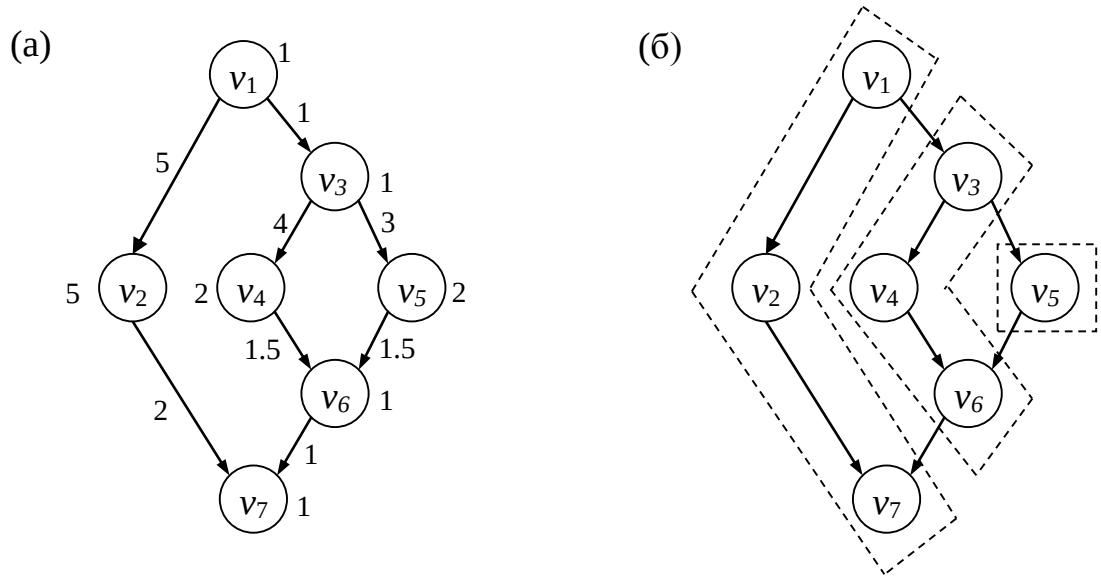

Рис. 6. (а) Граф с весами и разметкой; (б) кластеризация графа с помощью алгоритма KB/L.

На первом шаге критический путь состоит из вершин v_1, v_2 и v_7. Объединяем данные вершины в один кластер и исключаем их из дальнейшего рассмотрения. Оставшийся граф содержит вершины v_3, v_4, v_5, v_6. Новый критический путь состоит из вершин v_3, v_4, v_6. Данные вершины также группируются в один кластер. Окончательно, имеем три кластера $W_0 = \{v_1, v_2, v_7\}$, $W_1 = \{v_3, v_4, v_6\}$ и $W_2 = \{v_5\}$.

Алгоритм KB/L имеет сложность $O\big(v(v+e)\big)$, где v – число вершин графа, e – число дуг графа.

Алгоритм Саркара

В работе [77] рассматривается алгоритм кластеризации Саркара (Sarkar). Суть алгоритма может быть описана следующим образом. Все дуги графа задания сортируются в порядке убывания их коммуникационных стоимостей. Начиная с дуги с большей коммуникационной стоимостью, производится процесс их обнуления. Коммуникационная стоимость дуги обнуляется только в том случае, если параллельное время не увеличится на

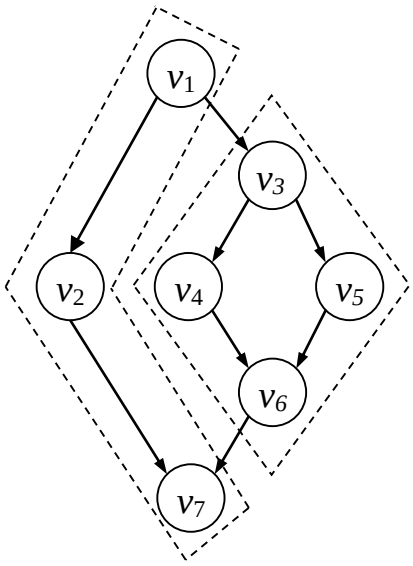

Рис. 7. Нелинейная кластеризация графа с помощью алгоритма Саркара.

следующем шаге. Алгоритм Саркара завершается, когда рассмотрены все дуги графа задания. Целевой функцией алгоритма также является минимизация параллельного времени графа.

Рассмотрим алгоритм Саркара на примере графа задания на рис. 6а. Результат кластеризации с помощью алгоритма Саркара с параллельным временем равным 10 приведен на рис. 7.

На первом шаге алгоритма Саркара выполним сортировку коммуникационных стоимостей дуг графа задания по убыванию. Получим список дуг $\{(v_1, v_2), (v_3, v_4), (v_3, v_5), (v_2, v_7), (v_4, v_6), (v_5, v_6), (v_1, v_3), (v_6, v_7)\}$.

В начальный момент времени каждая вершина графа задания находится в отдельном кластере. Параллельное время равно 14. На втором шаге обнуляем две первые дуги из списка: (v_1, v_2) и (v_3, v_4). При этом параллельное время уменьшится до 12,5. На третьем шаге обнуляем коммуникационную стоимость дуги (v_3, v_5) и т.д. Окончательно, имеем два кластера $W_0 = \{v_1, v_2, v_7\}$ и $W_1 = \{v_3, v_4, v_5, v_6\}$ и параллельное время равное 10.

Алгоритм Саркара имеет сложность $O\big(e(v + e)\big)$, где v – число вершин графа, e – число дуг графа.

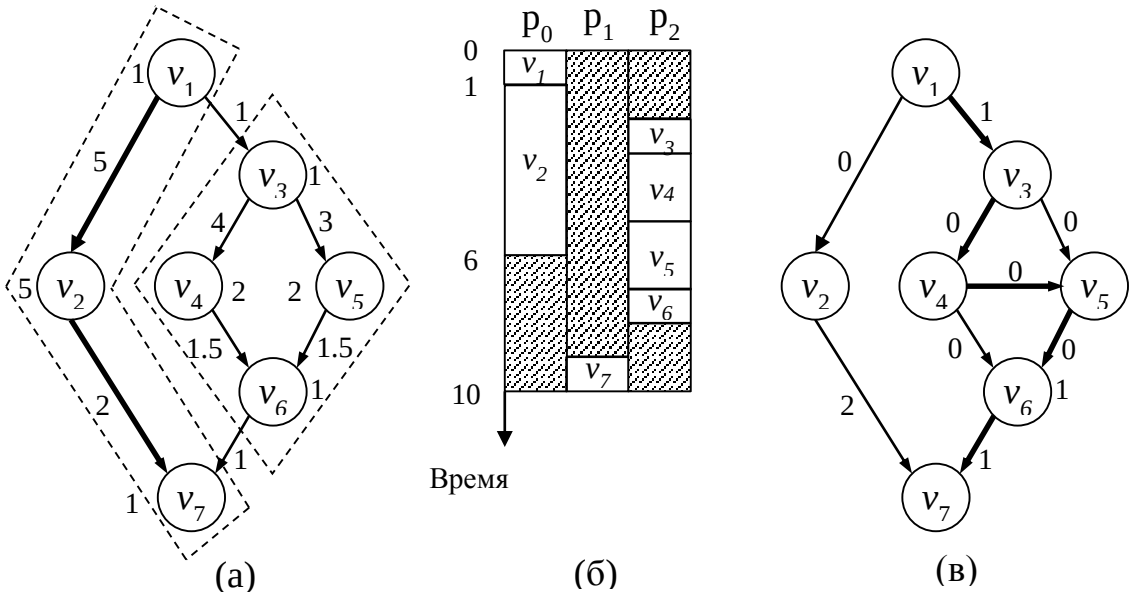

Рис. 8. (а) Кластеризованный граф и его критический путь; (б) диаграмма Ганта; (в) распланированный граф и его доминирующая последовательность.

Алгоритм DSC

Алгоритм кластеризации доминирующей последовательности DSC (Dominant Sequence Clustering) [96] относится к эвристикам кластеризации. Главная идея алгоритма DSC состоит в выполнении последовательности шагов по обнулению коммуникационных стоимостей дуг графа задания с целью сокращения длины доминирующей последовательности, т.е. минимизации параллельного времени. Здесь *доминирующая последовательность* — это критический путь распланированного графа (рис. 8в). Заметим, что обычно под критическим путем графа понимают путь наибольшей длины, включающий вычислительные стоимости его вершин и ненулевые коммуникационные стоимости дуг (рис. 8а).

В начальный момент времени до выполнения алгоритма DSC каждая вершина содержится в отдельном кластере, все дуги графа задания помечаются как «нерассмотренные». После рассмотрения некоторой дуги на предмет возможности ее обнуления, дуга обозначается как «рассмотренная», а

вершина, из которой исходит дуга, помечается как «распланированная». Распланированные вершины образуют множество SN, не распланированные вершины составляют множество USN. Вершина называется «свободной», если все ее предшествующие вершины распланированы.

На первом шаге алгоритма DSC из дуг, принадлежащих доминирующей последовательности графа задания, выбирается первая нерассмотренная дуга. Поиск дуги в доминирующей последовательности производится сверху вниз. На втором шаге дуга обнуляется, а ее вершины объединяются в один кластер, если параллельное время не увеличивается. Порядок выполнения задач в кластере определяется наибольшим значением $bot\_level(n_x, i)$, которое рассчитывается как сумма всех коммуникационных стоимостей дуг и вычислительных стоимостей вершин между вершиной n_x и нижней вершиной графа в доминирующей последовательности. Алгоритм DSC завершается, когда все дуги рассмотрены.

Рассмотрим алгоритм DSC на примере графа задания на рис. 6а. Результат кластеризации с помощью алгоритма DSC с параллельным временем равным 10.5 приведен на рис. 9.

На первом шаге алгоритма DSC определим доминирующую последовательность $DS_1 = \langle n_1, n_2, n_7 \rangle$ и параллельное время $PT_1 = 14$. Рассмотрим первую дугу из доминирующей последовательности – (n_1, n_2). Обнуление коммуникационной стоимости дуги уменьшит параллельное время до 13.5, поэтому произведем объединение вершин n_1, n_2 в один кластер.

Находим следующую доминирующую последовательность $DS_2 = \langle n_1, n_3, n_4, n_6, n_7 \rangle$ и параллельное время $PT_1 = 13.5$. Рассмотрим первую дугу (n_1, n_3) из этой доминирующей последовательности. Ее обнуление уменьшит критический путь до 12.5. Следовательно, обнуляем данную дугу.

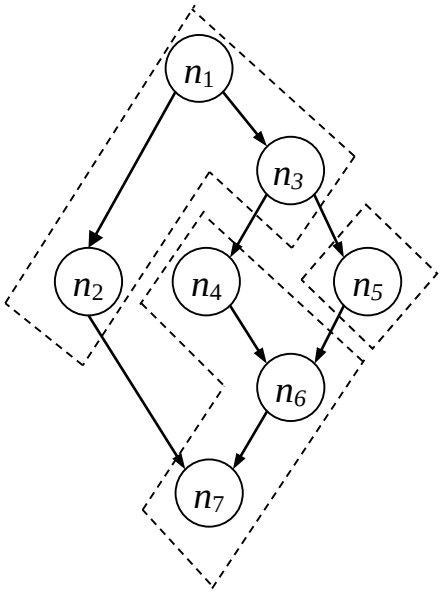

Рис. 9. Нелинейная кластеризация графа с помощью алгоритма DSC

На третьем шаге алгоритма доминирующая последовательность не изменилась $DS_3 = DS_2$. Рассмотрим дугу (n_3, n_4). Ее обнуление приведет к увеличению параллельного времени до 13.5. Следовательно, вершины n_3 и n_4 остаются в разных кластерах.

Аналогично, рассматриваем дуги (n_4, n_6), (n_3, n_5), (n_5, n_6) и (n_6, n_7) соответственно. Окончательно, имеем три кластера $M_0 = \{n_1, n_2, n_3\}$, $M_1 = \{n_4, n_6, n_7\}$, $M_2 = \{n_5\}$ и параллельное время равное 10.5.

Алгоритм DSC в общем случае имеет сложность $O\big((e + v)\log v\big)$, где v – число вершин графа, e – число дуг графа.

1.3. Выводы по главе 1

На сегодняшний день предложено большое количество алгоритмов планирования, ориентированных на использование в распределенных вычислительных средах. Среди них можно выделить алгоритмы, производящие планирование с учетом потоков работ в сложных приложениях. Потоки работ возникают в проблемно-ориентированных средах, в которых каждое вычислительное задание может быть представленного в виде ориентированного графа, узлы которого соответствуют типовым задачам, а дуги представляют

потоки данных между отдельными задачами. Данный подход позволяет добиться более эффективного распределения вычислительных ресурсов между задачами. Однако все известные алгоритмы, производящие планирование с учетом потоков работ, допускают запуск отдельной задачи только на одном ядре. Это существенно ограничивает их применение на современных кластерных вычислительных системах, узлы которых оснащены многоядерными ускорителями, позволяющими выполнять отдельную задачу на множестве процессорных ядер.

В связи с этим, перспективным является направление, связанное с разработкой алгоритмов планирования ресурсами в проблемно-ориентированных средах, ориентированных на современные кластерные вычислительные системы, узлы которых оснащены многоядерными ускорителями.

ГЛАВА 2. ПЛАНИРОВАНИЕ В ПРОБЛЕМНО-ОРИЕНТИРОВАННЫХ СРЕДАХ

В данной главе формулируются определения основных понятий проблемно-ориентированных сред. Приводится пример проблемно-ориентированной среды компьютерного инженерного проектирования в области вычислительной гидро-газодинамики с помощью инженерного пакета ANSYS CFX. Далее строится математическая метамодель проблемно-ориентированной распределенной вычислительной среды и даются примеры ее использования для описания проблемно-ориентированных распределенных вычислительных сред. Центральной частью главы является новый проблемно-ориентированный алгоритм планирования ресурсов *POS* для заданий с потоковой структурой, ориентированный на распределенные вычислительные среды, формируемые на базе вычислительных кластеров с многоядерными ускорителями. В заключение делаются выводы по второй главе.

2.1. Проблемно-ориентированные среды

2.1.1. Основные понятия проблемно-ориентированной среды

Дадим определения основных понятий, необходимых для определения проблемно-ориентированной среды. В этих определениях мы будем использовать понятие потока работ [20].

Задание – это совокупность *задач*, организованных в виде потока работ, и направленных на достижение некоторого полезного результата. Задание определяет порядок запуска задач, условия при которых та или иная задача будет запущена, взаимную синхронизацию задач и информационные потоки между задачами.

Задача представляет собой параметризованную спецификацию некоторого промежуточного результата, получаемого в ходе выполнения зада-

ния. Задача является наименьшей единицей структурирования задания. Параметры задачи могут формироваться из входных данных задания, либо динамически определяться в ходе выполнения потока работ. Результатом задачи являются выходные данные, которые по информационным потокам передаются на вход другим задачам либо формируют результат выполнения всего задания в целом.

Проблемная область – конечное множество заданий, характерных для некоторой предметной области.

Ресурсы – аппаратно-программное обеспечение, необходимое для выполнения задач. Ресурсы располагаются в распределенной вычислительной среде. Основной структурной единицей аппаратных ресурсов является *вычислительный узел*. Вычислительный узел включает в себя определенное количество процессорных ядер, определенный объем оперативной и дисковой памяти. Вычислительные узлы объединяются в целевую систему (см. 1.1.3). Мы исходим из допущения, что все целевые системы гомогенны (однородны), то есть процессорные ядра, модули памяти и диски имеют одинаковые характеристики во всех целевых системах.

Сервис представляет собой спецификацию программных ресурсов, позволяющих решить конкретную задачу. Кроме этого сервис должен определять формат и способ передачи входных и выходных данных задачи. Для одной задачи может существовать несколько различных сервисов, позволяющих ее решить. Сервис привязывается к конкретным целевым системам, расположенным в распределенной вычислительной среде.

Активность – выделение на целевой системе необходимых ресурсов и запуск конкретного сервиса.

Технологический процесс (*Т-процесс*) – совокупность активностей, организованных в виде потока работ, представляющего собой реализацию некоторого задания. Т-процесс получается из задания путем замены задач на соответствующие активности.

Проблемно-ориентированная среда – совокупность ресурсов, сервисов и *программного обеспечения промежуточного слоя* [92], ориентированная на выполнение Т-процессов для некоторой предметной области.

Для описания потоков работ принято использовать графические нотации. Наиболее употребительными примерами таких нотаций являются BPMN [24], блок-схема [4], функциональная блок-схема [7], событийная цепочка процессов [88], диаграммы деятельности UML [3]. В диссертационной работе для описания потока работ мы будем использовать нотацию диаграмм деятельности UML.

2.1.2. Пример проблемно-ориентированной среды

В качестве примера рассмотрим структуру проблемно-ориентированной среды компьютерного инженерного проектирования в области вычислительной гидро-газодинамики [16] с помощью инженерного пакета ANSYS CFX 13.0 [5]. Назовем эту среду *CFD* (*Computational Fluid Dynamics*) *средой*. В качестве примера задания возьмем расчет течения в статическом миксере [89]. Данное задание состоит из следующих задач.

1. Создание геометрической модели.
2. Построение расчетной сетки.
3. Создание файла описания задания.
4. Расчет в CFX.
5. Визуализация результатов в постпроцессоре CFX.
6. Если критерии оптимизации неудовлетворительны:
    6.1. Корректировка геометрической модели для препроцессора CFX.
7. Если результат неудовлетворительный:
    7.1. Уточнение сетки.
    7.2. Корректировка файла описания задания.
    7.3. Повторный расчет в CFX.
    7.4. Визуализация результатов в постпроцессоре CFX.

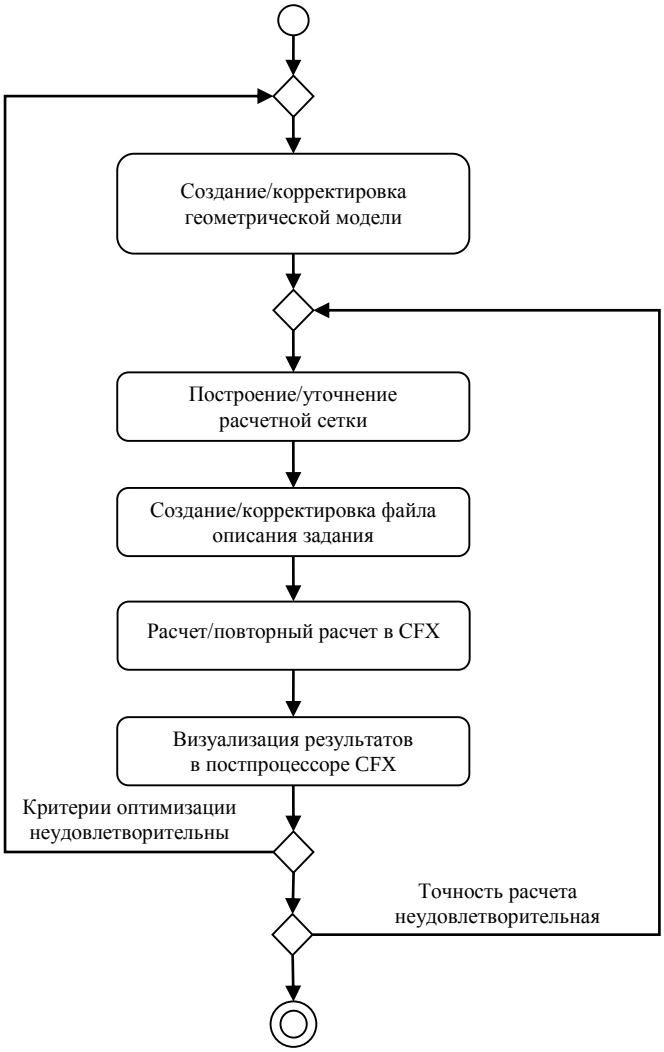

Рис. 10. Структура задания «Расчет течения в статическом миксере».

Указанные задачи организуются в поток работ так, как это показано на рис. 10.

Задача *Создание геометрической модели* предполагает построение новой геометрической модели статического миксера. Использование входных данных данная задача не предполагает. Задача *Корректировка геометрической модели* заключается в корректировке уже существующей модели. Входными данными является файл геометрической модели в формате *.agdb [11]. Выходными данными в обоих случаях является файл в формате *.agdb.

Задача *Построение расчетной сетки* заключается в задании критериев создания тетрагональной сетки, генерации раздельных двух- и трехмерных

регионов, генерации расчетной сетки на основе существующей геометрической модели. Входными данными является файл геометрической модели в формате *.agdb. В задаче *Уточнение расчетной сетки* генерация сетки производится на основе существующей геометрической модели, но с другими параметрами, или перестраивается некоторая область уже существующей сетки. Входными данными является файл расчетной сетки в формате *.cmdb [11, 8]. Выходными данными в обоих случаях является файл расчетной сетки в формате *.cmdb.

Задача *Создание файла описания задания* включает в себя следующие этапы.

1. Импорт расчетной сетки.Определение физической модели:Граничные и начальные условия;

   2.2.Интерфейсы областей;

   2.3.Материалы;

   2.4.Переменные, выражения и другие параметры [8].

3. Определение параметров решателя.

Задача *Корректировка файла описания задания* подразумевает только импорт новой расчетной сетки. В обоих случаях входной файл для данных задач имеет формат *.cmdb, выходной файл - *.def [8, 10].

Задача *Расчет в CFX* заключается в запуске решателя, в качестве параметров которого указываются:

- файл описания задания (definition file),
- режим запуска решателя:
  - последовательный режим выполнения расчета,
  - параллельный режим выполнения расчета на локальном компьютере,
  - параллельный режим выполнения расчета в распределенной среде,
- настройка связи решателя CFX с внешними решателями (например, ANSYS MultiField),

– точность расчета и другие [10].

Входными данными является файл описания задания в формате *.def. Задача *Повторный расчет* в CFX подразумевает перезапуск решателя с автоматической интерполяцией результатов. Интерполяция результатов позволяет использовать уже имеющиеся результаты вычислений как начальные приближения. Входными данными являются файл описания задания для решателя в формате *.def и файл результатов в формате *.res [10]. По окончании расчета в обоих случаях формируется выходной файл результатов, имеющий расширение *.res. Расчет в CFX может производиться с помощью командной строки

```
<CFXROOT>\cfx5solve –def <filename.def> -par-dist ‘hostA, hostB’,
```

где в качестве аргумента опции –def указывается файл описания задания, а опция –par-dist показывает, что расчет выполняется в распределенной среде в параллельном режиме на компьютерах с идентификаторами hostA и hostB.

Задача *Визуализация результатов в постпроцессоре CFX* состоит в запуске постпроцессора и включает в себя импорт результатов решения, построение графических объектов (точка, линия, плоскость, изоповерхность, объем и др.), отображение модели в окне просмотра, задание положения камеры, создание изображений, графиков и отчетов. Все вышеперечисленные действия сохраняются в файле сессии, который используется постпроцессором CFX для их автоматического исполнения. Входными файлами для данной задачи являются файл результатов решения в формате *.res и файл сессии в формате *.cse [9], выходными данными является графический файл - *.png. Визуализация результатов может производиться с помощью командной строки в пакетном режиме

```
<CFXROOT>\bin\cfdpost -batch <filename.cse> <filename.res>
```

с указанием таких аргументов, как файл сессии *.cse и файл результатов решения в формате *.res.

После вычисления *критерия оптимизации* определяется необходимость повторного расчета. В данном задании критерием оптимизации является уменьшение уровня температурной неравномерности потока на выходе. Результат расчета критерия оптимизации представляет собой логическое значение. Если значение ложно, то происходит переход на первую задачу в потоке работ, если значение истинно – переход к следующей задаче, согласно потоку работ на рис. 10.

При выполнении анализа результатов рассматривается *точность расчетов*, которая может быть получена путем сопоставления результатов вычислений с уже имеющимися результатами точного решения. В реальных задачах точное решение заранее неизвестно, поэтому практически единственным методом оценки точности является проведение последовательности решений, которая будет сходиться к точному решению. Методом оценки точности является построение зависимостей результатов решения от разбиения расчетной сетки и их экстраполяция с учетом опыта пользователя. Если точность проведенных расчетов неудовлетворительная, то следующей выполняется задача уточнения сетки и т.д. Если точность расчетов устраивает пользователя, выполнение задания завершается.

В общем случае при реализации задания «Расчет течения в статическом миксере» возможны альтернативные варианты с использованием других форматов для передачи данных между задачами. Для задачи создания/корректировки геометрической модели для препроцессора CFX сохранить выходные данные можно файле формата *.x_t [106], созданном с помощью приложения SolidWorks [105].

```
<!— ANSYS CFX-Solver -->
<idb:IDBApplication>
  <idb:ApplicationName>CFX-Solver</idb:ApplicationName>
  <idb:ApplicationVersion>1.0</idb:ApplicationVersion>
  <jsdl:POSIXApplication xmlns:jsdl="http://sche-
mas.ggf.org/jsdl/2005/11/jsdl-posix">
      <jsdl:Executable>c:\ANSYS\CFX\bin\cfx5solve.exe</jsdl:Executable>
      <jsdl:Argument>-def$SOURCE?</jsdl:Argument>
    </jsdl:POSIXApplication>
  </idb:IDBApplication>
```

Рис. 11. Описание пакета ANSYS CFX-Solver в целевой системе UNICORE.

Задача построения/уточнения расчетной сетки допускает сохранение выходных данных в следующих форматах *.gtm, *.cfx, *.cmdb, *.dsdb, *.cfx5, *.msh, *.cdb, *.inp, *.cas, *.neu и другие. Полный перечень форматов файлов, допускающих импорт расчетной сетки в ANSYS CFX-Pre, описаны в документации ANSYS CFX-Pre User's Guide 13.0 [8].

Задача визуализации результатов в постпроцессоре CFX позволяет сохранять графические файлы в форматах *.png, *.jpeg, сохранять файлы анимации - в форматах *.avi, *.mpeg4, *.can, файлы отчетов сохраняются в форматах *.html или *.txt. Полный перечень возможностей и форматов файлов построцессора ANSYS CFD-Post, описаны в документации ANSYS CFD-Post User's Guide 13.0 [9].

В качестве *ресурсов* для решения приведенного выше задания используется аппаратное обеспечение (вычислительные узлы) и программное обеспечение: Design Modeler [11], CFX-Mesh [12], ANSYS CFX-Pre, ANSYS CFX-Solver [10], ANSYS CFD-Post и лицензия ANSYS Academic Research.

Для организации и запуска сервисов CFD-среды можно использовать платформу UNICORE, описанную в разделе 1.1.3.

Дадим пример описания в целевой системе средствами UNICORE сервиса CFX-Solver, решающего задачу «Расчет в CFX» с использованием

```
{
  ApplicationName: CFX-Solver,
  ApplicationVersion: 1.0,
  Environment: ["SOURCE=input.def",
      ],
      Imports: [
              {File: "input.def", To: "input.def"},
      ],
      Exports: [
              {File: "output.res", To: "output.res"},
      ]
}
Resources: {
      #memory per CPU (bytes)
      Memory: 268435456,
      #time per CPU (seconds)
      Runtime: 86400,
      #number of nodes
      Nodes: 8,
      #CPUs per node
      CPUs: 2
}
```

Рис. 12. Файл спецификаций задачи «Расчет в CFX».

инженерного пакета ANSYS CFX-Solver. Описание пакета ANSYS CFX-Solver на рис. 11 содержит название приложение, его версию, путь к исполняемому файлу и аргументы.

Активность, реализующая запуск сервиса CFX-Solver, описывается в среде UNICORE в виде файла спецификаций задачи, хранящегося на стороне клиента UNICORE (рис. 12).

2.2. Формальные методы представления проблемно-ориентированных распределенных вычислительных сред

В данном разделе строится формальная метамодель проблемно-ориентированной среды.

2.2.1. Базовые определения

Ориентированным графом называется четверка $G = \langle V, E, init, fin \rangle$, где V – множество вершин; E – множество дуг; $init: E \to V$ – функция, определяющая *начальную вершину* дуги; $fin: E \to V$ – функция, определяющая *конечную вершину* дуги.

Вершина $v_1, v_2 \in V$ называются *смежными*, если

$$\exists e \in E \left(\left(v_1 = init(e) \,\&\, v_2 = fin(e) \right) \vee \left(v_1 = fin(e) \,\&\, v_2 = init(e) \right) \right), \quad (2)$$

другими словами, существует дуга e, соединяющая эти вершины. Если $v_1 = init(e) \,\&\, v_2 = fin(e)$, мы будем обозначать это следующим образом: $(v_1, v_2) = e \in E$ и говорить, что вершины v_1, v_2 *инцидентны* дуге e.

Пусть $v_1, v_2 \in V$ и $n \geq 1$. Упорядоченная последовательность дуг

$$(e_1, e_2, \dots, e_n) \in E^n$$

называется *путем длины n*, от вершины v_1 к вершине v_2, если $v_1 = init(e_1)$, $v_2 = fin(e_n)$ и $fin(e_i) = init(e_{i+1})$ для всех $i \in \{1, \dots, n-1\}$. Если $(e_1, e_2, \dots, e_n) \in E^n$ – путь от вершины v_1 к вершине v_2, то *обратным путем* от вершины v_2 к вершине v_1 называется упорядоченная последовательность дуг $(e_n, e_{n-1}, \dots, e_1) \in E^n$. Путь $(e_1, e_2, \dots, e_n)$ называется *простым*, если $init(e_1), init(e_2), \dots, init(e_n)$ все различны между собой и если $fin(e_1), fin(e_2), \dots, fin(e_n)$ тоже все различны. *Циклом* называется простой путь от некоторой вершины к ней самой. Ориентированный граф называется *ациклическим*, если он не содержит циклов [6].

Вершины $v_1, v_2 \in V$ называются *независимыми*, если не существует прямого или обратного пути от вершины v_1 к вершине v_2. В противном случае, вершины v_1, v_2 *зависимы*.

Пусть задан ориентированный граф $G = \langle V, E, init, fin \rangle$. *Взвешиванием графа G* будем называть функцию $\delta: E \to \mathbb{Z}_{\geq 0}$. *Разметкой графа G* будем называть функцию

$$\gamma: V \to \mathbb{N}^2. \quad (3)$$

2.2.2. Модель вычислительной среды

Графом задания называется размеченный взвешенный ориентированный ациклический граф $G = \langle V, E, init, fin, \delta, \gamma \rangle$, где V – множество вершин, соответствующих задачам, E – множество дуг, соответствующих потокам данных.

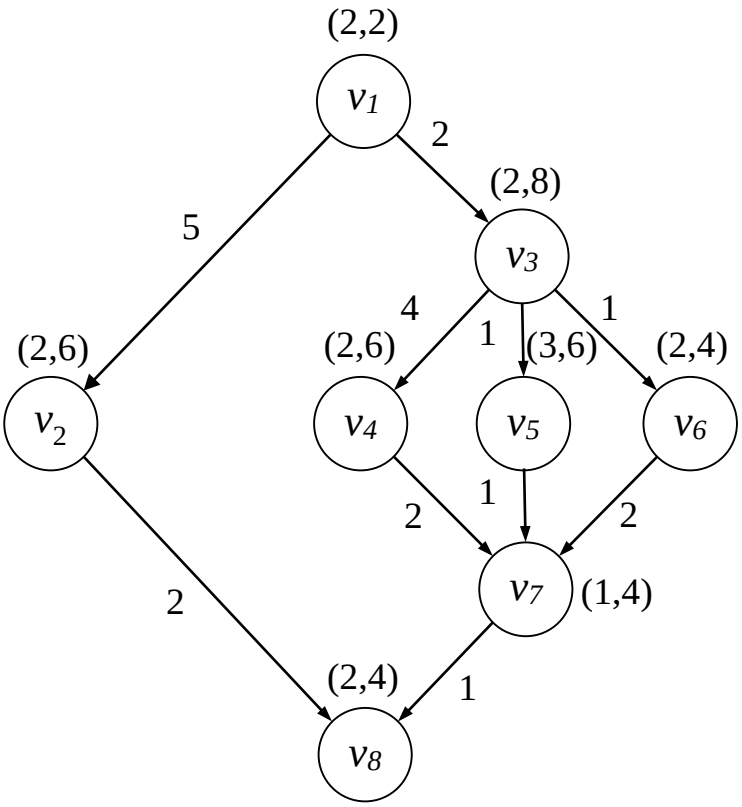

Рис. 13. Граф задания.

Вес $\delta(e)$ дуги e определяет объем данных, который необходимо передать по дуге e от задачи, ассоциированной с вершиной $init(e)$ к задаче, ассоциированной с вершиной $fin(e)$. Метка

$$\gamma(v) = (m_v, t_v) \tag{4}$$

определяет максимальное количество процессорных ядер m_v, на которых задача v имеет ускорение, близкое к *линейному*, и время t_v выполнения задачи v на одном ядре. Данная модель предполагает, что *вычислительная стоимость* $\chi(v, j_v)$ задачи v на j_v процессорных ядрах определяется следующей формулой:

$$\chi(v, j_v) = \begin{cases} t_v/j_v\,, & \text{если } 1 \le j_v \le m_v; \\ t_v/m_v\,, & \text{если } m_v < j_v. \end{cases} \tag{5}$$

Другими словами, при наращивании количества процессорных ядер в диапазоне от 1 до m_v, мы будем получать прямо пропорциональное уменьшение

времени счета; при увеличении количества ядер в интервале от m_v до $+\infty$ ускорение будет отсутствовать.

На рис. 13 приведен пример графа задания, содержащего 8 вершин. Для каждой из вершин указана метка в виде пары (m_v, t_v). Каждой дуге графа сопоставляется вес – объем данных $\delta(e)$, передаваемых по данной дуге.

Вычислительным узлом P называется упорядоченное множество процессорных ядер $\{c_0, \dots, c_{d-1}\}$.

Вычислительной системой называется упорядоченное множество вычислительных узлов $\mathfrak{P} = \{P_0, \dots, P_{k-1}\}$. В реальности вычислительная система может являться распределенной вычислительной системой, объединяющей несколько вычислительных кластеров, каждый из которых является отдельным узлом этой системы.

Кластеризацией называется однозначное отображение $\omega: V \to \mathfrak{P}$ множества вершин V графа задания G на множество вычислительных узлов $\mathfrak{P}$.

Пусть задана вычислительная система $\mathfrak{P} = \{P_0, \dots, P_{k-1}\}$, состоящая из k узлов. *Кластер* W_i – это подмножество всех вершин, отображаемых на вычислительный узел $P_i \in \mathfrak{P}$:

$$W_i = \{v \in V \mid \omega(v) = P_i \in \mathfrak{P}\}. \quad (6)$$

Имеем

$$W_i \cap W_j = \emptyset \text{ для } i \neq j; \quad (7)$$

$$V = \bigcup_{i=0}^{k-1} W_i. \quad (8)$$

Пусть задан граф задания $G = \langle V, E, init, fin, \delta, \gamma \rangle$, для которого определена функция кластеризации $\omega(v)$. Будем называть такой граф *кластеризованным* и обозначать как $G = \langle V, E, init, fin, \delta, \gamma, \omega \rangle$.

В рамках модели мы предполагаем, что время передачи любого объема данных между узлами, принадлежащими одному кластеру, близко к нулю, а время передачи данных между узлами, принадлежащими разным кластерам,

пропорционально объему передаваемых данных с коэффициентом 1. В соответствии с эти мы можем определить функцию $\sigma: E \to \mathbb{Z}_{\geq 0}$, вычисляющую *коммуникационную стоимость* (время) передачи данных по дуге $e \in E$, следующим образом:

$$\sigma(e) = \begin{cases} 0, & \text{если } \omega\big(init(e)\big) = \omega\big(fin(e)\big); \\ \delta(e), & \text{если } \omega\big(init(e)\big) \neq \omega\big(fin(e)\big). \end{cases} \tag{9}$$

Пусть задан кластеризованный граф $G = \langle V, E, init, fin, \delta, \gamma, \omega \rangle$. *Расписанием* называется отображение $\xi: V \to \mathbb{Z}_{\geq 0} \times \mathbb{N}$, которое произвольной вершине $v \in V$ сопоставляет двойку чисел

$$\xi(v) = (\tau_v, j_v), \tag{10}$$

где τ_v определяет время запуска задачи v, j_v – количество процессорных ядер, выделяемых задаче v. Обозначим через s_v – время останова задачи v. Имеем

$$s_v = \tau_v + \chi(v, j_v)\,, \tag{11}$$

где χ – функция временной сложности, определенная с помощью формулы (5). Расписание называется *корректным*, если оно удовлетворяет следующим условиям:

$$\forall\, e \in E\ \left(\tau_{fin(e)} \geq \tau_{init(e)} + \chi\big(init(e), j_{init(e)}\big) + \sigma(e)\right); \tag{12}$$

$$\forall\, v \in V\ (j_v \leq m_v)\,; \tag{13}$$

$$\forall t \in \mathbb{N}\left(\forall i \in [0,\ldots,k-1]\left(\sum_{\substack{v \in W_i \wedge \\ \tau_v < t \leq s_v}} j_v \leq |P_i|\right)\right). \tag{14}$$

Условие (12) означает, что для любых двух смежных вершин $v_1 = init(e)$ и $v_2 = fin(e)$ время запуска v_2 не может быть меньше суммы следующих величин: время запуска v_1, время выполнения v_1, коммуникационная стоимость дуги e. Условие (13) означает, что количество ядер, выделяемое за-

даче v_1, не превышает границы линейной масштабируемости, заданной разметкой γ в контексте формулы (4). Условие (14) означает, что в любой момент времени t общее количество процессорных ядер, выделяемых задачам на узле с номером i, не может превосходить количества ядер на этом узле. В дальнейшем мы будем рассматривать только корректные расписания, если явно не оговорено противное.

Кластеризованный граф $G = \langle V, E, init, fin, \delta, \gamma, \omega \rangle$ с заданным расписанием ξ будем называть *распланированным* и обозначать как $G = \langle V, E, init, fin, \delta, \gamma, \omega, \xi \rangle$.

Ярусно-параллельной формой (*ЯПФ*) называется разбиение множества вершин V ориентированного ациклического графа $G = \langle V, E, init, fin \rangle$ на перенумерованные подмножества (*ярусы*) L_i $(i = 1, \ldots, r)$, удовлетворяющие следующим свойствам:

$$\left.\begin{array}{l} V = \bigcup_{i=1}^{r} L_i; \\ \forall i \neq j \in \{1,\ldots,r\}\left(L_i \cap L_j = \varnothing\right); \\ \forall (v_1, v_2) \in E\left(\forall i \neq j \in \{1,\ldots.r\}\left(v_1 \in L_i \wedge v_2 \in L_j \Rightarrow i < j\right)\right). \end{array}\right\} \quad (15)$$

Последнее условие означает, что если из вершины v_1 идет дуга в вершину v_2, то вершина v_2 должна располагаться на ярусе с большим номером по отношению к ярусу, на котором располагается вершина v_1. количество вершин в ярусе L_i называется его *шириной*. Количество ярусов в ЯПФ называется ее *высотой*, а максимальная ширина ее ярусов – *шириной ЯПФ*. ЯПФ называется *канонической* [2], если все *входные* (не имеющие входных дуг) вершины принадлежат ярусу с номером 1, и максимальная длина пути, оканчивающегося в вершине, принадлежащей ярусу k, равна $k-1$.

Пусть в распланированном графе $G = \langle V, E, init, fin, \delta, \gamma, \omega, \xi \rangle$ задан путь $y = (e_1, e_2, \ldots, e_n)$. Так как граф G ациклический, то любой путь в нем, в том числе и y, будет простым. *Стоимостью* пути $u(y)$ называется величина

$$u(y) = \chi\left(fin(e_n), j_{fin(e_n)}\right) + \sum_{i=1}^{n}\left(\chi\left(init(e_i), j_{init(e_i)}\right) + \max\left(\sigma(e_i), \tau_{fin(e_i)} - s_{init(e_i)}\right)\right), \quad (16)$$

где χ – функция, определяемая формулой (5), значением которой является вычислительная стоимость вершины; σ – функция, определяемая формулой (9), значением которой является коммуникационная стоимость дуги; j_v и τ_v определяется по формуле (10); s_v определяется по формуле (11).

Пусть Y – множество всех путей в распланированном графе $G = \langle V, E, init, fin, \delta, \gamma, \omega, \xi \rangle$. Путь $\bar{y} \in$ Y называется *критическим*, если

$$u(\bar{y}) = \max_{y \in Y} u(y), \quad (17)$$

то есть, критический путь обладает максимальной стоимостью.

Утверждение 1. Любой критический путь в распланированном графе $G = \langle V, E, init, fin, \delta, \gamma, \omega, \xi \rangle$ начинается с входной вершины и заканчивается в *выходной* (не имеющей выходных дуг) вершине.

Доказательство проведем методом от противного. Предположим, что существует критический путь $\bar{y} = (e_1, \ldots, e_n) \in Y$, начинающийся не с входной вершины. Тогда существует дуга e_h такая, что $fin(e_h) = init(e_1)$. Так как G ациклический, то отсюда следует, что $\tilde{y} = (e_h, e_1, \ldots, e_n) \in Y$. В силу (3) – (5) и (16) получаем $u(\bar{y}) < u(\tilde{y})$. Это противоречит (17). Аналогичным образом приходим к противоречию при предположении, что существует критический путь, заканчивающийся не в выходной вершине. Утверждение доказано.

2.2.3. Пример построения моделей

Рассмотрим пример вычислительной системы $\mathfrak{P} = \{P_0, P_1\}$, состоящей из двух вычислительных узлов. Вычислительный узел P_0 включает в себя четыре процессорных ядра: $P_0 = \{c_{00}, c_{01}, c_{02}, c_{03}\}$.

Табл. 2. Функция разметки γ

Вершина v	$\gamma(v) = (m_v, t_v)$	
	m_v	t_v
v_1	2	2
v_2	2	6
v_3	2	8
v_4	2	6
v_5	3	6
v_6	2	4
v_7	1	4
v_8	2	4

Табл. 3. Функция кластеризации ω

Вершина v	$\omega(v)$
v_1	P_0
v_2	P_0
v_3	P_1
v_4	P_1
v_5	P_1
v_6	P_1
v_7	P_1
v_8	P_0

Табл. 4. Весовая функция δ

Дуга e	$\delta(e)$
(v_1, v_2)	5
(v_1, v_3)	2
(v_2, v_8)	2
(v_3, v_4)	4
(v_3, v_5)	1
(v_3, v_6)	1
(v_4, v_7)	2
(v_5, v_7)	1
(v_6, v_7)	2
(v_7, v_8)	1

Табл. 5. Коммуникационная стоимость σ

Дуга e	**Коммуникационная стоимость $\sigma(e)$**
(v_1, v_2)	0
(v_1, v_3)	2
(v_2, v_8)	0
(v_3, v_4)	0
(v_3, v_5)	0
(v_3, v_6)	0
(v_4, v_7)	0
(v_5, v_7)	0
(v_6, v_7)	0
(v_7, v_8)	1

Вычислительный узел P_1 включает в себя шесть процессорных ядер: $P_1 = \{c_{10}, c_{11}, c_{12}, c_{13}, c_{14}, c_{15}\}$. Рассмотрим граф задания G, приведенный на рис. 13. Определим функцию разметки $\gamma(v) = (m_v, t_v)$ для графа G с помощью табл. 2. В соответствии с (4) величина m_v определяет верхнюю границу линейной масштабируемости, а величина t_v – время выполнения на одном процессорном ядре задачи, ассоциированной с вершиной v.Определим для графа G с помощью табл. 4 весовую функцию δ, задающую объем данных, передаваемых по каждой дуге.

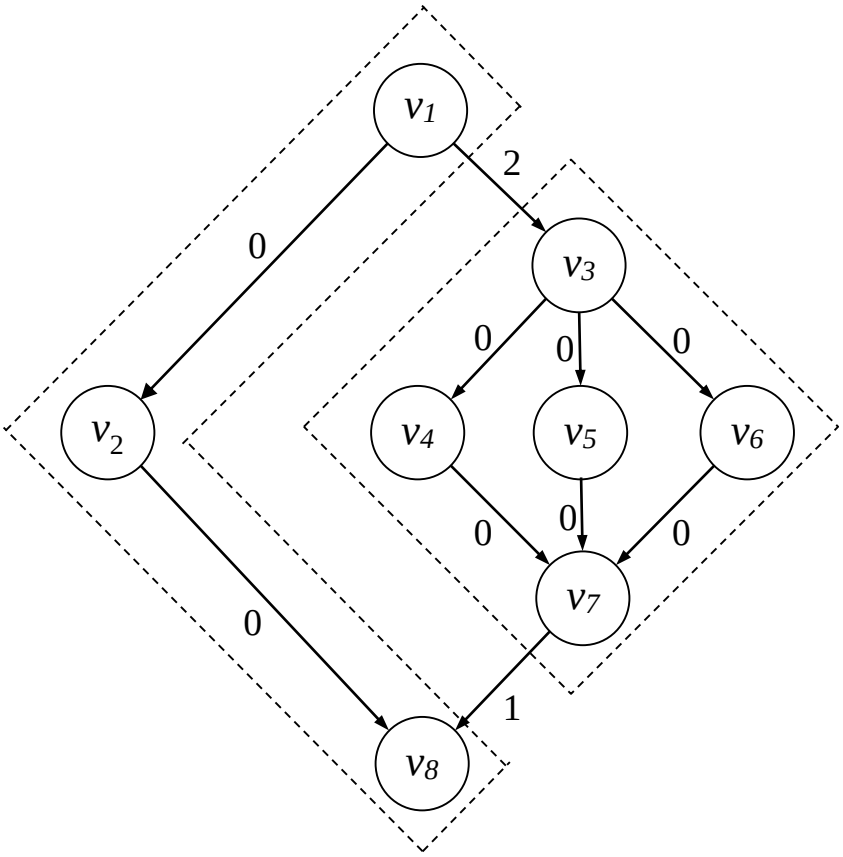

Рис. 14. Кластеризованный граф с указанием коммуникационной стоимости для дуг.

Зададим для графа G и вычислительной системы $\mathfrak{P}$ функцию кластеризации ω с помощью табл. 3. Вершины v_1, v_2, v_8 отображаются на вычислительный узел P_0, а вершины v_3, v_4, v_5, v_6, v_7 отображаются на вычислительный узел P_1. В соответствии с этим, множество вершин V графа задания G разбивается функцией кластеризации $\omega(v)$ на два кластера: $W_0 = \{v_1, v_2, v_8\}$ и $W_1 = \{v_3, v_4, v_5, v_6, v_7\}$.

С помощью формулы (9) вычислим коммуникационные стоимости для дуг графа G, кластеризованного с использованием функции ω, заданной табл. 3. Полученные значения приведены в табл. 5.

Кластеризованный таким образом граф $G = \langle V, E, init, fin, \delta, \gamma, \omega \rangle$ представлен на рис. 14, где для каждой дуги указана получившаяся коммуникационная стоимость. Мы видим, что коммуникационная стоимость

Табл. 6. Расписание

Вершина v	$\xi(v) = (\tau_v, j_v)$	
	Время запуска τ_v	**Количество ядер j_v**
v_1	0	2
v_2	1	2
v_3	3	2
v_4	7	2
v_5	7	2
v_6	7	2
v_7	10	1
v_8	15	2

Табл. 7. Вычислительная стоимость

Вершина v	$\chi(v, j_v)$
v_1	1
v_2	3
v_3	4
v_4	3
v_5	3
v_6	2
v_7	4
v_8	2

передачи данных внутри кластеров равна нулю. Ненулевая стоимость передачи данных сохранилась для дуг, соединяющих вершины из разных кластеров: (v_1, v_3) и (v_7, v_8).

С помощью табл. 6 зададим для графа G, изображенного на рис. 14, функцию расписания ξ. Покажем, что данное расписание является корректным. Для этого необходимо и достаточно проверить выполнение условий (12)-(14). Проверим условие (12) означающее, что для любых двух смежных вершин $v_1 = init(e)$ и $v_2 = fin(e)$ время запуска v_2 не может быть меньше суммы следующих величин: время запуска v_1, вычислительная стоимость v_1, коммуникационная стоимость дуги e. По формуле (5) рассчитаем вычислительную стоимость χ для каждой вершины графа G (см. табл. 7).

С помощью табл. 6 зададим для графа G, изображенного на рис. 14, функцию расписания ξ. Покажем, что данное расписание является корректным. Для этого необходимо и достаточно проверить выполнение условий (12)-(14). Проверим условие (12) означающее, что для любых двух смежных вершин $v_1 = init(e)$ и $v_2 = fin(e)$ время запуска v_2 не может быть меньше суммы следующих величин: время запуска v_1, вычислительная стоимость v_1, коммуникационная стоимость дуги e. По формуле (5) рассчитаем вычислительную стоимость χ для каждой вершины графа G (см. табл. 7).

Табл. 8. Данные для проверки условия (12)

Дуга $e = (v, v')$	$\tau_{v'}$	$\tau_v + \chi(v, j_v) + \sigma(e)$
(v_1, v_2)	1	1
(v_1, v_3)	3	3
(v_2, v_8)	14	4
(v_3, v_4)	7	7
(v_3, v_5)	7	7
(v_3, v_6)	7	7
(v_4, v_7)	10	10
(v_5, v_7)	10	10
(v_6, v_7)	10	9
(v_7, v_8)	15	15

Вычислим значения, приведенные в табл. 8. Значения $\tau_{v'}$ мы получаем из табл. 6, значения выражения $\tau_v + \chi(v, j_v) + \sigma(e)$ вычисляются с использованием данных из табл. 5, табл. 6 и табл. 7.

Из таблицы видно, что для любой дуги (v, v') выполняется условие $\tau_{v'} \geq \tau_v + \chi(v, j_v) + \sigma(e)$, что равносильно

$$\tau_{fin(e)} \geq \tau_{init(e)} + \chi\big(init(e), j_{init(e)}\big) + \sigma(e).$$

Следовательно, условие (12) для расписания ξ выполняется.

Покажем, что выполняется условие (13), означающее, что количество ядер, выделяемое задаче v_1, не превышает границы линейной масштабируемости, заданной разметкой γ в контексте формулы (4). Для этого заполним табл. 9. Значения j_v взяты из табл. 6, значения m_v – из табл. 2. Мы видим, что для всех вершин v графа G выполняется необходимое условие $j_v \leq m_v$.

Табл. 9. Данные для проверки условия (13)

Вершина v	j_v	m_v
v_1	2	2
v_2	2	2
v_3	2	2
v_4	2	2
v_5	2	3
v_6	2	2
v_7	1	1
v_8	2	2

Табл. 10. Время запуска и останова для кластера W_0

Вершина v	**Время запуска τ_v**	**Время останова s_v**
v_1	0	1
v_2	1	4
v_8	15	17

Проверим условие (14) означающее, что в любой момент времени t общее количество процессорных ядер, выделяемых задачам на узле с номером i, не может превосходить количества ядер на этом узле . В нашем примере граф G был разбит на два кластера W_0 и W_1. Поэтому нам достаточно проверить условие (14) для каждого кластера в отдельности. Сначала проверим условие (14) для кластера W_0. Для этого заполним табл. 10, содержащую данные о времени запуска и останова задач, ассоциированных с вершинами кластера W_0. Для каждой вершины v время запуска τ_v получаем из табл. 6, а время останова s_v рассчитывается по формуле (11).

Теперь построим табл. 11. Согласно табл. 10 мы должны рассмотреть моменты времени t от 1 до 17 (нижняя граница временного интервала равна <наименьшее время запуска>+1, верхняя граница равна <наибольшее время останова>. Вычислим значения $j_{v_1}, j_{v_2}, j_{v_8}$ для вершин, входящих в кластер W_0, для момента времени t_1=1 (первая строка таблицы). Из табл. 10 мы видим, что условие $\tau_{v_1} < t_1 \leq s_{v_1}$ – истинно. Следовательно, задача v_1 выполняется в момент времени t_1. Из табл. 6 мы находим количество процессорных ядер $j_{v_1} = 2$, выделяемых задаче v_1 в соответствии с заданным расписанием ξ. Помещаем это число в первую строку таблицы в колонку «j_{v_1}». Для задачи v_1, в соответствии с табл. 10, мы видим, что условие $\tau_{v_2} < t_2 \leq s_{v_2}$ – ложно. Следовательно, задача v_2 не выполняется в момент времени t_1. Поэтому в столбец «j_{v_2}» первой стоки помещаем значение 0. По этой же причине в столбце « j_{v_8} » первой стоки также помещается значение 0.

Табл. 11. Данные для проверки условия (14) для кластера W_0

Момент времени t	**Количество ядер, выделенных задаче**			$\sum_{\substack{v\in W_0\wedge \\ \tau_v<t\le s_v}} j_v$
	j_{v_1}	j_{v_2}	j_{v_8}	
1	2	0	0	2
2	0	2	0	2
3	0	2	0	2
4	0	2	0	2
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0
9	0	0	0	0
10	0	0	0	0
11	0	0	0	0
12	0	0	0	0
13	0	0	0	0
14	0	0	0	0
15	0	0	0	0
16	0	0	2	2
17	0	0	2	2

Аналогичным образом заполняем остальные строки таблицы. Для каждой строки считаем сумму значений из столбцов «$j_{v_1}, j_{v_2}, j_{v_8}$». Очевидно, что полученное число будет совпадать со значением выражения $\sum_{\substack{v\in W_0\wedge \\ \tau_v<t\le s_v}} j_v$. Таким образом мы получаем

$$\forall t \in [1,17]\left(\sum_{\substack{v\in W_0\wedge \\ \tau_v<t\le s_v}} j_v \le 4 = \left|P_0\right|\right).$$

Поскольку

$$\forall t \notin [1,17]\left(\sum_{\substack{v\in W_0\wedge \\ \tau_v<t\le s_v}} j_v = 0\right),$$

Табл. 12. Время запуска и останова для кластера W_1

Вершина v	**Время запуска τ_v**	**Время останова s_v**
v_3	3	7
v_4	7	10
v_5	7	10
v_6	7	9
v_7	10	14

получаем, что в любой момент времени t выполняется неравенство

$$\sum_{\substack{v\in W_0\wedge \\ \tau_v<t\le s_v}} j_v \le |P_0|,$$

что обеспечивает выполнение условие (14) для кластера W_0.

Теперь проверим выполнение условия (14) для кластера W_1. Сначала строим табл. 12. Затем строим табл. 13, из которой мы видим, что

$$\forall t\in[4,14]\left(\sum_{\substack{v\in W_1\wedge \\ \tau_v<t\le s_v}} j_v \le 6 = |P_1|\right).$$

Табл. 13. Данные для проверки условия (14) для кластера W_1

Момент времени t	**Количество ядер, выделенных задаче**					$\sum_{\substack{v\in W_1\wedge \\ \tau_v<t\le s_v}} j_v$
	j_{v_3}	j_{v_4}	j_{v_5}	j_{v_6}	j_{v_7}	
4	2	0	0	0	0	2
5	2	0	0	0	0	2
6	2	0	0	0	0	2
7	2	0	0	0	0	2
8	0	2	2	2	0	6
9	0	2	2	2	0	6
10	0	2	2	0	0	4
11	0	0	0	0	1	1
12	0	0	0	0	1	1
13	0	0	0	0	1	1
14	0	0	0	0	1	1

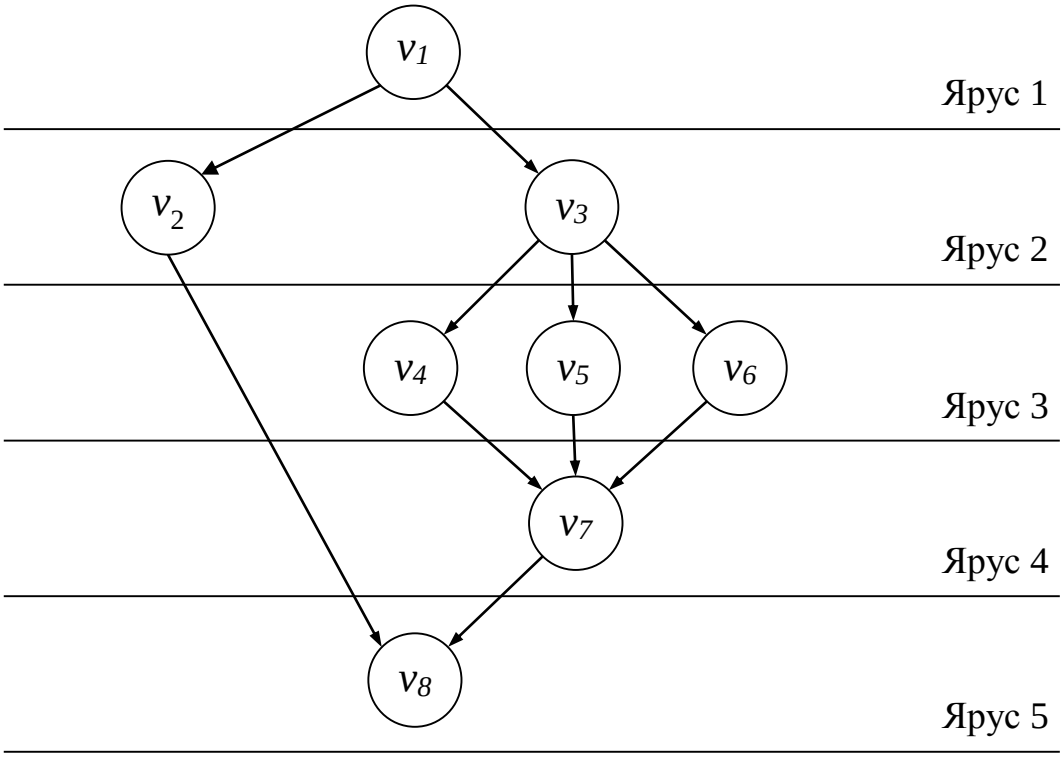

Рис. 15. Каноническая ЯПФ.

Поскольку

$$\forall t \notin [4,14]\left(\sum_{\substack{v\in W_1\wedge \\ \tau_v<t\le s_v}} j_v = 0\right),$$

получаем, что в любой момент времени t выполняется неравенство

$$\sum_{\substack{v\in W_1\wedge \\ \tau_v<t\le s_v}} j_v \le |P_1|,$$

что обеспечивает выполнение условие (14) для кластера W_1, а значит и для всего графа *G*.

Таким образом, мы показали, что условия (12)-(14) выполняются для расписания ξ, следовательно оно корректно.

Построим каноническую ЯПФ для графа G, приведенного на рис. 13. Для этого мы используем алгоритм, приведенный в [1]. В графе G выделим

все входные вершины и присвоим им номер яруса 1. Для графа G в ярусе с номером 1 окажется только вершина $\{v_1\} = L_1$. Удалим эту вершину и все выходящие из нее дуги из графа G. В получившимся графе выделим все входные вершины и присвоим им номер яруса 2. В нашем примере в ярусе с номером 2 будут окажутся две вершины: $\{v_2, v_3\} = L_2$. Продолжим этот процесс, пока не исчерпаем все вершины графа. Результат разбиения графе G в каноническую ЯПФ изображен на рис. 15.

2.3. Алгоритм планирования ресурсов POS

В данном разделе описывается новый *алгоритм POS* (*Problem-Oriented Scheduling*) планирования ресурсов в распределенных проблемно-ориентированных средах. Отличительной особенностью алгоритма POS является то, что при планировании ресурсов он учитывает знания о специфике предметной области. В рамках модели, описанной в разделе 2.2, эти знания выражаются в метках задач-вершин, задающих время выполнения задачи на одном ядре и ее масштабируемость, и в весах дуг, задающих объемы передаваемых данных.

Для упрощения описания и понимания алгоритма *POS* мы будем использовать трехуровневую структуру процедур, представляющих алгоритм. Процедура первого уровня является головной. Некоторый шаг процедуры первого уровня может быть описан как процедура второго уровня. Такой шаг выделяется полужирным шрифтом. Аналогичный подход может быть применен в описании процедуры второго уровня.

2.3.1. Головная процедура

Пусть задана вычислительная система в виде упорядоченного множества вычислительных узлов: $\mathfrak{P} = \{P_0, \dots, P_{k-1}\}$. Пусть имеется граф задания $G = \langle V, E, init, fin, \delta, \gamma \rangle$. Предположим, что выполняются следующие условия:

$$|V| \le |\mathfrak{P}|; \tag{18}$$

$$\forall v \in V\left(\forall P \in \mathfrak{P}\left(m_v \le |P|\right)\right), \tag{19}$$

где m_v – порог линейной масштабируемости, задаваемый функцией разметки γ. Зададим для графа G разбиение в каноническую ЯПФ с ярусами L_i $(i = 1, \dots, r)$. Пронумеруем вершины $V = \{v_1, \dots, v_q\}$ графа G таким образом, чтобы выполнялось следующее свойство:

$$\forall i, j \in \{1,\dots,q\}\left(\left(v_i \in L_a \wedge v_j \in L_b \wedge a < b\right) \Rightarrow i < j\right), \tag{20}$$

то есть на нижних ярусах располагаются вершины с большими номерами.

В самом общем виде головная процедура выглядит следующим образом:

Шаг 1. **Построить начальную конфигурацию G_0**;

Шаг 2. $i := 0$;

Шаг 3. **Построить конфигурацию G_{i+1}**;

Шаг 4. Если остались нерассмотренные дуги, $i := i + 1$ и перейти на шаг 3;

Шаг 5. **Уплотнить конфигурацию G_{i+1}**;

Шаг 6. Стоп.

Таким образом, работа процедуры заключается в построении последовательности конфигураций. При этом, при переходе к очередной конфигурации, как минимум одна дуга графа помечается как просмотренная. Поскольку количество дуг конечно, процедура завершится на некоторой итерации. Последняя построенная конфигурация G_{i+1} выбирается как результирующая.

2.3.2. Процедура построения начальной конфигурации G_0

Шаг 1.1. Зададим функцию начальной кластеризации ω_0 следующим образом:

$$\forall i \in \{1,\dots,q\}\left(\omega_0(v_i) = P_{i-1}\right), \tag{21}$$

то есть каждая вершина отображается на отдельный вычислительный узел, и, соответственно, каждый кластер включает в себя только одну вершину.

Шаг 1.2. Зададим начальное расписание $\xi_0(v) = (\tau_v, j_v)$ следующим образом. Определим время запуска τ_v рекурсивно по уровням ЯПФ:

$$\left.\begin{array}{l} \forall v \in L_1 \left(\tau_v := 0\right); \\ \forall v \in L_{i>1} \left(\tau_v := \max\limits_{\substack{v'' \in L_i; \\ v' \in L_{j<i}}} \left(\lambda(v', v'')\right) \right). \end{array}\right\} \quad (22)$$

Здесь

$$\lambda(v', v'') = \begin{cases} s_{v'}, \text{ если } (v', v'') \notin E; \\ s_{v'} + \sigma((v'', v')), \text{ если } (v', v'') \in E; \end{cases} \quad (23)$$

где $s_{v'}$ вычисляется по формуле (11). Определим количество ядер j_v, выделяемых вершине v, следующим образом:

$$\forall v \in V \left(j_v = m_v \right). \quad (24)$$

Шаг 1.3. $G_0 := \langle V, E, init, fin, \delta, \gamma, \omega_0, \xi_0 \rangle$;

Шаг 1.4. Конец процедуры.

Утверждение 2. Расписание ξ_0, построенное в приведенной процедуре, является корректным.

Доказательство. Нам необходимо и достаточно проверить выполнение условий (12)-(14). Проверим сначала условие (12). Пусть $e \in E$. Из (22) имеем

$$\tau_{fin(e)} \geq s_{init(e)} + \sigma(e) = \tau_{init(e)} + \chi\left(init(e), j_{init(e)}\right) + \sigma(e),$$

то есть условие (12) выполняется. Условие (13) выполняется в силу (24). Условие (14) вытекает из (19), (21) и (24). Утверждение доказано.

2.3.3. Процедура построения конфигурации G_{i+1}

Определим *субкритический путь*, как путь, имеющий максимальную стоимость среди всех путей, содержащих хотя бы одну нерассмотренную дугу.

Шаг 3.1. Найти в G_i субкритический путь $\tilde{y}_i = (e_1, \ldots, e_n)$ (если таких путей несколько, выбрать любой из них);

Шаг 3.2. Найти первую от начала пути нерассмотренную дугу e_j $(1 \le j \le n)$ в $\tilde{y}_i$ и пометить ее как рассмотренную;

Шаг 3.3. Если $i = 0$, то пометить вершину $init(e_j)$ как зафиксированную;

Шаг 3.4. Если вершины $init(e_j)$ и $fin(e_j)$ зафиксированы, то перейти на шаг 3.14;

Шаг 3.5. Если вершина $fin(e_j)$ не зафиксирована, то

$v'' := fin(e_j), v' := init(e_j)$;

Шаг 3.6. Если вершина $init(e_j)$ не зафиксирована, то

$v'' := init(e_j), v' := fin(e_j)$;

Шаг 3.7. Построить функцию кластеризации ω_{i+1}, отличающуюся от функции ω_i только в одном значении: $\omega_{i+1}(v'') := \omega_i(v')$;

Шаг 3.8. **Построить расписание** ξ_{i+1};

Шаг 3.9. $G_{i+1} := \langle V, E, init, fin, \delta, \gamma, \omega_{i+1}, \xi_{i+1} \rangle$

Шаг 3.10. Найти критический путь $\bar{y}_i$ в G_i (если таких путей несколько выбрать любой из них);

Шаг 3.11. Найти критический путь $\bar{y}_{i+1}$ в G_{i+1} (если таких путей несколько выбрать любой из них);

Шаг 3.12. Если $u(\bar{y}_{i+1}) \le u(\bar{y}_i)$, то перейти на шаг 3.16;

Шаг 3.13. $G_{i+1} := G_i$

Шаг 3.14. Если в $\tilde{y}_i$ остались нерассмотренные дуги, перейти на шаг 3.2;

Шаг 3.15. Если в G_i остались нерассмотренные дуги, перейти на шаг 3.1.

Шаг 3.16. Конец процедуры.

2.3.4. Процедура построения расписания ξ_{i+1}

Введем следующие обозначения: $T(x)$ – номер яруса, которому принадлежит вершина x; $W_{\omega_i(x)} = \{v \mid v \in V, \omega_i(v) = \omega_i(x)\}$ – кластер, которому принадлежит вершина x.

Шаг 3.8.1 $R := W_{\omega_i(v')} \cap L_{T(v'')}$;

Шаг 3.8.2 Если $R = \varnothing$ или $\sum_{v \in R} j_v \le \left| P_{\omega_i(v')} \right|$ то перейти на шаг 3.8.7;

Шаг 3.8.3 Для $h = q, \ldots, T(fin(e_j)) + 1$ выполнить $L_{h+1} := L_h$;

Шаг 3.8.4 $L_{T(v'')+1} := \{v''\}$;

Шаг 3.8.5 $L_{T(v'')} := L_{T(v'')} \setminus \{v''\}$;

Шаг 3.8.6 $q := q + 1$;

Шаг 3.8.7 Построить новое расписание ξ_{i+1} путем вычисления времени запуска τ_v всех вершин $v \in V$ с помощью формулы (22);

Шаг 3.8.8 Пометить вершину v'' как зафиксированную;

Шаг 3.8.9 Конец процедуры.

2.3.5. Процедура уплотнения конфигурации G_{i+1}

Целью процедуры уплотнения является минимизация количества задействованных вычислительных узлов. Данная процедура применяется только к кластерам, содержащим одну вершину. Указанное ограничение мотивировано тем, что, если в кластере имеются две смежные вершины, то перемещение одной из них в другой кластер может увеличить общее время выполнения задачи.

Шаг 5.1. . $\mathfrak{W} := \varnothing$;

Шаг 5.2. Для всех $v' \in V$ выполнить цикл

Шаг 5.2.1. $W = \{v \mid v \in V, \omega_{i+1}(v) = \omega_{i+1}(v')\}$;

Шаг 5.2.2. Если $W \in \mathfrak{W}$, перейти к следующей итерации цикла;

Шаг 5.2.3. $\mathfrak{W} := \mathfrak{W} \cup \{W\}$;

Шаг 5.3. Конец цикла;

Шаг 5.4. $\mathfrak{E} := \left\{W \in \mathfrak{W} \middle\| W| = 1\right\}$; $\mathfrak{V} := \left\{W \in \mathfrak{W} \middle\| W| > 1\right\}$

Шаг 5.5. Для всех $W' \in \mathfrak{E}$ выполнить цикл

Шаг 5.5.1. Для $l = 1, \ldots, r$ выполнить цикл

Шаг 5.5.1.1. Если $W' \cap L_l = \varnothing$, перейти к следующей итерации цикла;

Шаг 5.5.1.2. Для всех $W'' \in \mathfrak{V}$ выполнить цикл

Шаг 5.5.1.2.1. Если $W'' \cap L_l = \varnothing$, перейти к следующей итерации цикла;

Шаг 5.5.1.2.2. Взять $v' \in W'$;

Шаг 5.5.1.2.3. Взять $v'' \in W''$;

Шаг 5.5.1.2.4. Если $m_{v'} + \sum_{v \in W'' \cap L_l} m_v > \left|\omega_{i+1}(v'')\right|$, перейти к следующей итерации цикла;

Шаг 5.5.1.2.5. $i := i + 1$;

Шаг 5.5.1.2.6. Построить функцию кластеризации ω_{i+1}, отличающуюся от функции ω_i только в одном значении: $\omega_{i+1}(v') := \omega_i(v'')$;

Шаг 5.5.1.2.7. $W'' := W'' \cup W'$

Шаг 5.5.1.2.8. Перейти на Шаг 5.5.3;

Шаг 5.5.1.3. Конец цикла;

Шаг 5.5.2. Конец цикла;

Шаг 5.5.3. Перейти к следующей итерации цикла;

Шаг 5.6. Конец цикла;

Шаг 5.7. Конец процедуры.

Дадим некоторые пояснения к процедуре уплотнения. Шаг 5.2 организует цикл, строящий $\mathfrak{W}$ – множество всех кластеров, на которые разбито задание. Шаг 5.4 вычисляет $\mathfrak{E}$ – множество *единичных кластеров* (кластеров, содержащих только одну вершину) и $\mathfrak{V}$ – множество *мультикластеров* (кластеров, содержащих две или более вершин). Шаг 5.5 организует цикл по всем единичным кластерам. Для каждого единичного кластера находим ярус параллельной формы, к которому принадлежит данный единичный кластер. В рамках этого яруса пытаемся объединить единичный кластер с некоторым мультикластером. Это возможно, если мультикластер задействует не все процессорные ядра узла, и количество свободных ядер достаточно для выполнения присоединяемого единичного кластера.

Планом выполнения расписания ξ называется множество многозначных отображений

$$\alpha = \{\alpha_i : W_i \to P_i | i = 0, \dots, k-1\},$$

удовлетворяющих следующим условиям $\forall i = 0, \dots, k-1$ и $(\tau_v, j_v) := \xi(v)$:

$$\forall\, v \in W_i \; (\, j_v = |\alpha_i(v)|); \tag{25}$$

$$\forall\, v_1, v_2 \in W_i \left(v_1 \neq v_2 \wedge \left[\tau_{v_1}, s_{v_1}\right] \cap \left[\tau_{v_2}, s_{v_2}\right] \neq \emptyset \;\Rightarrow\; \alpha_i(v_1) \cap \alpha_i(v_2) = \emptyset\right). \tag{26}$$

Условие (25) означает, что количество процессорных ядер, выделяемых задаче v в соответствии с планом α, равно количеству процессорных ядер, указанных для этой задачи в расписании ξ. Условие (26) означает, что, если в соответствии с расписанием ξ задачи v_1, и v_2 пересекаются по времени своего выполнения, то им должны выделяться различные процессорные ядра (мы предполагаем, что на одном процессорном ядре в каждый момент времени не может выполняться более одной задачи).

Кластеризованный граф с заданными расписанием ξ и соответствующим планом α будем называть *распланированным* и обозначать как $G = \langle V, E, init, fin, \delta, \gamma, \omega, \xi, \alpha \rangle$.

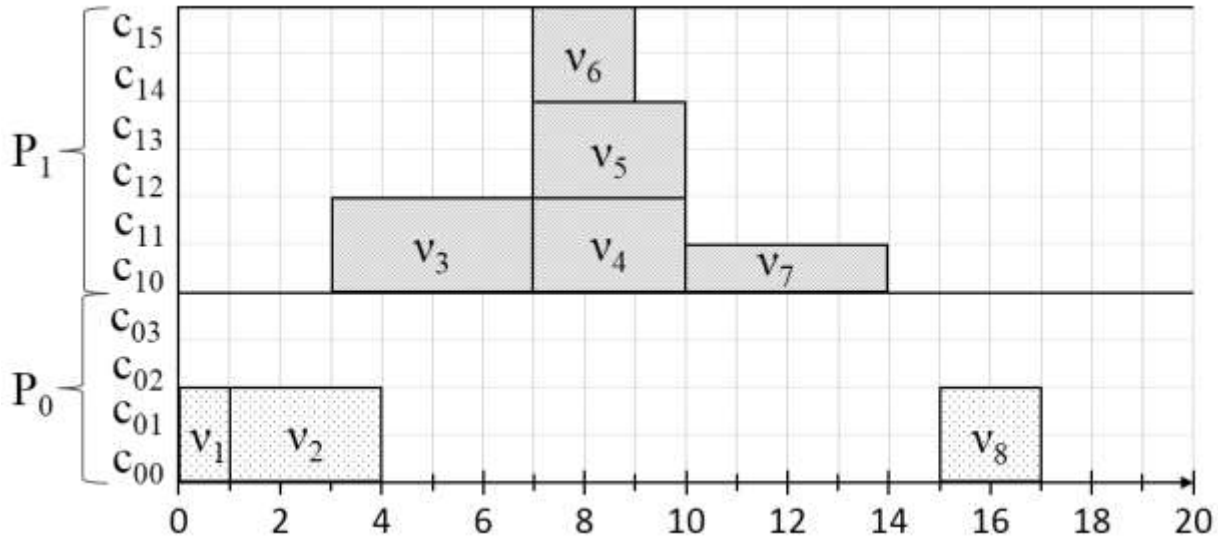

Рис. 16. Диаграмма Ганта.

На рис. 16 представлена диаграмма Ганта [107] для расписания ξ, которая в графическом виде описывает временное и пространственное размещение задач графа G на вычислительных узлах $P_i \in \mathfrak{P},\ i = \{0,1\}$. По вертикальной оси диаграммы Ганта отмечены процессорные ядра, по горизонтальной оси – время. Каждой задаче сопоставлена прямоугольная область, показывающая на каких ядрах она выполняется и какое время занимает. Данные для построения прямоугольников берутся из табл. 10, табл. 12.

Мы видим, что задачи, размещенные на одном вычислительном узле, не требуют времени на передачу данных между ними. Можно также отметить, что задачи, ассоциируемые с вершинами v_4, v_5 и v_6, являются независимыми по ресурсам.

2.4. Выводы по главе 2

В главе 2 были даны определения основных понятий проблемно-ориентированных сред и приведен пример проблемно-ориентированной среды компьютерного инженерного проектирования в области вычислительной гидро-газодинамики с помощью инженерного пакета ANSYS CFX. Предложена математическая метамодель проблемно-ориентированной распределенной вычислительной среды. В данной метамодели вычислительное задание,

представляющее собой поток работ, моделируется в виде размеченного взвешенного ориентированного ациклического графа задания. Каждая задача-вершина помечается парой натуральных чисел, первое из которых задает время выполнения задачи на одном процессорном ядре, а второе – максимальное количество ядер, до которого задача демонстрирует ускорение близкое к линейному. Веса дуг задают объем данных, передаваемый между задачами. На базе предложенной метамодели вычислительной среды был разработан проблемно-ориентированный алгоритм планирования ресурсов *POS* (*Problem-Oriented Scheduling*) для заданий с потоковой структурой. Отличительной особенностью алгоритма POS является то, что при планировании ресурсов он учитывает знания о специфике предметной области и ориентирован на распределенные вычислительные среды, формируемые на базе вычислительных кластеров с многоядерными ускорителями. Приведен пример планирования графа задания при помощи алгоритма планирования ресурсов POS.

ГЛАВА 3. БРОКЕР РЕСУРСОВ ДЛЯ ПРОБЛЕМНО-ОРИЕНТИРОВАННЫХ СРЕД

На базе описанных в главе 2 методов представления проблемно-ориентированных распределенных вычислительных сред и алгоритма планирования ресурсов разработан брокер ресурсов DiVTB Broker. Брокер ресурсов представляет собой веб-сервис, позволяющий управлять аппаратными и программными ресурсами с помощью дополнительных знаний о проблемно-ориентированной специфике потоков работ в сложных приложениях, которая выражается в указании времени выполнения задачи и объеме передаваемых данных.

Исходные тексты брокера ресурсов свободно доступны в сети Интернет по адресу: https://bitbucket.org/shamakina/pos.

3.1. Модель вариантов использования брокера ресурсов

Раздел содержит спецификацию вариантов использования брокера ресурсов DiVTB Broker для поддержки проблемно-ориентированных распределенных вычислительных сред. Требования к брокеру ресурсов задаются при помощи модели вариантов использования, построенной на основе унифицированного языка моделирования UML версии 2.0. Вариант использования описывается в соответствии со следующей схемой. Во-первых, приводится краткое описание варианта использования и предварительные условия, необходимые для начала его выполнения. Во-вторых, приводится описание основных шагов, составляющих поток событий варианта использования. В-третьих, обсуждаются альтернативные потоки событий и приводятся дополнительные требования.

На

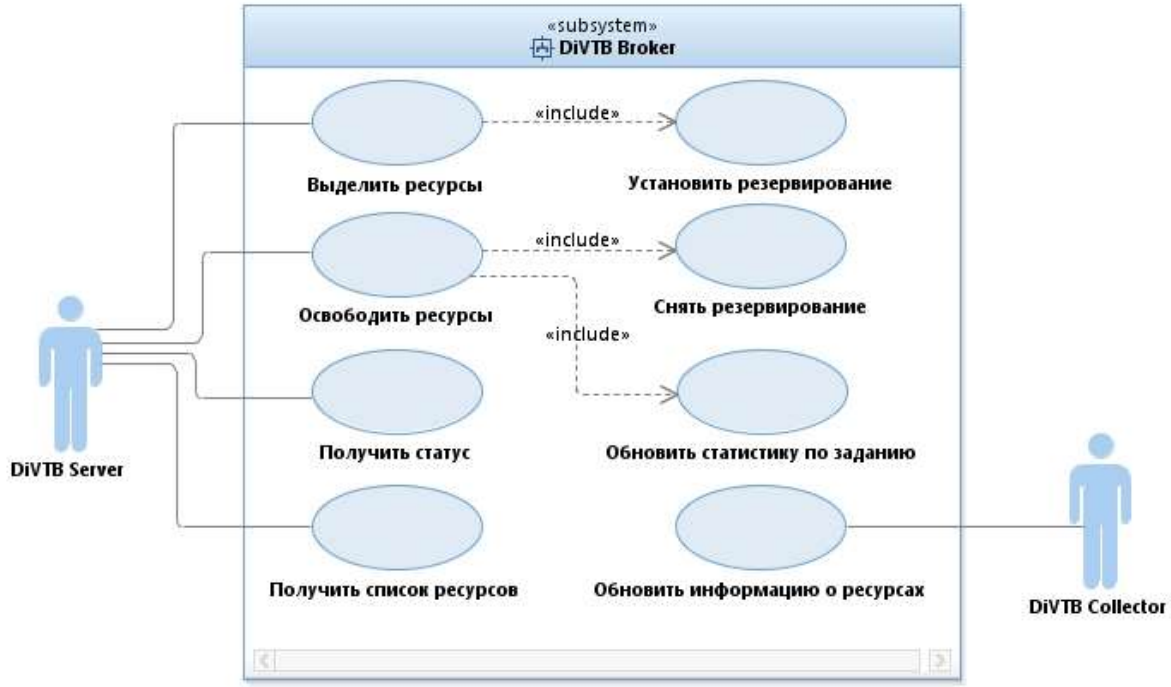

рис. 17 приведены варианты использования брокера ресурсов. Далее приводится детальное описание каждого варианта использования.

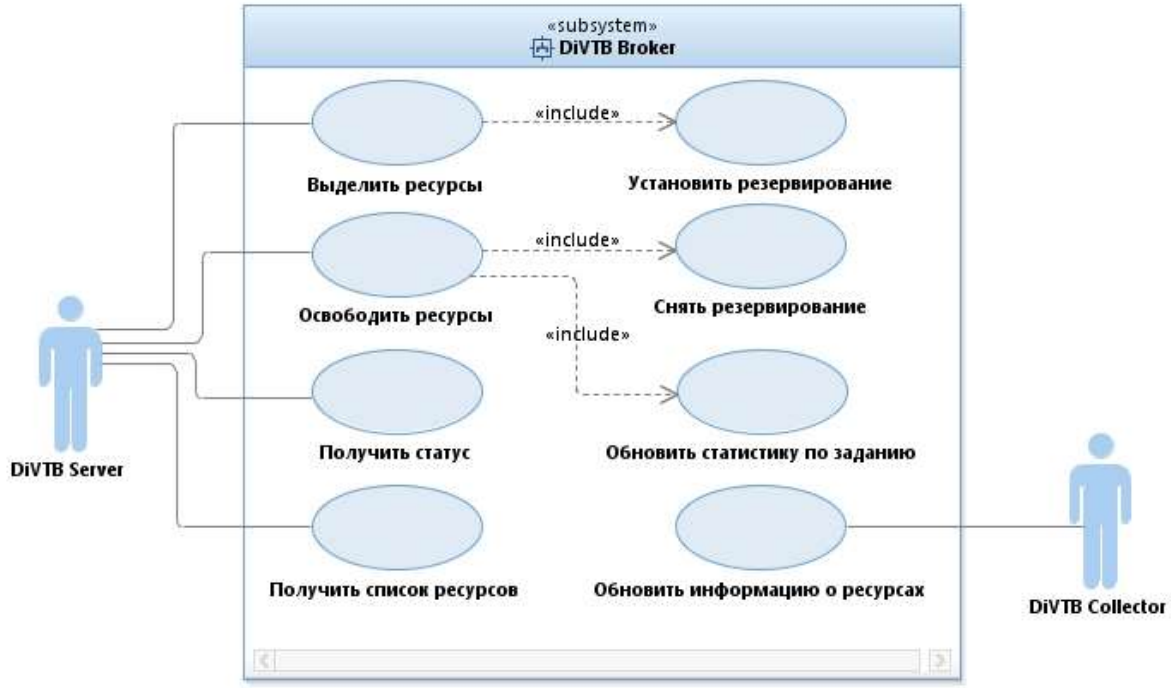

Рис. 17. Диаграмма вариантов использования подсистемы DiVTB Broker.

DiVTB Server представляет собой программный компонент, взаимодействующий с брокером ресурсов. DiVTB Broker отвечает за исполнение заданий и их мониторинг в системе DiVTB. Брокер ресурсов допускает вызов с помощью веб-методов и может быть использован веб-сервисами.

DiVTB Collector представляет собой вспомогательный программный компонент, взаимодействующий с брокером ресурсов. Коллектор осуществляет сбор информации о вычислительных ресурсах в распределенной вычислительной среде.

Вариант использования «Выделить ресурсы». Передает запрос от компонента DiVTB Server к брокеру ресурсов системы DiVTB. Запрос – это XML-документ, в котором задаются программно-аппаратные требования для выполнения задания.

Вариант использования начинается, когда DiVTB Server вызывает функцию «Выделить ресурсы». Основной поток событий состоит из следующих шагов:

1. сохраняет информацию о требованиях задания;
2. создается экземпляр планировщика потока работ.

Альтернативные потоки отсутствуют.

У варианта использования имеется одна точка включения: «Установить резервирование».

Вариант использования «Установить резервирование». Устанавливает резервирование ресурсов. Данный вариант начинается, когда DiVTB Broker выполняет функцию «Выделить ресурсы». Основной поток событий: устанавливается резервирование ресурсов. Альтернативные потоки отсутствуют.

Вариант использования «Освободить ресурсы». Уведомляет брокер ресурсов о необходимости освобождения ресурсов. Вариант использования начинается, когда DiVTB Server вызывает функцию «Освободить ресурсы».

Основной поток событий состоит из следующих шагов:

1. снимается резервирование ресурсов;
2. удаляется экземпляр планировщика потока работ;
3. обновляется статистика по заданиям.

Альтернативные потоки отсутствуют. Для варианта использования задано предусловие: выделены ресурсы.

У варианта использования имеются две точки включения: «Снять резервирование» и «Обновить статистику по заданию».

Вариант использования «Снять резервирование». Снимает резервирование с указанных ресурсов. Данный вариант начинается, когда DiVTB Broker выполняет функцию «Освободить ресурсы». Основной поток событий: снимается резервирование с ресурсов. Альтернативные потоки отсутствуют.

Вариант использования «Обновить статистику по заданию». Обновляет статистику по заданию. Данный вариант начинается, когда DiVTB Broker выполняет функцию «Освободить ресурсы». Основной поток событий: обновляется статистика по выполненным задачам. Статистика включает в себя: параметры задачи; время выполнения задачи; архитектуру вычислительных узлов, на которых выполнена задача. Альтернативные потоки отсутствуют.

Вариант использования «Получить статус». Возвращает информацию о процессе выделения ресурсов. Вариант использования начинается, когда DiVTB Server вызывает функцию «Получить статус». Основной поток событий: получить статус у соответствующего экземпляра планировщика потока работ. Альтернативные потоки отсутствуют. Для варианта использования задано предусловие: выделены ресурсы.

Вариант использования «Получить список ресурсов». Возвращает список выделенных ресурсов для выполнения задания. Вариант использования начинается, когда DiVTB Server вызывает функцию «Получить список ре-

сурсов». Основной поток событий: получить список ресурсов. Альтернативные потоки отсутствуют. Для варианта использования задано предусловие: успешное выделение ресурсов.

Вариант использования «Обновить информацию о ресурсах». Обновляет информацию о вычислительных ресурсах в базе данных брокера ресурсов. Вариант использования начинается, когда коллектор вызывает функцию «Обновить информацию о ресурсах». Основной поток событий: обновить информацию о ресурсах. Альтернативные потоки отсутствуют.

3.2. Архитектура брокера ресурсов DiVTB Broker

Согласно представленной на

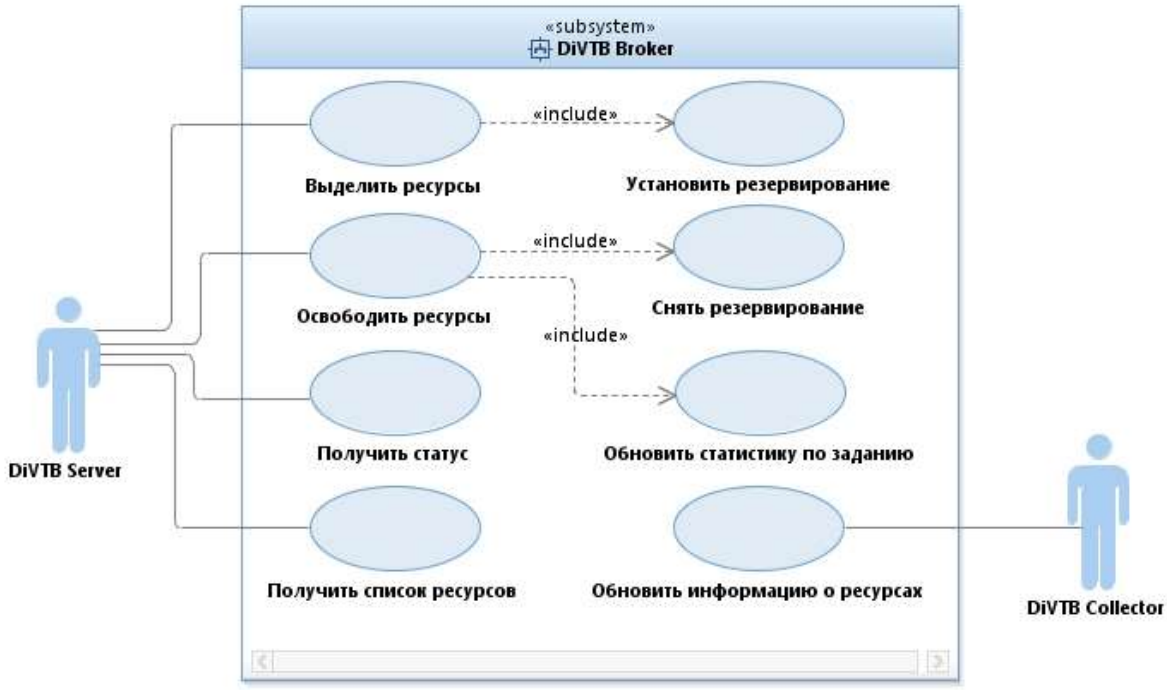

рис. 17 диаграмме вариантов использования была разработана следующая архи-
тектура DiVTB Broker (

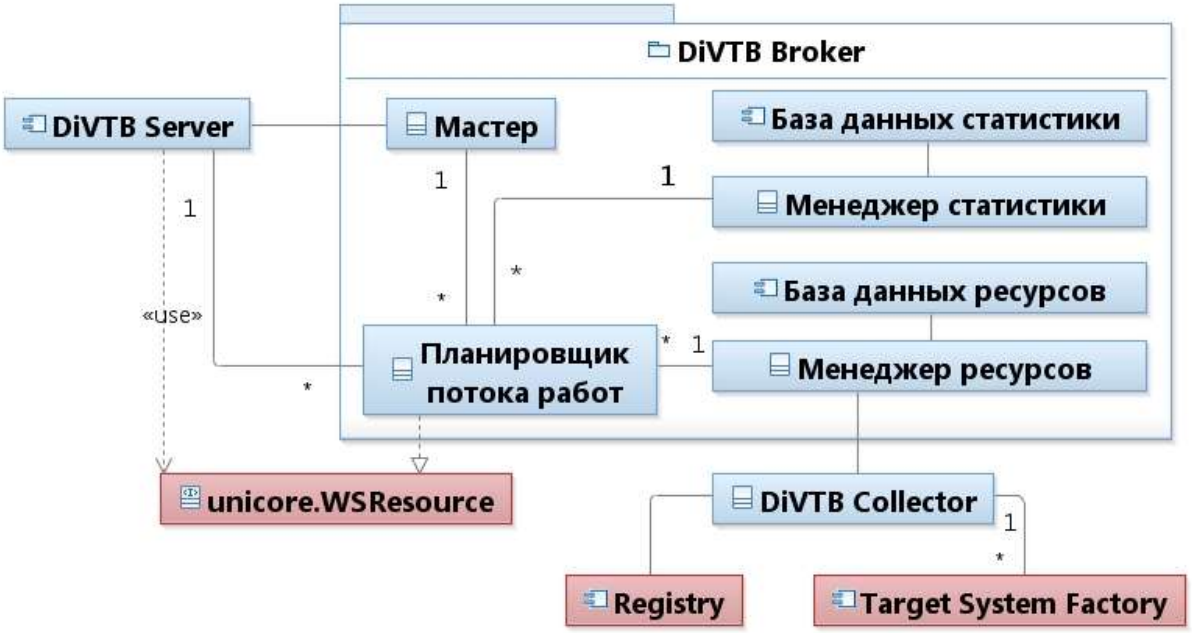

рис. 18).

DiVTB Broker состоит из следующих компонентов.

– Мастер принимает запросы от DiVTB Server и создает экземпляр планировщика потока задач, представляющий собой WS-ресурс в терминах UNICORE.

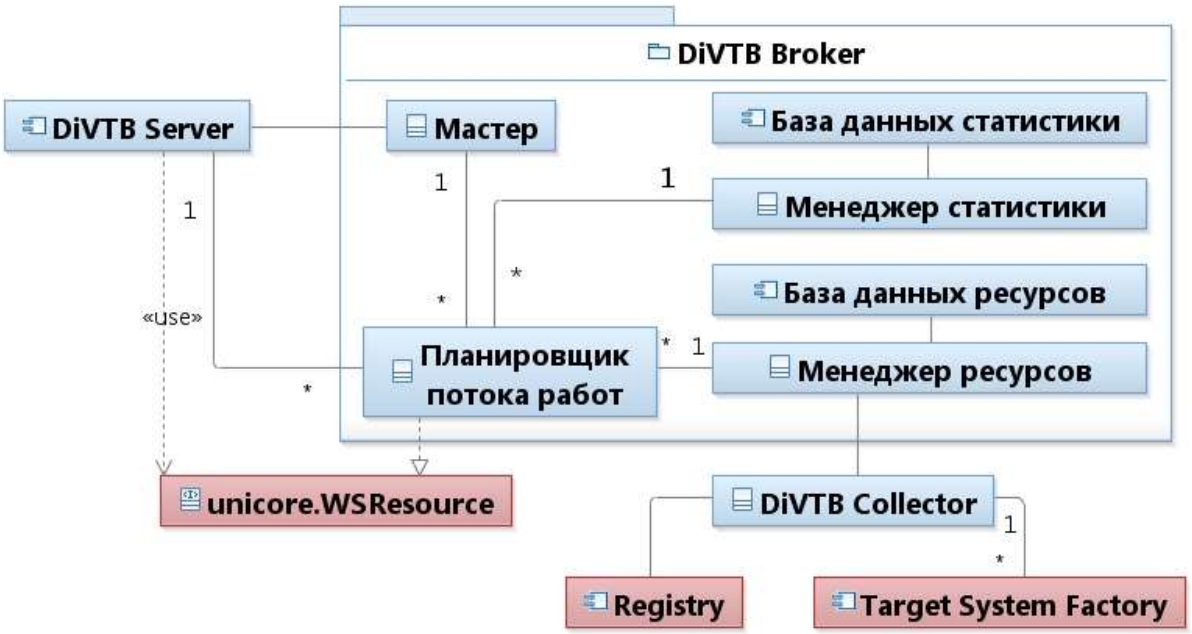

Рис. 18. Архитектура брокера ресурсов DiVTB Broker.

– Экземпляр планировщика потока задач осуществляет обработку одного запроса. Формирует список требуемых для исполнения задания ресурсов и производит их резервирование.
– Менеджер ресурсов управляет базой данных ресурсов, содержащей информацию о целевых системах и резервировании ресурсов.
– Менеджер статистики управляет базой данных статистики, содержащей информацию о статистике выполнения задач.
– Коллектор работает независимо от DiVTB Broker и осуществляет сбор информации для базы данных ресурсов.

Красным цветом на

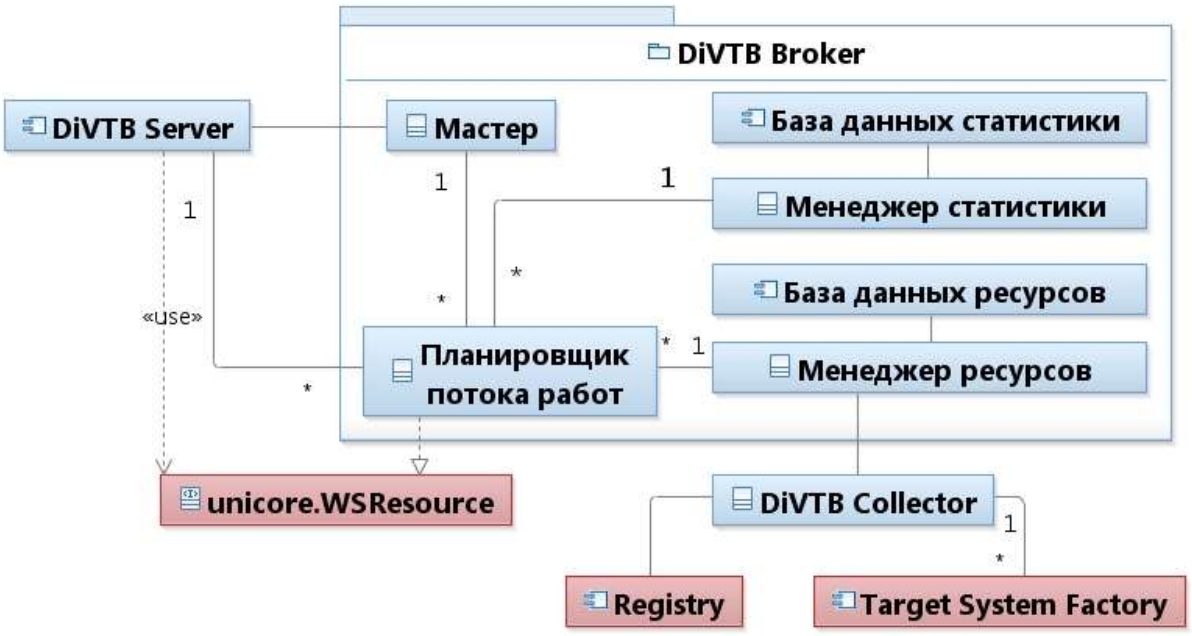

рис. 18 выделены компоненты платформы UNICORE, с которыми взаимодействуют компоненты системы DiVTB.

3.3. Принципы работы брокера ресурсов DiVTB Broker

Взаимодействие компонентов брокера ресурсов, представленных на

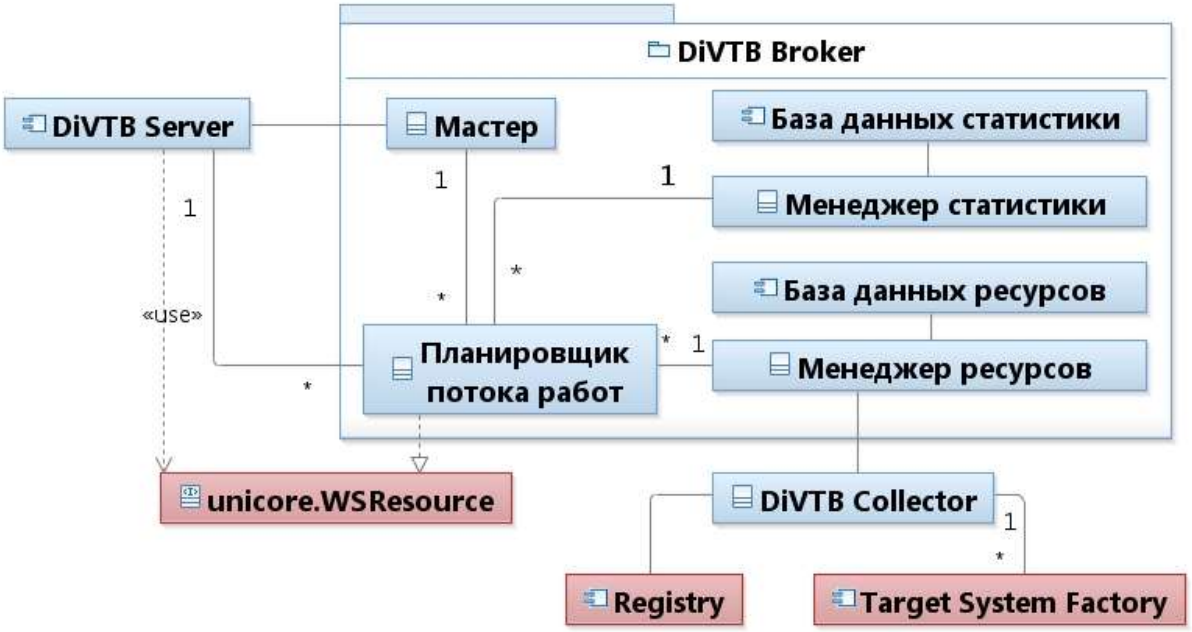

рис. 18, осуществляется согласно диаграммам последовательности выделения ресурсов (

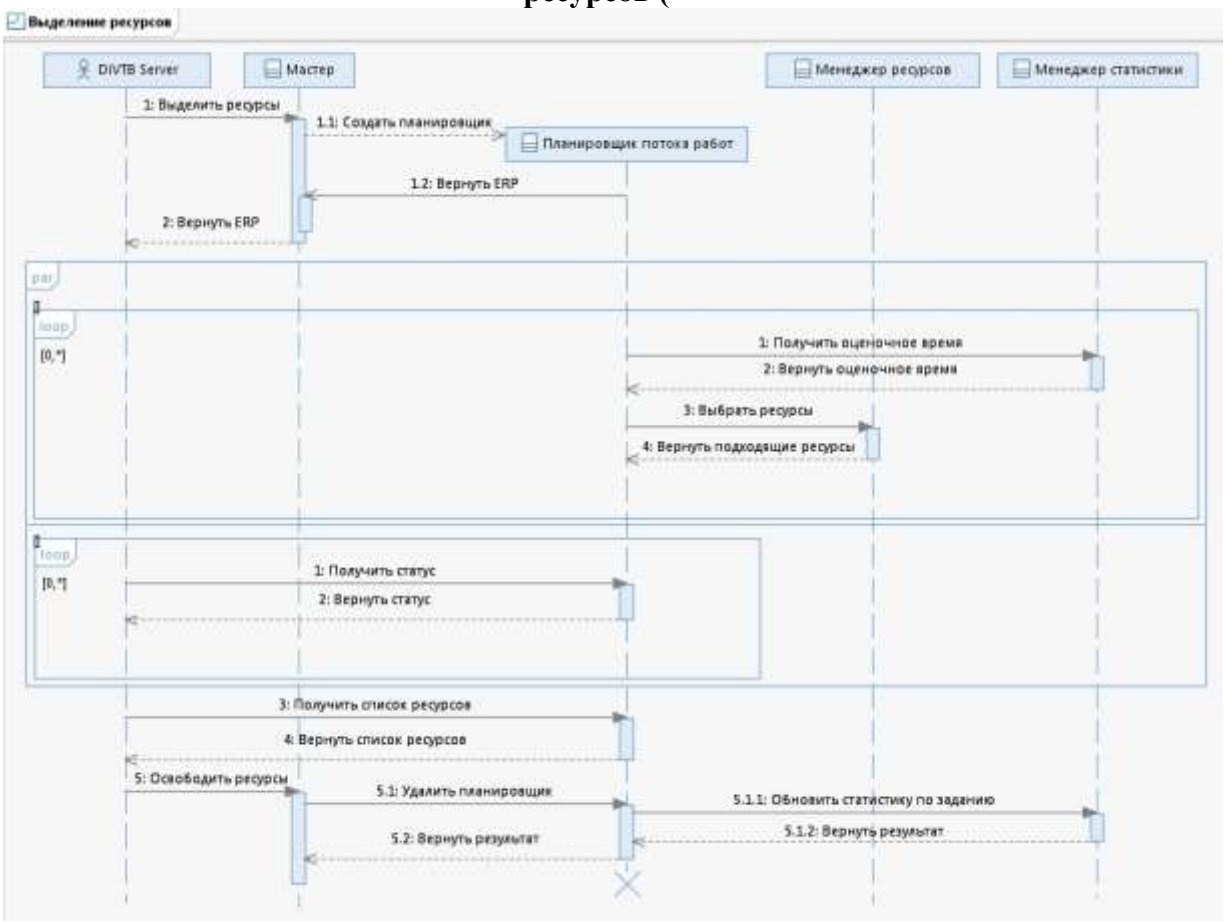

рис. 19) и сбора информации (

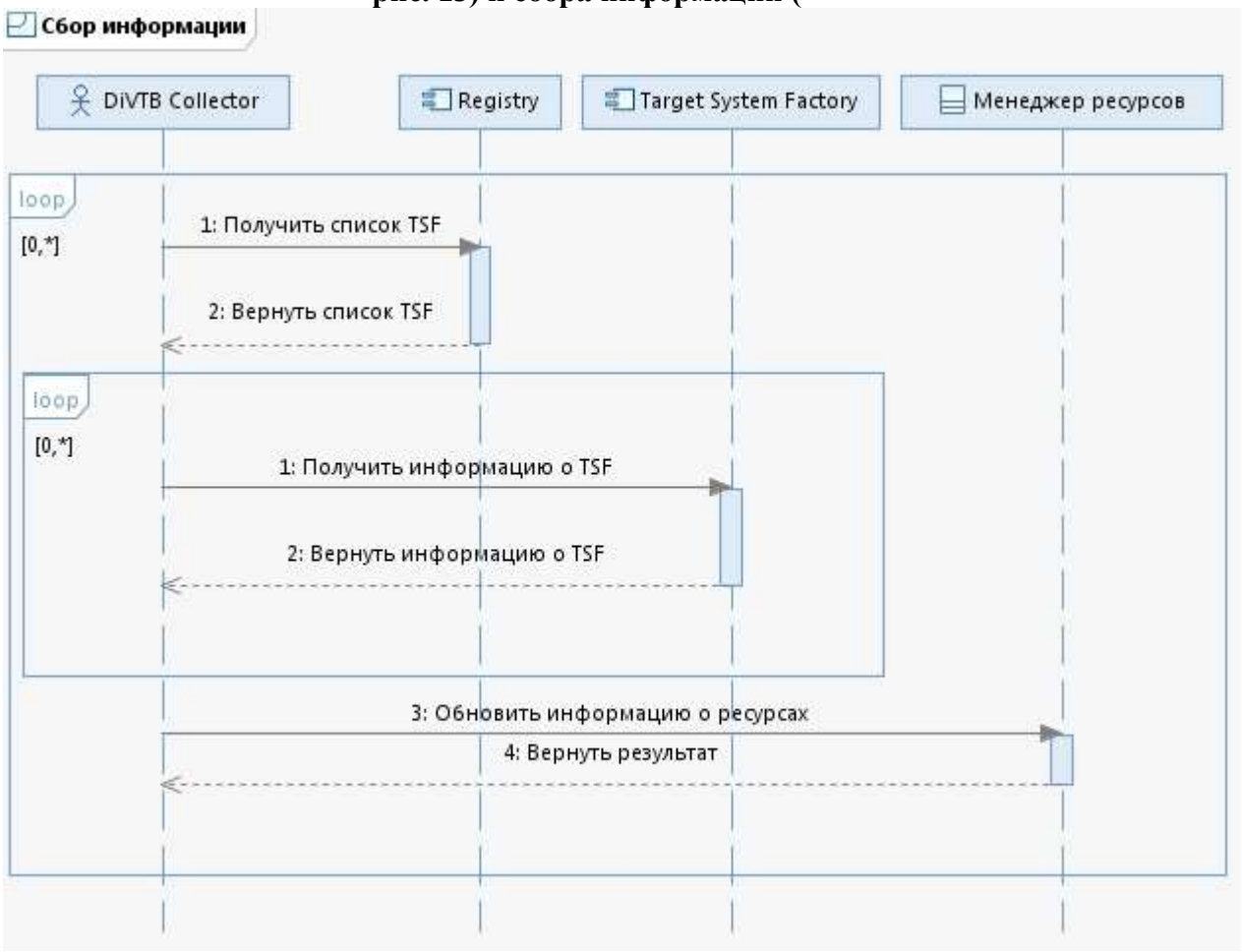

рис. 20).

Рассмотрим процесс выделения ресурсов компонентом DiVTB Broker.

– Компоненту DiVTB Server предоставляется доступ к брокеру ресурсов посредством компонента «Мастер». DiVTB Server подключается к нему с помощью имеющегося сертификата безопасности. Затем DiVTB Server вызывает метод «Выделить ресурсы» для выделения ресурсов заданию и передает методу в качестве входного параметра абстрактный поток работ, содержащий требования к ресурсам для каждой из задач.
– После вызова компонентом DiVTB Server метода «Выделить ресурсы» компонент «Мастер» выполняет следующую последовательность действий.

- Мастер создает экземпляр планировщика потока работ в виде WS-ресурса. Планировщик потока задач возвращает компоненту «Мастер» свой уникальный идентификатор. В дальнейшем, любую информацию о процессе поиска ресурсов для задания можно получить непосредственно у экземпляра планировщика потока работ, обратившись к нему с помощью данного идентификатора.
- Мастер записывает уникальный идентификатор созданного экземпляра планировщика потока работ в свой пул.

  Мастер инициализирует специальный объект - конкретный поток работ, извлекая необходимую информацию из абстрактного потока работ. Конкретный поток работ содержит отображение каждой задачи из задания на конкретный ресурс. Мастер записывает конкретный поток задач с помощью фреймворка Ehcahe сначала в кэш, а потом на диск. В качестве ключа к данному объекту используется уникальный идентификатор планировщика потока работ.
- Мастер возвращает компоненту DiVTB Server уникальный идентификатор созданного экземпляра планировщика потока работ с помощью метода «Вернуть EPR». В качестве идентификатора выступает адрес конечной точки (Endpoint References, EPR).

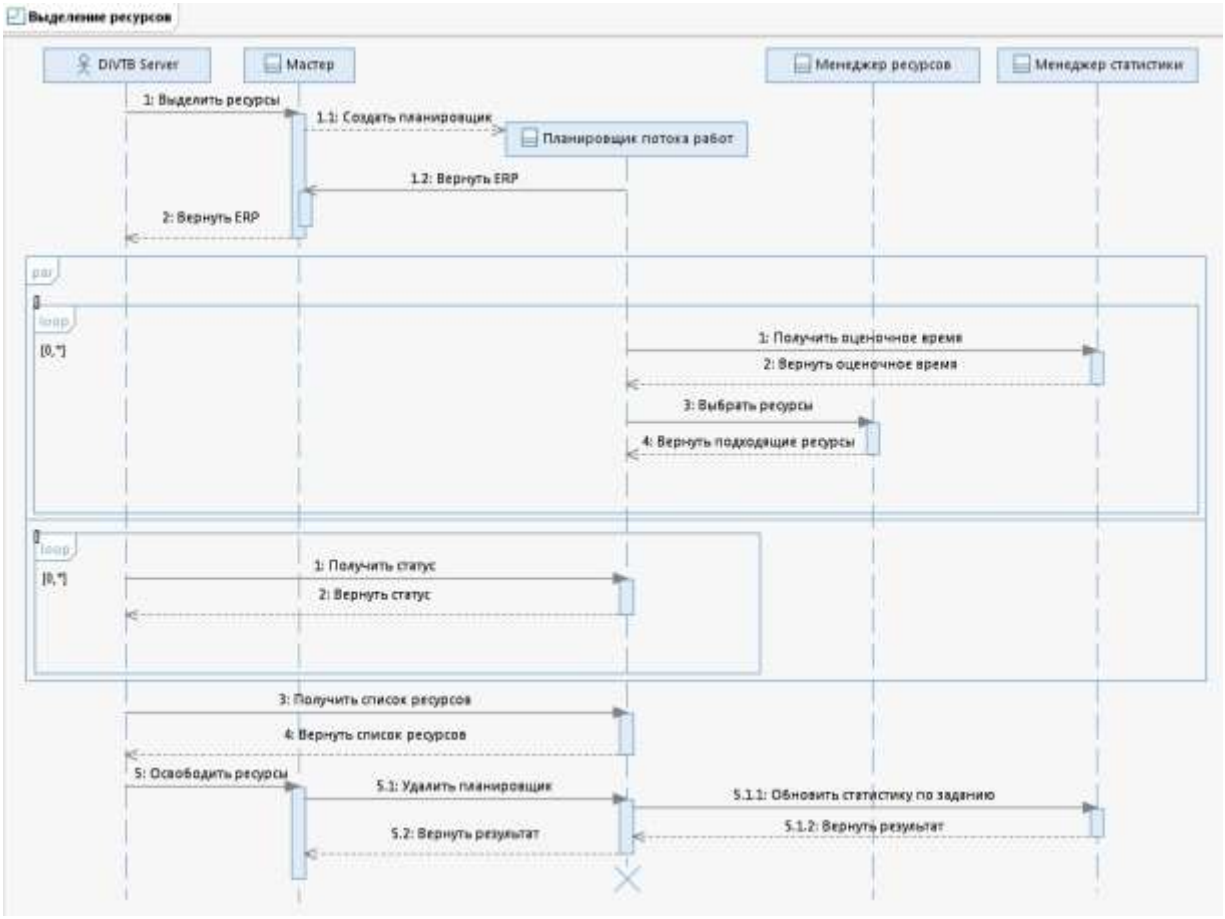

Рис. 19. Диаграмма последовательности выделения ресурсов в DiVTB Broker.

– После создания планировщик потока работ считывает соответствующий конкретный поток работ из кэша или с диска. Дальнейшие шаги последовательно повторяются до завершения процесса выделения ресурсов.
    - Планировщик потока работ получает оценку времени выполнения для каждой задачи у менеджера статистики, вызвав метод «Получить оценочное время».
    - Планировщик потока работ начинает поиск требуемых ресурсов в базе данных ресурсов помощью метода «Выбрать ресурсы» посредством запроса к менеджеру ресурсов. В случае успешного поиска ресурсов планировщик потока задач получает список ресурсов, в противном случае – получает пустое сообщение. Выбранные ресурсы резервируются.

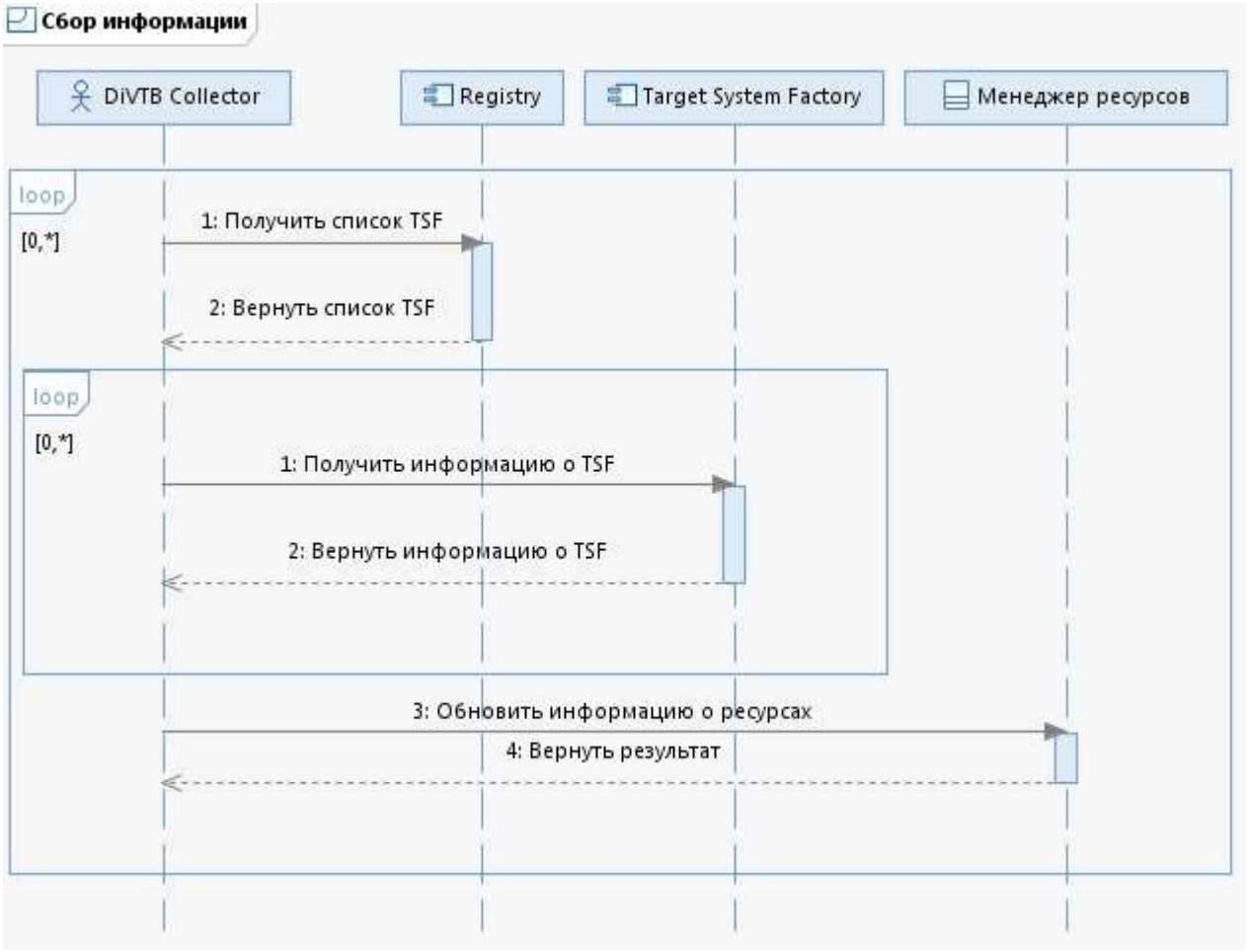

Рис. 20. Диаграмма последовательности сбора информации в системе DiVTB Broker.

- DiVTB Server проверяет статус выделения ресурсов, обращаясь непосредственно к своему экземпляру планировщика потока работ.

– В случае успешного выделения ресурсов DiVTB Server запрашивает список выделенных для его задания ресурсов с помощью метода «Получить список ресурсов».
– После выполнения задания DiVTB Server отправляет компоненту «Мастер» запрос на освобождение ресурсов.
– После вызова компонентом «Мастер» метода «Удалить планировщик» экземпляр планировщика потока задач снимает резервирование с выделенных ресурсов и обновляет статистику по выполнившемуся заданию в базе данных статистики.

Процесс сбора информации для базы данных ресурсов представлен на

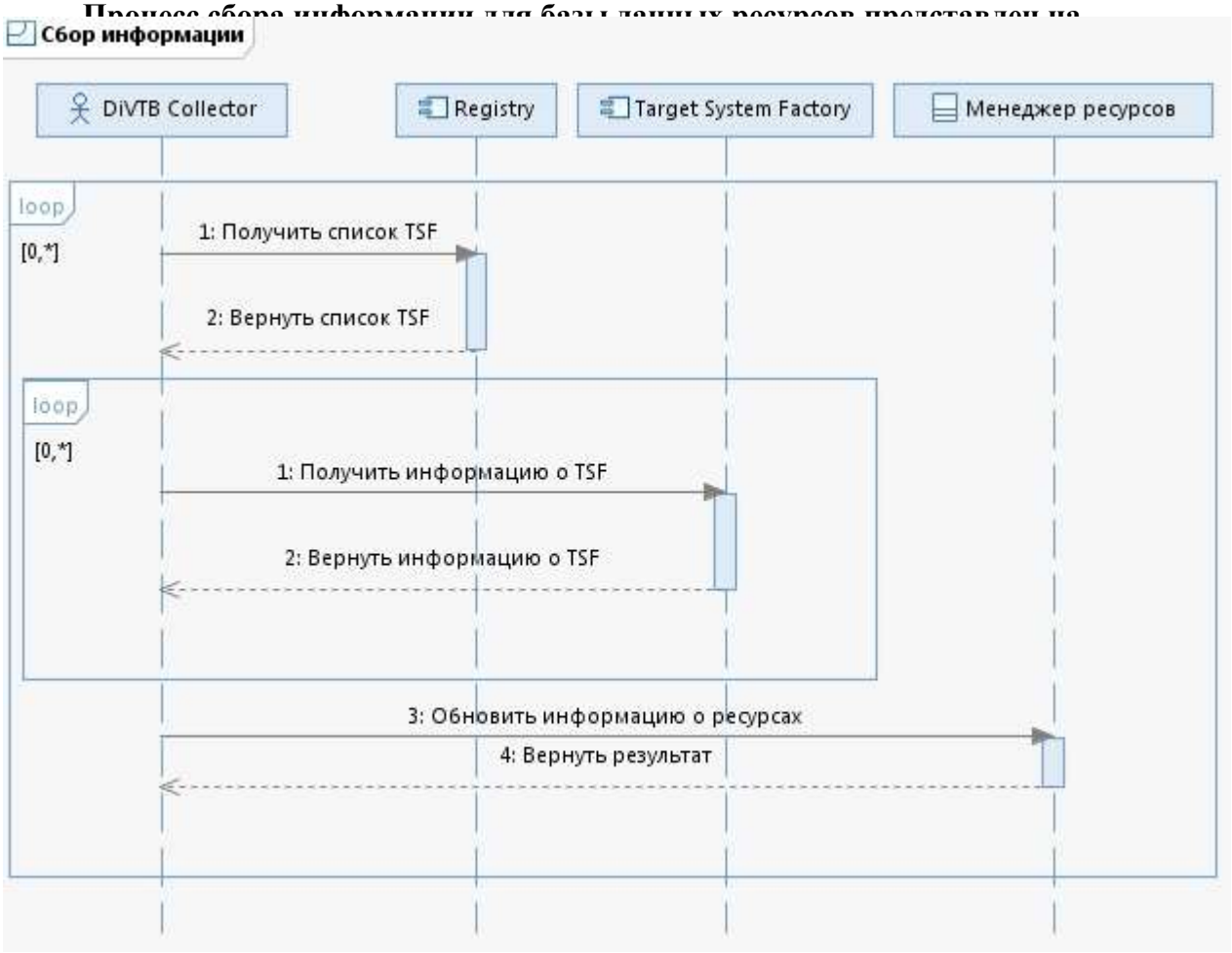

рис. 20. Шаги по обновлению информации повторяются последовательно в течении всего времени исполнения компонента «Коллектор». Обновление информации о ресурсах подразумевает сбор информации об аппаратных и программных ресурсах, доступных DiVTB Broker. Статистика выполнения задач включает информацию о реальном времени выполнения задач с конкретными параметрами на конкретных вычислительных узлах.

3.4. Выводы по главе 3

На базе предложенной модели вычислительной среды был разработан *брокер ресурсов (DiVTB Broker)*, позволяющий эффективно планировать задания в виде потоков работ в проблемно-ориентированных распределенных вычислительных средах. Проектирование брокера производилось при помощи унифицированного языка моделирования UML 2.0. Требования к

DiVTB Broker зафиксированы при помощи модели вариантов использования. Приводятся диаграмма компонентов брокера ресурсов для описания архитектуры *DiVTB Broker*. Построены диаграммы последовательности для операций выделения ресурсов и сбора информации о ресурсах компонентом *DiVTB Collector*. *DiVTB Broker* был реализован на языке Java. Исходные тексты брокера ресурсов свободно доступны в сети Интернет по адресу: https://bitbucket.org/shamakina/pos.

ГЛАВА 4. ВЫЧИСЛИТЕЛЬНЫЕ ЭКСПЕРИМЕНТЫ

В данной главе отражены результаты вычислительных экспериментов по исследованию адекватности и эффективности разработанных в диссертации моделей и методов планирования для проблемно-ориентированных вычислительных сред. В разделе 4.1 описывается методика проведения экспериментов и указываются методы генерации тестовых графов заданий. Раздел 4.2 посвящен экспериментам, подтверждающим адекватность модели вычислительной среды и эффективность алгоритма планирования ресурсами POS на различных графах вычислительных заданий. Приводятся результаты вычислительных экспериментов по сравнению алгоритмов POS, DSC и Min-Min.

4.1. Методика проведения экспериментов

Исследование алгоритма POS, разработанного в диссертационной работе, проводилось на следующих двух классах заданий:

1) многокритериальная оптимизация (МКО);
2) случайные задания (СЗ).

4.1.1. Класс МКО

Класс МКО представляет вычислительные задания по многокритериальной оптимизации, составляющие большой процент загрузки современных суперкомпьютерных и распределенных вычислительных систем. Графы заданий класса МКО имеют следующую структуру: ярусно-параллельная форма состоит из трех ярусов. Первый и третий ярусы параллельной формы содержат по одной вершине. Второй ярус содержит w вершин. Число w указывается при генерации графа. Вершина первого яруса соединена дугами со всеми вершинами второго яруса. Вершина третьего яруса также соединена дугами со всеми вершинами второго яруса. Каждой задаче-вершине

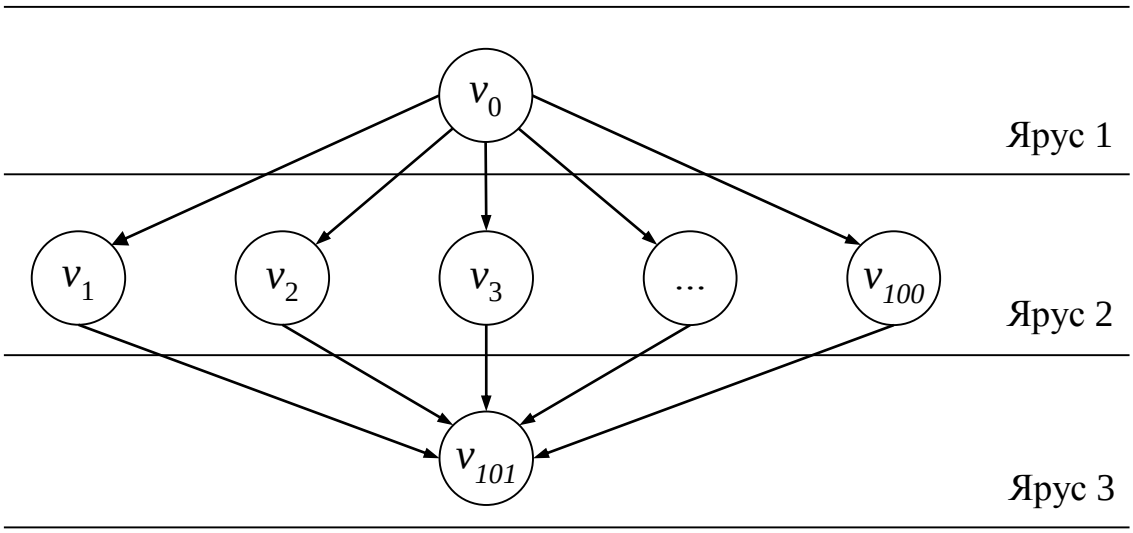

Рис. 21. Пример графа задания класса МКО.

сопоставляется пара чисел: максимальная масштабируемость задачи m_v и время выполнения задачи на одном процессорном ядре t_v. Числа m_v и t_v являются константами и имеют для всех задач одинаковые значения. Аналогично, каждой дуге сопоставлялась константа δ – объем передаваемых данных, одинаковая для всех дуг. На рис. 21 приведен пример графа задания класса МКО для ширины $w = 100$.

4.1.2. Класс СЗ

Класс СЗ представляет *случайные* задания с различным числом вершин и дуг. Качественной характеристикой графа задания в классе СЗ является отношение T/Δ, где T – среднее время выполнения задачи на одном ядре, Δ – средний вес дуги. По этому параметру в классе СЗ могут быть выделены следующие три важные группы [47].

1. $M1$ – *сбалансированные* графы заданий, у которых $T/\Delta \in (0.8,1.2)$. В таких заданиях время на передачу данных сопоставимо с временем вычислений.
2. $M2$ – *крупнозернистые* графы заданий, у которых $T/\Delta \in (3,10)$. В таких заданиях большая часть времени тратится на вычисления.

3. $МЗ$ – *мелкозернистые* графы заданий, у которых $T/\Delta \in (0.1, 0.3)$. В таких заданиях значительное время затрачивается на передачу данных, а вычисления требуют незначительного времени.

4.2. Результаты экспериментов

4.2.1. Плотность расписания

В первой серии экспериментов исследовалась *плотность* расписания, генерируемого алгоритмом POS. Под плотностью здесь понимается величина, обратная количеству вычислительных узлов, задействованных для выполнения задания.

Сначала плотность расписания была исследована для задач класса МКО. Эксперименты проводились для трех значений параметра w, определяющего ширину графа задания: 100, 200, 300. Для всех вершин и дуг графа использовались одинаковые значения весов и маркировки, приведенные в табл. 14.

Табл. 14. Параметры для построения графов заданий класса МКО

Параметр	Семантика	Значение
m_v	Масштабируемость задач	10
t_v	Время выполнения задачи на 1 ядре	100
δ	Объем данных, передаваемых по дуге	50

Результаты экспериментов приведены на рис. 22 в виде зависимости количества задействованных вычислительных узлов от количества процессорных ядер в одном вычислительном узле. Графики показывают, что при увеличении количества ядер в вычислительном узле плотность расписания для заданий МКО возрастает и стремится к максимальному значению 1 во всех рассмотренных случаях. При этом плотность расписания существенно возрастает при увеличении ширины графа задания.

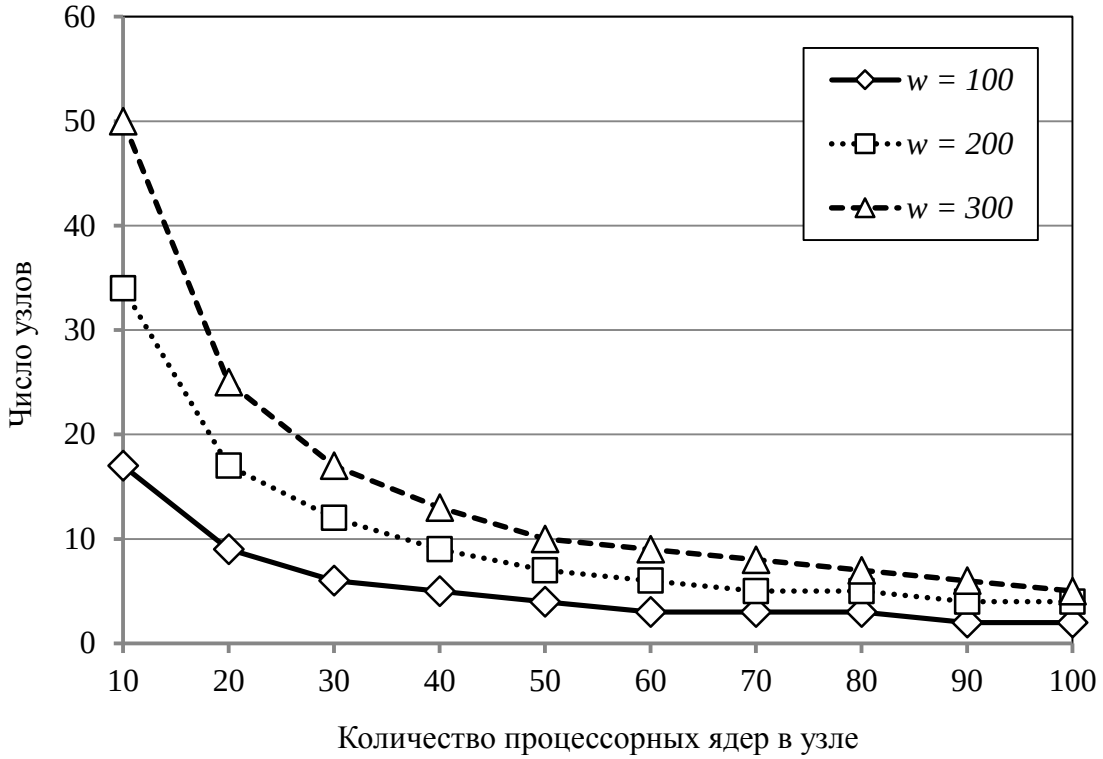

Рис. 22. Зависимость количества задействованных вычислительных узлов от количества процессорных ядер в вычислительном кластере для задания МКО.

Затем плотность расписания была исследована для задач класса С3. Эксперименты проводились для трех значений параметра w, определяющего ширину графа задания: 30, 50, 70. Для всех вершин и дуг графа использовались одинаковые значения весов и маркировки, приведенные в табл. 15.

Табл. 15. Параметры для построения графов заданий класса С3

Параметр	Семантика	Значение
l	Высота графа задания	10
m_v	Масштабируемость задач	10
t_v	Время выполнения задачи на 1 ядре	100
δ	Объем данных, передаваемых по дуге	50

Результаты экспериментов приведены на рис. 23. Графики показывают, что при увеличении количества ядер в вычислительном узле плотность расписания для заданий С3 также возрастает и стремится к максимальному значению 1 во всех рассмотренных случаях. При этом плотность расписания существенно возрастает при увеличении ширины графа задания.

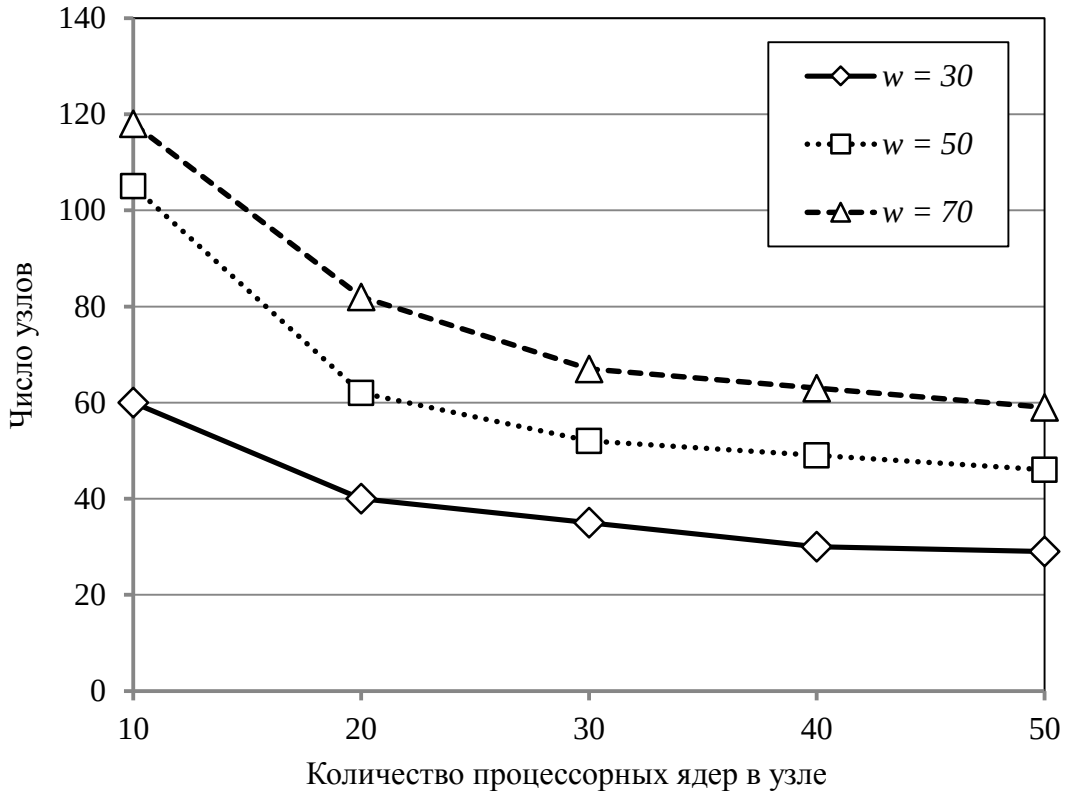

Рис. 23. Зависимость количества задействованных вычислительных узлов от количества процессорных ядер в вычислительном кластере для класса С3.

4.2.2. Сравнительный анализ POS с другими алгоритмами

Во второй серии экспериментов была исследована эффективность алгоритма POS в сравнении с двумя другими алгоритмами планирования: DSC [96] и Min-Min [63]. Для этого были подготовлены три группы графов заданий $M1$, $M2$ и $M3$ с параметрами, приведенными в табл. 16.

Табл. 16. Параметры построения графов заданий

Параметр	Семантика	Значение
m_v	Масштабируемость задачи	20
t_v	Время выполнения задачи на 1 ядре	40
δ_{M1}	Средний вес дуги для группы $M1$	40
δ_{M2}	Средний вес дуги для группы $M2$	10
δ_{M3}	Средний вес дуги для группы $M3$	400
d	Число ядер на вычислительном узле	100

В каждой группе варьировались высота l и ширина w графа. Результаты экспериментов представлены в табл. 17, табл. 18 и табл. 19. Для сравнения алгоритма POS с алгоритмами DSC и Min-Min вычислялась *относительная эффективность* $\left(1 - \frac{T_{pos}}{T}\right) \cdot 100\%$, где T_{pos} – время выполнения задания для алгоритма POS, T – время выполнения задания алгоритмом DSC или Min-Min соответственно. Во всех случаях алгоритм POS, разработанный в диссертационной работе, демонстрирует существенное ускорение времени выполнения задания по сравнению с алгоритмами DSC и Min-Min. При этом алгоритм POS существенно превосходит DSC по плотности получаемого расписания и показывает результаты, сравнимые по этому показателю с алгоритмом Min-Min.

Табл. 17. Сравнение алгоритмов POS, DSC и Min-Min для группы *M*1

№ п/п	l	w	$\|V\|$	$\|E\|$	Количество задействованных вычислительных узлов			Относительная эффективность	
					POS	DSC	MinMin	POS/DSC	POS/MinMin
1	5	10-20	51	126	7	21	4	73,57%	88,72%
2	10	10-20	117	296	4	36	4	72,76%	89,51%
3	10	20-30	211	505	16	68	6	73,45%	88,04%
4	20	5-10	137	252	2	46	2	90,18%	94,36%
Среднее значение								77,49%	90,16%

Табл. 18. Сравнение алгоритмов POS, DSC и Min-Min для группы *M*2

№ п/п	l	w	$\|V\|$	$\|E\|$	Количество задействованных вычислительных узлов			Относительная эффективность	
					POS	DSC	MinMin	POS/DSC	POS/MinMin
1	5	10-20	49	113	3	25	4	85,00%	82,35%
2	10	10-20	130	390	12	42	4	79,14%	60,16%
3	10	20-30	206	464	17	73	6	75,32%	71,14%
4	20	5-10	141	290	13	41	2	79,14%	32,17%
Среднее значение								79,65%	61,46%

Табл. 19. Сравнение алгоритмов POS, DSC и Min-Min для группы М3

№ п/п	l	w	$\|V\|$	$\|E\|$	Количество задействованных вычислительных узлов			Относительная эффективность	
					POS	DSC	MinMin	POS/DSC	POS/MinMin
1	5	10-20	59	190	4	21	4	77,80%	92,40%
2	10	10-20	123	378	3	34	4	79,55%	97,64%
3	10	20-30	201	453	9	61	6	51,36%	90,72%
4	20	5-10	133	287	2	29	2	68,04%	87,31%
Среднее значение								69,19%	92,02%

4.3. Выводы по главе 4

Проведенные вычислительные эксперименты показывают, что для представительных классах задач МКО и СЗ алгоритм POS генерирует расписания, которые качественно превосходят по эффективности расписания, генерируемые известными алгоритмами планирования DSC и Min-Min. Это достигается за счет того, что алгоритм POS анализирует все зависимости задач по данным, как и алгоритм DSC, и обеспечивает возможность распределения задач по процессорным ядрам, подобно алгоритму Min-Min.

ЗАКЛЮЧЕНИЕ

В диссертационной работе были рассмотрены вопросы, связанные с методами управления ресурсами в проблемно-ориентированных распределенных вычислительных средах. Исследованы современные подходы к планированию ресурсов. Приведена общая классификация алгоритмов планирования, выполнен обзор наиболее популярных алгоритмов планирования. Рассмотрены особенности планирования в распределенных вычислительных средах. Разработаны методы для представления проблемно-ориентированных распределенных вычислительных сред, включая модель задания и модель вычислительной системы, а также алгоритм планирования ресурсов POS. Данные методы и алгоритм планирования ресурсов отличаются от других известных методов и алгоритмов планирования для распределенных вычислительных сред, тем, что они учитывают проблемно-ориентированную специфику потоков работ в сложных приложениях и ориентированы на современные многоядерные кластерные архитектуры. На основе предложенных методов и алгоритма планирования ресурсов, разработан брокер ресурсов DiVTB, позволяющий выполнять эффективное планирование ресурсов в проблемно-ориентированных распределенных вычислительных средах. Отлаженный код системы составил 5800 строк на языке Java.

Проведены вычислительные эксперименты для планирования очереди потоков работ при помощи брокера ресурсов DiVTB в распределенной суперкомпьютерной сети Южно-Уральского государственного университета. Проведенные эксперименты подтвердили эффективность предложенных подходов.

Работа выполнена при финансовой поддержке Минобрнауки РФ в рамках ФЦП «Исследования и разработки по приоритетным направлениям развития научно-технологического комплекса России на 2014-2020 годы» (гос. контракт № 14.574.21.0035).

В заключение перечислим основные полученные результаты диссертационной работы, приведем данные о публикациях и апробациях, и рассмотрим направления дальнейших исследований в данной области.

Основные результаты диссертационной работы

На защиту выносятся следующие новые научные результаты.

1. Формальные методы представления проблемно-ориентированных распределенных вычислительных сред.
2. Проблемно-ориентированный алгоритм планирования ресурсов *POS* для заданий с потоковой структурой.
3. Брокер ресурсов *DiVTB* для проблемно-ориентированных распределенных вычислительных сред, реализующий алгоритм планирования *POS*.
4. Вычислительные эксперименты, подтверждающие эффективность предложенных подходов.

Публикации по теме диссертации

Статьи в журналах из перечня ВАК

1. *Радченко Г.И., Соколинский Л.Б., Шамакина А.В.* Модели и методы профилирования и оценки времени выполнения потоков работ в суперкомпьютерных системах // Вычислительные методы и программирование: Новые вычислительные технологии. 2013. Т. 14, Вып. 4. С. 96-103.
2. *Шамакина А.В.* CAEBeans Broker: брокер ресурсов системы CAEBeans // Вестник ЮУрГУ. Серия "Математическое моделирование и программирование". 2010. № 16(192). С. 107-115.
3. *Бухановский А.В., Марьин С.В., Князьков К.В., Сиднев А.А., Жабин С.Н., Баглий А.П., Штейнберг Р.Б., Шамакина А.В., Воеводин В.В., Головченко Е.Н., Фалалеев Р.Т., Духанов А.В., Тарасов А.А., Шамардин Л.В., Моисеенко А.И.* Результаты реализации проекта "Мобильность моло-

дых ученых" в 2010 году: развитие функциональных элементов технологии IPSE и расширение состава прикладных сервисов // Известия высших учебных заведений. Приборостроение. 2011. Т. 54, № 10. С. 80-86.

4. *Московский А.А., Перминов М.П., Соколинский Л.Б., Черепенников В.В., Шамакина А.В.* Исследование производительности суперкомпьютеров семейства "СКИФ Аврора" на индустриальных задачах // Вестник ЮУрГУ. Серия "Математическое моделирование и программирование". 2010. № 35(211). С. 66-78.

5. *Шамакина А.В.* Организация брокера ресурсов в системе CAEBeans // Вестник Южно-Уральского государственного университета. Серия "Математическое моделирование и программирование". 2008. № 27(127). Вып. 2. С. 110-116.

Статья, индексированная в SCOPUS

6. *Shamakina A.* Brokering Service for Supporting Problem-Oriented Grid Environments // UNICORE Summit 2012 Proceedings, Forschungszentrum Julich, 2012. P. 67-75.

Статьи в изданиях, индексируемых в РИНЦ

7. *Шамакина А.В.* Обзор технологий распределенных вычислений // Вестник ЮУрГУ. Серия "Вычислительная математика и информатика". 2014. Т. 3, №3. С. 51-85.

8. *Шамакина А.В.* Брокер ресурсов для поддержки проблемно-ориентированных сред // Вестник ЮУрГУ. Серия "Вычислительная математика и информатика". 2012. № 46(305). Вып. 1. С. 88-98.

9. *Шамакина А.В., Соколинский Л.Б.* Формальная модель задания в распределенных вычислительных средах // Параллельные вычислительные технологии (ПаВТ'2014): труды международной научной конференции

(1–3 апреля 2014 г., г. Ростов-на-Дону). Челябинск: Издательский центр ЮУрГУ, 2014. С. 343–354.

10. *Московский А.А., Перминов М.П., Соколинский Л.Б., Черепенников В.В., Шамакина А.В.* Опыт использования суперкомпьютера "СКИФ Аврора" для решения научно-технических задач // Параллельные вычислительные технологии (ПаВТ'2010): Труды международной научной конференции (Уфа, 29 марта - 2 апреля 2010 г.). Челябинск: Издательский центр ЮУрГУ, 2010. С. 258-267.
11. *Шамакина А.В.* CAEBeans Broker: брокер ресурсов системы CAEBeans // Параллельные вычислительные технологии (ПаВТ’2010): Труды международной научной конференции (Уфа, 29 марта - 2 апреля 2010 г.). Челябинск: Издательский центр ЮУрГУ, 2010. С. 643-650.
12. *Радченко Г.И., Соколинский Л.Б., Шамакина А.В.* Разработка компонентно-ориентированных CAEBean-оболочек для пакета ANSYS CFX // Параллельные вычислительные технологии: Труды международной научной конференции (28 января - 1 февраля 2008 г., г. Санкт-Петербург). Челябинск: Изд-во ЮУрГУ. 2008. С. 438-443.

Другие публикации

13. *Шамакина А.В.* Алгоритм планирования ресурсов POS для распределенных проблемно-ориентированных сред // Научный сервис в сети Интернет: многообразие суперкомпьютерных миров: Труды Всероссийск. науч. конф. (22-27 сентября 2014 г., Новороссийск). М.: Изд-во МГУ, 2014. С 124-136.
14. *Шамакина А.В.* Протокол взаимодействия с брокером ресурсов в системе CAEBeans // Научный сервис в сети Интернет: масштабируемость, параллельность, эффективность: Труды Всероссийск. науч.

конф. (21-26 сентября 2009 г., Новороссийск). М.: Изд-во МГУ, 2009. С. 400-402.

15. *Шамакина А.В.* Организация брокера ресурсов в системе CAEBEANS // Научный сервис в сети Интернет: решение больших задач: Труды Всероссийск. науч. конф. (22-27 сентября 2008 г., Новороссийск). М.: Изд-во МГУ, 2008. С. 326-327.

Свидетельства о регистрации программ и баз данных

16. *Шамакина А.В.* Свидетельство Роспатента о государственной регистрации программы для ЭВМ "RaVIS Broker" № 2011610338 от 11.01.2011, правообладатели: ИПС им. А.К. Айламазяна РАН, ГОУ ВПО "ЮУрГУ".

17. *Юрков В.В., Дорохов В.А., Радченко Г.И., Насибулина Р.С., Шамакина А.В.* Свидетельство Роспатента о государственной регистрации программы для ЭВМ "CAEBeans Toolbox: программная среда для разработки проблемно-ориентированных оболочек для грид" № 2008614485 от 03.10.2008, правообладатель: ГОУ ВПО "ЮУрГУ".

18. *Юрков В.В., Дорохов В.А., Радченко Г.И., Насибулина Р.С., Шамакина А.В.* Свидетельство Роспатента о государственной регистрации программы для ЭВМ "CAEBeans Sphere: программное средство для поддержки распределенных вычислительных сред на базе платформы Microsoft.NET" № 2008614486 от 03.10.2008, правообладатель: ГОУ ВПО "ЮУрГУ".

19. *Радченко Г.И., Насибулина Р.С., Шамакина А.В., Юрков В.В., Федянин О.Н., Дорохов В.А.* Свидетельство Роспатента о государственной регистрации программы для ЭВМ "Пакет проблемно-ориентированных оболочек CAEBeans для решения типовых инженерных задач" № 2008611898 от 04.05.2008, правообладатель: ГОУ ВПО "ЮУрГУ".

20. *Радченко Г.И., Шамакина А.В., Худякова Е.С., Репина К.В., Захаров Е.А.* Свидетельство Роспатента о государственной регистрации базы данных "Справочно-библиографический ресурс "Проблемно-ориентированные грид-сервисы в распределенных вычислительных средах" № 2013620973 от 22.08.2013, правообладатель: ФГБОУ ВПО "ЮУрГУ" (НИУ).
21. *Радченко Г.И., Шамакина А.В., Худякова Е.С., Репина К.В.* Свидетельство Роспатента о государственной регистрации базы данных "Справочно-библиографический ресурс "Распределенная обработка данных и облачные вычислительные среды" № 2013621019 от 27.08.2013, правообладатель: ФГБОУ ВПО "ЮУрГУ" (НИУ).
22. *Соколинский Л.Б., Радченко Г.И., Шамакина А.В.* Свидетельство Роспатента о государственной регистрации базы данных "Справочно-библиографический ресурс "Грид-технологии" № 2013620033 от 09.01.2013, правообладатель: ФГБОУ ВПО "ЮУрГУ" (НИУ).
23. *Соколинский Л.Б., Радченко Г.И., Шамакина А.В.* Свидетельство Роспатента о государственной регистрации базы данных "Справочно-библиографический ресурс "Суперкомпьютерные технологии" № 2013620032 от 09.01.2013, правообладатель: ФГБОУ ВПО "ЮУрГУ" (НИУ).

В статье [1] А.В. Шамакиной принадлежит раздел 4, (стр. 100-102). В статье [3] А.В. Шамакиной принадлежит раздел «Расширение состава прикладных сервисов HPC-NASIS», (стр. 83). В статье [4] А.В. Шамакиной принадлежит раздел 3, (стр. 72-74). В статье [10] А.В. Шамакиной принадлежит раздел 3, (стр. 73). В статье [12] А.В. Шамакиной принадлежит раздел 2, (стр. 439). В работе [9] Л.Б. Соколинскому принадлежит постановка задачи, А.В. Шамакиной принадлежат все полученные результаты.

Апробация работы

Основные положения диссертационной работы, разработанные модели, методы, алгоритмы и результаты вычислительных экспериментов докладывались автором на следующих международных и всероссийских научных конференциях:

- на Международной научной конференции ISC’2014 (22-26 июня 2014 г., Лейпциг, Германия);
- на Международной научной конференции ISC’2013 (16-20 июня 2013 г., Лейпциг, Германия);
- на Международной научной конференции «Параллельные вычислительные технологии 2014» (1–3 апреля 2014 г., Ростов-на-Дону);
- на Международной научной конференции «Параллельные вычислительные технологии 2010» (29 марта–2 апреля 2010 г., Уфа);
- на Всероссийской научной конференции «Научный сервис в сети Интернет 2009: масштабируемость, параллельность, эффективность» (21–26 сентября 2009 г., Новороссийск);
- на Всероссийской научной конференции «Научный сервис в сети Интернет 2008: решение больших задач» (22–27 сентября 2008 г., Новороссийск).

Направления дальнейших исследований

Теоретические исследования и практические разработки, выполненные в рамках этой диссертационной работы, предполагается продолжить по следующим направлениям.

1. Исследовать возможность изменение количества процессорных ядер, выделяемых отдельной задаче в ходе планирования.
2. Обобщение алгоритма POS на гетерогенные вычислительные среды, в которых различные узлы могут иметь различное количество процессорных ядер, различные типы ядер и различную скорость передачи данных между вычислительными узлами.

ЛИТЕРАТУРА

1. *Воеводин В.В.* Математические модели и методы в параллельных процессах. Наука, 1986. 296 стр.
2. *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. Санкт-Петербург: БХВ-Петербург, 2002. 608 с.
3. *Буч Г., Рамбо Д., Якобсон И.* Язык UML. Руководство пользователя. 2-е изд. М.: ДМК Пресс, 2007. 496 с.
4. ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. URL: http://cert.obninsk.ru/gost/282/282.html (дата обращения: 23.04.2014).
5. *Мелехина О., Новаковский Г., Фролов Д.* Новые возможности ANSYS 13.0 // САПР и графика. 2011. № 4. С. 54-58.
6. *Кнут Д.Э.* Искусство программирования, т. 1. Основные алгоритмы, 3-е изд. М.: Издательский дом «Вильямс», 2000. 720 с.
7. Функциональная блок-схема в Microsoft Office Visio 2007. URL: http://office.microsoft.com/ru-ru/visio-help/HP001207677.aspx (дата обращения: 23.04.2014).
8. ANSYS CFX-Pre User's Guide 13.0. URL: http://www1.ansys.com/customer/content/documentation/130/cfx_pre.pdf (дата обращения: 28.12.2013).
9. ANSYS CFD-Post User's Guide 13.0. URL: http://www1.ansys.com/customer/content/documentation/130/cfx_post.pdf (дата обращения: 18.01.2013).
10. ANSYS CFX Solver User's Guide 13.0. URL: http://www1.ansys.com/customer/content/documentation/130/cfx_solv.pdf (дата обращения: 18.01.2013).

11. ANSYS Design Modeler User's Guide 13.0. URL: http://www1.ansys.com/customer/content/documentation/130/wb_dm.pdf (дата обращения: 18.01.2013).

12. ANSYS CFX-Mesh User's Guide 13.0. URL: http://www1.ansys.com/customer/content/documentation/130/wb_cm.pdf (дата обращения: 18.01.2013).

13. *Abraham A., Buyya R. and Nath B.* Nature's Heuristics for Scheduling Jobs on Computational Grids // Proceedings of 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000). Cochin, India, 2000. P. 45-52.

14. *Aggarwal A. K., Kent R. D.* An Adaptive Generalized Scheduler for Grid Applications // Proceedings of the 19th Annual International Symposium on High Performance Computing Systems and Applications (HPCS'05). Guelph, Ontario Canada, 2005. P. 15-18.

15. *Aggarwal M., Kent R.D., Ngom A.* Genetic Algorithm Based Scheduler for Computational Grids // Proceedings of the 19th Annual International Symposium on High Performance Computing Systems and Applications (HPCS'05). Guelph, Ontario Canada, 2005. P. 209-215.

16. Anderson J.D.Jr. Computational Fluid Dynamics : The Basics With Applications. McGraw-Hill, Inc., 1995. 547 p.

17. *Arora M., Das S.K., Biswas R.* A Decentralized Scheduling and Load Balancing Algorithm for Heterogeneous Grid Environments // Proceedings of International Conference on Parallel Processing Workshops (ICPPW'02). Vancouver, British Columbia Canada, 2002. P. 499-505.

18. *Bajaj R., Agrawal D. P.* Improving Scheduling of Tasks in A Heterogeneous Environment // IEEE Transactions on Parallel and Distributed Systems. 2004. Vol. 15, No. 2. P. 107-118.

19. *Baker M., Buyya R., Laforenza D.* Grids and Grid Technologies for Wide-area Distributed Computing // Journal of Software-Practice & Experience. 2002. Vol. 32, No. 15. P. 1437-1466.

20. Belhajjame K., Vargas-Solar G., Collet C. A flexible workflow model for process-oriented applications // Proceedings of the Second International Conference on Web Information Systems Engineering (WISE'01), December 3-6, 2001, Kyoto, Japan. IEEE Computer Society. Vol. 1, No. 1. P. 72-80.

21. *Berman F.* High-Performance Schedulers // Foster I., Kesselman C. chapter in The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann edited by Publishers, 1999. P. 279-309.

22. *Berman F., Wolski R. et al.* Adaptive Computing on the Grid Using AppLeS // IEEE Transactions on Parallel and Distributed Systems. 2003. Vol. 14, No. 4. P. 369-382.

23. *Berman F., Wolski R. et al.* Application-Level Scheduling on Distributed Heterogeneous Networks. // Proceedings of the ACM/IEEE conference on Supercomputing, Pittsburgh, Pennsylvania USA, 1996. Article No. 39.

24. Business Process Model and Notation. URL: http://www.omg.org/spec/BPMN/ (дата обращения: 23.11.2011).

25. *Braun R., Siegel H. et al.* A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems // Journal of Parallel and Distributed Computing. 2001. Vol. 61, No. 6, P. 810-837.

26. *Buyya R., Abramson D. et al.* Economic Models for Resource Management and Scheduling in Grid Computing // Journal of Concurrency and Computation: Practice and Experience. Vol. 14, Issue 13-15. 2002. P. 1507-1542.

27. *Buyya R., Abramson D., Venugopal S.* The Grid Economy // Proceedings of the IEEE, March 2005, New York, USA. IEEE Press. 2005. Vol. 93, No. 3. P. 698-714.

28. *Buyya R., Giddy J., Abramson D.* An Evaluation of Economy-based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications // Proceedings of the 2nd International Workshop on Active Middleware Services (AMS 2000), Pittsburgh, USA, August 2000. P. 221-230.

29. *Cao J., Jarvis S. A. et al.* GridFlow: Workflow Management for Grid Computing // Proceedings of the 3rd International Symposium on Cluster Computing and the Grid (CCGrid'03), Tokyo, Japan, May 2003. P. 198-205.

30. *Casanova H., Legrand A. et al.* Heuristics for Scheduling Parameter Sweep Applications in Grid Environments // Proceedings of the 9th hetero-geneous Computing Workshop (HCW'00), Cancun, Mexico, May 2000. P. 349-363.

31. *Casavant T., Kuhl J.* A Taxonomy of Scheduling in General-purpose Distributed Computing Systems // IEEE Transactions on Software Engineering. 1988.Vol. 14, No. 2. P. 141-154.

32. *Chapin S. J., Katramatos D. et al.* The Legion Resource Management System // Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '99), in conjunction with the International Parallel and Distributed Processing Symposium (IPDPS '99), Lecture Notes in Computer Science, San Juan, Puerto Rico, April 1999. Vol. 1659. P. 162-178.

33. *Chen H., Maheswaran M.* Distributed Dynamic Scheduling of Composite Tasks on Grid Computing Systems // Proceedings of the 16th International Parallel and Distributed Processing Symposium (IPDPS 2002), Fort Lauderdale, Florida USA, April 2002. P. 88-97.

34. Condor. URL: http://www.cs.wisc.edu/condor (дата обращения: 04.12.2010).

35. *Cooper K., Dasgupta A. et al.* New Grid Scheduling and Rescheduling Methods in the GrADS Project // Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04), Santa Fe, New Mexico USA, April 2004. P. 199-206.

36. *Czajkowski K., Fitzgerald S. et al.* Grid Information Services for Distributed Resource Sharing // Proceedings the 10th IEEE International Symposium on High- Performance Distributed Computing (HPDC-10), San Francisco, California, USA, August 2001. P. 181-194.

37. *Czajkowski K., Foster I. et al.* A Resource Management Architecture for Metacomputing Systems // Proceedings of the 4th Workshop on Job Scheduling Strategies for Parallel Processing, Orlando, Florida USA, March 1998, LNCS. 1998. Vol. 1459. P. 62-82.

38. *Darbha S., Agrawal D.P.* Optimal Scheduling Algorithm for Distributed Memory Machines // IEEE Transactions on Parallel and Distributed Systems. January 1998. Vol. 9, No. 1. P. 87-95.

39. *Deelman E., Blythe J. et al.* Pegasus: Mapping Scientific Workflows onto the Grid // Proceedings of Grid Computing: Second European AcrossGrids Conference (AxGrids 2004), Nicosia, Cyprus, January 2004. P. 11-26.

40. *Dong F, Akl S.G.* Scheduling algorithms for grid computing: State of the art and open problems. Technical Report No. 2006-504, Queen's University, Canada, 2006. P. 55.

41. *El-Rewini H., Lewis T., Ali H.* Task Scheduling in Parallel and Distributed Systems, ISBN: 0130992356, PTR Prentice Hall, 1994.

42. *Ernemann C., Hamscher V., Yahyapour R.* Economic Scheduling in Grid Computing // Proceedings of 8th Workshop on Job Scheduling Strategies for

Parallel Processing, in conjunction with HPDC/GGF 5, Edinburgh, Scotland, UK, July 2002. P. 128-152.

43. *Foster I., Kesselman C.* The Grid. Blueprint for a new computing infrastructure. San Francisco: Morgan Kaufman, 1999. 677 p.

44. *Foster I., Roy A., Sander V.* A Quality of Service Architecture That Combines Resource Reservation and Application Adaptation // Proceedings 8th Int. Workshop on Quality of Service, Pittsburgh, PA, USA, June 2000. P. 181-188.

45. *Foster I., Kesselman C., Tuecke S.* The Anatomy of the Grid: Enabling Scalable Virtual Organizations // International Journal of Supercomputer Applications and High Performance Computing. 2001. Vol. 15, No 3. P. 200-222.

46. *Gehring J., Preiss T.* Scheduling a Metacomputer with Uncooperative Subschedulers // Proceedings of the 5th Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes on Computer Science, San Juan, Puerto Rico, April 1999. Vol. 1659. P. 179–201.

47. Gerasoulis A., Yang T. A comparison of clustering heuristics for scheduling directed acyclic graphs on multiprocessors // Journal of Parallel and Distributed Computing, 1992. Vol. 16, No. 4. P. 276-291.

48. *Hamscher V., Schwiegelshohn U. et al.* Evaluation of Job-Scheduling Strategies for Grid Computing // Proceedings of GRID 2000 GRID 2000, First IEEE/ACM International Workshop, Bangalore, India, December 2000. P. 191-202.

49. *He X., Sun X., Laszewski G.* A QoS Guided Min-Min Heuristic for Grid Task Scheduling // Journal of Computer Science and Technology. July 2003. Special Issue on Grid Computing, Vol.18, No. 4. P. 442-451.

50. *Iverson M., Ozguner F.* Dynamic, Competitive Scheduling of Multiple DAGs in a Distributed Heterogeneous Environment // Proceedings of Seventh Heterogeneous Computing Workshop, Orlando, Florida USA, March 1998. P. 70-78.

51. *James H. A.* Scheduling in Metacomputing Systems, Ph.D. Thesis, The Department Of Computer Science, University of Adelaide, Australia, 1999.

52. *Jennings R.* Cloud Computing with the Windows Azure Platform, 2009. 360 p.

53. *Khokhar A.A., Prasanna V.K. et al.* Heterogeneous Computing: Challenges and Opportunities // IEEE Computer. 1993. Vol. 26, No. 6. P. 18-27.

54. *Kim S.J.* A general approach to multiprocessor scheduling. Report TR-88-04. Department of Computer Science, University of Texas at Austin, 1988.

55. *Kim S.J.*, Browne J.C. A general approach to mapping of parallel computation upon multiprocessor architectures // Proceedings of the International Conference on Parallel Processing, 1988. Vol. 3. P. 1-8.

56. *Kim S., Weissman J.B.* A Genetic Algorithm Based Approach for Scheduling Decomposable Data Grid Applications // Proceedings of the 2004 International Conference on Parallel Processing (ICPP'04), Montreal, Quebec Canada, August 2004. P. 406-413.

57. *Kurowski K., Ludwiczak B. et al.* Improving Grid Level Throughput Using Job Migration And Rescheduling // Scientific Programming. 2004. Vol. 12, No. 4. P. 263-273.

58. *Laszewski G., Foster I. et al.* CoG Kits: A Bridge between Commodity Distributed Computing and High-Performance Grids // Proceedings of the ACM Java Grande 2000 Conference, CA, USA, June 2000. P. 97-106.

59. *Liou J., Palis M.A.* A Comparison of General Approaches to Multiprocessor Scheduling // Proceedings of the 11th International Symposium on Parallel Processing, April 1997. P. 152-156.

60. *Liou J., Palis M.A.* An Efficient Task Clustering Heuristic for Scheduling DAGs on Multiprocessors // Proceedings of Workshop on Resource Management, Symposium of Parallel and Distributed Processing, 1996.

61. *Liu Y.* Survey on Grid Scheduling (for Ph.D Qualifying Exam), Department of Computer Science, University of Iowa, April 2004.

62. *Ma T., Buyya R.* Critical-Path and Priority based Algorithms for Scheduling Workflows with Parameter Sweep Tasks on Global Grids // Proceedings of the 17th International Symposium on Computer Architecture and High Performance Computing, Rio de Janeiro, Brazil, October 2005.

63. *Maheswaran M., Ali S. et al.* Dynamic Matching and Scheduling of a Class of Independent Tasks onto Heterogeneous Computing Systems // Journal of Parallel and Distributed Computing. 1999. Vol. 59, No. 2. P. 107-131.

64. *Marshall, P., Keahey K., Freeman, T.* Improving Utilization of Infrastructure Clouds // Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2011), Newport Beach, CA. May 2011. P. 205-214.

65. *Mateescu G.* Quality of Service on the Grid via Metascheduling with Resource Co-Scheduling and Co-Reservation // International Journal of High Performance Computing Applications. 2003. Vol. 17, No. 3. P. 209-218.

66. *Muthuvelu N., Liu J. et al.* A Dynamic Job Grouping-Based Scheduling for Deploying Applications with Fine-Grained Tasks on Global Grids // Proceedings of the 3rd Australasian Workshop on Grid Computing and e-Research (AusGrid 2005), Newcastle, Australia, 30 January – 4 February, 2005.

67. NIST Special Publication 800-145. A NIST Definition of Cloud Computing. URL: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf (дата обращения: 01.02.2012).

68. OpenPBS. URL: http://www.openpbs.org (дата обращения: 14.12.2010).

69. *Radulescu A., Gemund A.J.C.* On the Complexity of List Scheduling Algorithms for Distributed Memory Systems // Proceedings of 13th International Conference on Supercomputing, Portland, Oregon, USA, November 1999. P. 68-75.

70. *Ranaweera S., Agrawal D.P.* A Task Duplication Based Scheduling Algorithm for Heterogeneous Systems // Proceedings of 14th International Parallel and Distributed Processing Symposium (IPDPS'00), Cancun, Mexico, May 2000. P. 445-450.

71. *Rotithor H. G.* Taxonomy of Dynamic Task Scheduling Schemes in Distributed Computing Systems // Proceedings on Computer and Digital Techniques, January 1994. Vol. 141, No. 1. P. 1-10.

72. *Sabin G., Kettimuthu R. et al.* Scheduling of Parallel Jobs in a Heterogeneous Multi-Site Environment // Proceedings of the 9th International Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes In Computer Science, Washington, U.S.A, June 2003. Vol. 2862.

73. *Sacerdoti F.D., Katz M.J. et al.* Wide area cluster monitoring with Ganglia // Proceedings of IEEE International Conference on Cluster Computing, Hong Kong, December 2003. P. 289-298.

74. *Sakellariou R., Zhao H.* A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems // Proceedings of 18th International Parallel and Distributed Processing Symposium (IPDPS'04), Santa Fe, New Mexico USA, April 2004. P. 111-123.

75. *Sakellariou R., Zhao H.* A Low-cost Rescheduling Policy for Efficient Mapping of Workflows on Grid Systems // Scientific Programming. 2004. Vol. 12, No. 4. P. 253-262.

76. *Sanderson D.* Programming Google App Engine: Build and Run Scalable Web Apps on Google`s Infrastructure (Animal Guide), 2009. 400 p.

77. *Sarkar V.* Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors. The MIT Press, Cambridge, MA, 1989. P. 215.

78. *Schopf J.* Ten Actions When SuperScheduling, document of Scheduling Working Group, Global Grid Forum. URL: http://www.ggf.org/documents/GFD.4.pdf (дата обращения: 13.03.2011).

79. *Shan H., Oliker L. et al.* Scheduling in Heterogeneous Grid Environments: The Effects of Data Migration // Proceedings of ADCOM2004: International Conference on Advanced Computing and Communication, Ahmedabad Gujarat, India, December 2004.

80. *Siegel H. J., Dietz H. G., Antonio J. K.*, Software Support for Heterogeneous Computing // ACM Computing Surveys. 1996. Vol. 28, No. 1. P. 237-239.

81. *Silva D.P., Cirne W., Brasileiro F.V.* Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids // Proceedings of Euro-Par 2003, Klagenfurt, Austria, August 2003. P. 169-180.

82. *Song S., Kwok Y., Hwang K.* Security-Driven Heuristics and A Fast Genetic Algorithm for Trusted Grid Job Scheduling // Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05), Denver, Colorado USA, April 2005. P. 65-74.

83. *Spooner D.P., Cao J. et al.* Localised Workload Management using Performance Prediction and QoS Contracts // Proceedings of the 18th Annual UK

Performance Engineering Workshop (UKPEW' 2002), University of Glasgow, UK, July 2002.

84. *Subramani V., Kettimuthu R. et al.* Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests // Proceedings of 11th IEEE Symposium on High Performance Distributed Computing (HPDC 2002), Edinburgh, Scotland, July 2002. P. 359- 366.

85. *Takefusa A., Matsuoka S. et al.* A Study of Deadline Scheduling for Client-Server Systems on the Computational Grid // Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10'01), San Francisco, California USA, August 2001. P. 406-415.

86. The Globus Toolkit. URL: http://www.globus.org (дата обращения: 04.01.2011).

87. *Topcuoglu H., Hariri S., Wu M.Y.* Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing // IEEE Transactions on Parallel and Distributed Systems. 2002. Vol. 13, No. 3. P. 260-274.

88. *Tsai A., Jiacun Wang Tepfenhart W., Rosea D.* EPC Workflow Model to WIFA Model Conversion // Proceedings of the 2006 IEEE International Conference on Systems, Man, and Cybernetics (SMC'06), Taipei, Taiwan, October, 8-11 2006. P. 2758-2763.

89. ANSYS CFX Tutorials 13.0. URL: http://www1.ansys.com/customer/content/documentation/130/cfx_tutr.pdf (дата обращения: 28.12.2011).

90. *Venugopal S., Buyya R.* A Deadline and Budget Constrained Scheduling Algorithm for eScience Applications on Data Grids // Proceedings of 6th International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP-2005), Melbourne, Australia, Oct. 2-5, 2005. P. 60-72.

91. *Wieczorek M., Prodan R., Fahringer T.* Scheduling of Scientific Workflows in the ASKALON Grid Environment // ACM SIGMOD Record. 2005. Vol. 34, No. 3. P. 56-62.

92. Emmerich W., Aoyama M., Sventek J. The impact of research on the development of middleware technology // ACM Transactions on Software Engineering and Methodology. 2008. Vol. 17, No. 4. P. 19-48.

93. *Wolski R., Spring N.T., Hayes J.* The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing // Future Generation Computing Systems. 1999. Vol. 15, No. 5-6. P. 757-768.

94. *Wright D.* Cheap Cycles from the Desktop to the Dedicated Cluster: Combining Opportunistic and Dedicated Scheduling with Condor // Proceedings of Conference on Linux Clusters: the HPC Revolution, Champaign Urbana, IL USA, June 2001.

95. *Wu M., Shu W., Zhang H.* Segmented Min-Min: A Static Mapping Algorithm for Meta-Tasks on Heterogeneous Computing Systems // Proceedings of the 9th Heterogeneous Computing Workshop (HCW'00), Cancun, Mexico, May 2000. P. 375-385.

96. *Yang T., Gerasoulis A.* DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors // IEEE Transactions on Parallel and Distributed Systems. 1994. Vol. 5, No. 9. P. 951-967.

97. *You S.Y., Kim H.Y. et al.* Task Scheduling Algorithm in GRID Considering Heterogeneous Environment // Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA '04, Nevada, USA, June 2004. P. 240-245.

98. *Young L., McGough S. et al.* Scheduling Architecture and Algorithms within the ICENI Grid Middleware // Proceedings of UK e-Science All Hands Meeting, Nottingham, UK, September 2003. P. 5-12.

99. *Yu J., Buyya R., Tham C.K.* QoS-based Scheduling of Workflow Applications on Service Grids // Proceedings of the 1st IEEE International Conference on e-Science and Grid Computing (e-Science'05), Melbourne, Australia, December 2005.

100. *Zhao H., Sakellariou R.* An Experimental Investigation into the Rank Function of the Heterogeneous Earliest Finish Time Scheduling Algorithm // Proceedings of Euro-Par 2003, Klagenfurt, Austria, August 2003. Springer-Verlag, LNCS 2790. P. 189-194.

101. *Zhu Y.* A Survey on Grid Scheduling Systems / Hong Kong University of science and Technology, Department of Computer Science, 2003.

102. *Zhu Y., Xiao L. et al.* Incentive-based P2P Scheduling in Grid Computing // Proceedings of the 3rd International Conference on Grid and Cooperative Computing (GCC2004), Wuhan, China, October 2004.

103. A. Streit. UNICORE: Getting to the heart of Grid technologies // eStrategies. Vol. 3. 2009. P. 8-9.

104. A. Streit. UNICORE - What lies beneath Grid functionality? // eStrategies. Vol. 7. 2008. P. 38-39.

105. D. Nedelcu, M.-D. Nedeloni, D. Daia. The kinematic and dynamic analysis of the crank mechanism with solidworks motion // Proceedings of the 11th WSEAS international conference on Signal processing, computational geometry and artificial vision, and Proceedings of the 11th WSEAS international conference on Systems theory and scientific computation (GAVTASC'11), Florence, Italy. August 23-25, 2011. P. 245-250.

106. Parasolid XT Format Reference http://www.13thmonkey.org/documentation/CAD/Parasolid-XT-format-reference.pdf [дата обращения 07.02.2012].

107. *Wilson J.M.* Gantt charts: A centenary appreciation // European Journal of Operational Research. 2003. Vol. 149, No. 2. P. 430-437.

ПРИЛОЖЕНИЕ. ОСНОВНЫЕ ОБОЗНАЧЕНИЯ

№ п/п	Обозначение	Значение	Раздел
1.	G	Граф задания	2.2.1
2.	V	Множество вершин графа задания	2.2.1
3.	E	Множество дуг графа задания	2.2.1
4.	$init(e)$	Функция определения начальной вершины дуги: $init: E \to V$	2.2.1
5.	$fin(e)$	Функция определения конечной вершины дуги: $fin: E \to V$	2.2.1
6.	$\delta(e)$	Весовая функция графа, определяющая объем данных для передачи по дуге: $\delta: E \to \mathbb{Z}_{\geq 0}$	2.2.1
7.	m_v	Количество процессорных ядер, на которых задача v имеет линейную масштабируемость	2.2.2
8.	t_v	Время выполнения задачи v на одном ядре	2.2.2
9.	$\gamma(v)$	Функция разметки графа: $\gamma: V \to \mathbb{N}^2$ и $\gamma(v) = (m_v, t_v)$	2.2.1 (2.2.2)
10.	j_v	Количество процессорных ядер, выделенных задаче v	2.2.2
11.	$\chi(v, j_v)$	Вычислительная стоимость задачи v на j ядрах: $\chi(v,j) = \begin{cases} t_v/j_v, & \text{если } 1 \leq j \leq m_v; \\ t_v/m_v, & \text{если } m_v < j_v. \end{cases}$	2.2.2
12.	P	Вычислительный узел: $P = \{c_0, \dots, c_{d-1}\}$	2.2.2
13.	$\mathfrak{P}$	Вычислительная система: $\mathfrak{P} = \{P_0, \dots, P_{k-1}\}$.	2.2.2

14.	$\omega(v)$	Функция кластеризации: $\omega: V \to P$	2.2.2
15.	W_i	Кластер: $W_i = \{v \in V \mid \omega(v) = P_i \in \mathfrak{P}\}$	2.2.2
16.	$\sigma(e)$	Коммуникационная стоимость для дуги: $\sigma(e) = \begin{cases} 0, & \text{если } \omega(init(e)) = \omega(fin(e)); \\ \delta(e), & \text{если } \omega(init(e)) \neq \omega(fin(e)). \end{cases}$	2.2.2
17.	τ_v	Время запуска задачи v	2.2.2
18.	$\xi(v)$	Расписание графа задания: $\xi(v) = (\tau_v, j_v)$	2.2.2
19.	s_v	Время останова задачи v: $s_v = \tau_v + \chi(v, j_v)$	2.2.2
20.	L_i	Ярус ярусно-параллельной формы графа	2.2.2
21.	y	Простой путь: $y = (e_1, e_2, \dots, e_n)$	2.2.2
22.	$u(y)$	Сложность пути	2.2.2
23.	$\bar{y}$	Критический путь: $u(\tilde{y}) = \max_{y \in Y} u(y)$	2.2.2
24.	$\tilde{y}$	Субкритический путь	2.3.3
25.	$\alpha(v)$	План выполнения расписания: $\alpha = \{\alpha_i: W_i \to P_i \mid i = 0, \dots, k-1\}$	2.3.5