РОССИЙСКАЯ АКАДЕМИЯ НАУК
ДАЛЬНЕВОСТОЧНОЕ ОТДЕЛЕНИЕ
ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР

На правах рукописи

Шаповалов Тарас Сергеевич

Планирование выполнения заданий в распределенных вычислительных системах с применением генетических алгоритмов

05.13.11 — математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Диссертация на соискание ученой степени кандидата технических наук

Научный руководитель
чл.-корр. РАН, д.ф.-м.н.
Смагин С. И.

Хабаровск — 2010

Оглавление

Основные сокращения

ВУ — виртуальный узел;

ГА — генетический алгоритм;

ОС — операционная система;

ПО — программное обеспечение;

РВС — распределенная вычислительная система;

СП — сервер планирования;

ЭВМ — электронно-вычислительная машина;

GSI — Grid Security Infrastructure (подсистема безопасности Grid);

JSDL — Job Submission Description Language (язык описания требований задания к ресурсам для его выполнения);

MPI — Message Passing Interface (стандарт передачи сообщений между процессами в вычислительных системах);

OGF — Open Grid Forum (открытое сообщество, созданное для продвижения высокопроизводительных вычислений);

OGSA — Open Grid Services Architecture (стандарт Grid-архитектуры, основанной на web-сервисах);

PBS — Portable Batch System (система пакетного планирования заданий);

XML — eXtensible Markup Language (расширяемый язык разметки).

Основные обозначения

D — размерность задачи планирования запуска заданий.

p_m — вероятность мутации.

p_c — вероятность скрещивания.

T_p — время составления расписания.

T_q — отрезок времени, в течение которого задания поступают для составления нового расписания.

I_m — число поколений изоляции.

S_q — расписание заданий, поступивших в период T_q.

U_q — множество процессов всех заданий, поступивших в период планирования T_q.

N_q — число процессов всех заданий в расписании S_q.

N_p — число процессов одного задания.

M — множество *номеров* всех виртуальных узлов PBC.

G — число всех виртуальных узлов PBC.

Ф — множество всех виртуальных узлов PBC.

ϕ_m — m-ый виртуальный узел PBC.

s_{imq} — слот, зарезервированный для процесса u_{iq} на виртуальном узле ϕ_m отрезок времени.

$|s_{imq}|$ — длина слота s_{imq}.

t_{imq} — момент времени начала слота s_{imq}.

X_q — хромосома, кодирующая расписание S_q.

Введение

Актуальность темы. В различных сферах человеческой деятельности присутствует множество ресурсоёмких задач, требующих интенсивных вычислений. Для их решения широкое применение находят распределенные вычислительные системы (РВС). РВС является обобщенным понятием вычислительной системы, включающей узлы (сайты РВС), объединенные сетью передачи данных. Часто сайты расположены географически удаленно и осуществляют взаимодействие через глобальную сеть передачи данных.

Примерами РВС могут служить системы, интегрирующие вычислительные ресурсы высокой производительности (построенные по технологии *Grid*) [29,85,86]; *Clouds Computing* [65,120]; системы, использующие процессорное время простаивающих рабочих станций [35, 101]. С начала 2000 года в мире введены в эксплуатацию ряд крупных Grid: EGEE[1], NorduGrid[2], Open Science Grid[3], TeraGrid[4] и другие [28]. Ведутся проекты по распределению вычислительной работы между простаивающими рабочими станциями с использованием сети Интернет: *Folder@Home* (исследование сворачивания белков); *GIMPS* (поиск простых чисел Мерсенна); *MilkyWay@Home* (создание трёхмерной динамической модели звёздных потоков в нашей галактике) и другие[5].

Основное назначение РВС — решение ресурсоемких вычислительных задач, для которых производительности имеющихся в распоряжении ЭВМ недостаточно. В результате распределения вычислительной работы по узлам может уменьшаться время расчета и увеличиваться точность решения. Эффективность эксплуатации РВС напрямую зависит от методов планирования выполнения заданий, включающих последовательное нахождение

[1] http://www.eu-egee.org
[2] http://www.nordugrid.org
[3] http://www.opensciencegrid.org
[4] http://www.teragrid.org
[5] http://distributed.ru

связанных расписаний выполнения заданий.

В общем случае, задача составления расписания является NP-полной [13, 95]. Ссылки на исследования NP-полных задач, формулировки которых близки к формулировкам задач составления расписаний, имеются в [107].

Методы составления расписаний. Сталкиваясь с задачами составления расписаний, включающих значительное число заданий и исполнителей, а также большим числом ограничений, точные методы становятся непрактичными. Под ограничениями понимаются правила, определяющие невозможность назначения работ исполнителю. Примерами ограничений при составлении расписания для РВС могут служить: превышение объема оперативной памяти, запрашиваемого для задания на вычислительном узле; различие архитектур доступных и требуемых для выполнения задания процессоров и так далее. Расписания, учитывающие подобные ограничения, носят название ресурсно-зависимых.

Метод критического пути — точный метод составления ресурсно-независимых расписаний для заданий с заранее известными длительностями. Он дает кратчайшую длину расписания при условии неограниченности ресурсов. В классическом варианте данный метод далек от практического применения в РВС. Исследования стохастической [128] и динамической [64] модификаций этого метода, содержащих вероятностные предположения о длительности заданий, делают его более практичным.

Множество задач составления расписаний можно сформулировать в терминах *линейного или целочисленного программирования*, но при условии сильного упрощения постановок задач [131]. Необходимо отметить метод ветвей и границ [116], адаптированный в различных формах для составления расписаний [108, 117, 123]. Данные методы привязаны к целевой функции и специфике ограничений на расписания. В работе [77] показывается, что большинство ограничений, которые содержатся в реальных задачах составления расписаний, плохо подходят для традиционных методов линейного и целочисленного программирования и могут применяться лишь к ограниченному кругу подобных задач.

Метод ограниченного перебора осуществляет поиск в дереве возмож-

ных решений, составленном на основании зависимостей между заданиями. Одним из вариантов его применения для составления расписаний является перебор всех ветвей дерева до тех пор, пока не будет найдено расписание, удовлетворяющее заданному условию. Изменение ограничений в переборных методах требует модификации алгоритма, приводящего ветви дерева в согласованное состояние. К тому же, в крупных задачах дерево может приобрести слишком большой размер.

Эвристические методы также широко используются при решении задач планирования. В этом случае эвристики трактуются как правила планирования. Данный вид эвристик, оперируя множеством заданий, устанавливает порядок их выполнения. Если задание можно сопоставить с различными исполнителями, то эвристика должна их указать. В работе [132] приводится обзор ряда эвристик для задач планирования. В [79] сравниваются восемь эвристик на множестве однотипных задач составления ресурсо-зависимых расписаний. В [80] сравниваются алгоритмы, основанные на эвристиках, с алгоритмами, основанными на методе ограниченного перебора. Результаты сравнения показали, что алгоритмы, основанные на эвристиках, плохо приспособлены для ситуаций, в которых ресурсы сильно ограничены.

Применению эвристических правил в алгоритмах планирования посвящены публикации [66,99,131,139]. Находят применение методы решения задач планирования, использующие *экспертные* и *основанные на знаниях* (knowledge-based) системы [89,103,138,142]. Каждый набор экспертных правил детально прорабатывается под ту или иную задачу, поэтому алгоритмы данной группы являются узкоспециализированными. Системы, основанные на знаниях, обычно делят задачу на подзадачи, решением которых занимаются подпрограммы-агенты. Вопросы динамического планирования, когда расписание может быть составлено повторно уже после запуска заданий, рассматриваются в работе [4].

Также ведутся исследования по составлению расписаний в мультипроцессорных вычислительных средах с помощью решения систем булевых уравнений, описывающих ресурсные ограничения [7]. Из преимуществ данного подхода можно отметить то, что он позволяет учитывать различ-

ные ограничения, предъявляемые к расписанию, а также использовать существующие эффективные решатели булевых уравнений (SAT-решатели). Однако применение SAT-решателей требует приведения булевых функций общего вида к дизъюнктивной нормальной форме, что, как правило, является трудной задачей в классической постановке [6].

При планировании заданий в РВС в основном используются: алгоритм FCFS, списочный алгоритм и алгоритм обратного заполнения. *Алгоритм FCFS* (First Come First Serve) — простой и хорошо изученный алгоритм. Задания выстраиваются в список согласно порядку их поступления. Как только необходимое количество вычислительных ресурсов становится доступным заданию, которое находится в начале списка, оно запускается. Данный метод имеет следующие преимущества: нет необходимости в информации о длительности выполнения заданий, алгоритм прост в реализации и не требует больших вычислительных затрат.

В ряде случаев с помощью данного алгоритма можно получать удовлетворительные расписания [140]. Тем не менее *FCFS* может приводить к появлению большого числа простаивающих процессоров, если в очереди находится множество параллельных заданий с большим числом процессов [159]. Часто на практике *FCFS* дополняется модификацией алгоритма обратного заполнения.

Списочный алгоритм [94] подразумевает составление списка заданий согласно некоторому множеству эвристик планирования. Затем происходит запуск очередного задания из списка, как только необходимое количество ресурсов становится доступным. В работе [100] приводятся результаты исследований нескольких эвристик планирования. Как и *FCFS*, списочные алгоритмы не требуют предварительной информации о длительности выполнения заданий.

Алгоритм обратного заполнения (Backfill) [119] требует информации о длительности выполнения заданий. Применительно к списку заданий, отсортированных по времени поступления, в случае, если очередное задание не может быть запущено ввиду отсутствия необходимых свободных ресурсов, то рассматривается следующее задание из списка, которое сможет использовать простаивающие ресурсы без задержки начала выполнения

предыдущего задания. Алгоритм обладает следующими достоинствами.

- Позволяет составлять расписания для гетерогенных РВС.
- Избегает зависания низкоприоритетных заданий в очередях, гарантируя их запуск.
- Может составлять достаточно плотные по времени расписания.
- Обладает приемлемыми характеристиками скорости работы.

Основные шаги алгоритма обратного заполнения следующие.

1. Выяснение, какие процессоры и на какое время будут свободны.
2. Свободные отрезки времени с одним началом объединяются в окна.
3. Из всех окон выбирается самое широкое.
4. Среди оставшихся заданий выбирается наиболее точно удовлетворяющее данному окну задание и планируется на запуск в начале выбранного окна.

Таким образом, окна, в которых будут спланированы задания с меньшим приоритетом, могут начинаться раньше, чем спланированные ранее более приоритетные задания. В результате обеспечивается более равномерная загрузка ресурсов, чем в случае *FCFS* или списочного алгоритма.

Одной из популярных систем планирования для вычислительных кластеров является Maui. В ее состав входит алгоритм обратного заполнения [110]. Этот алгоритм также задействован во многих других планировщиках [124, 136, 151, 169, 170]. Подробнее с данным алгоритмом можно ознакомиться в работе [34]. Построение архитектуры систем планирования для Grid, а также детальное сравнение основных алгоритмов планирования обсуждаются в работе [165].

Генетические алгоритмы (ГА). Предпосылкой возникновения ГА стали несколько открытий в биологии. В 1859 году Чарльз Дарвин опубликовал работу “Происхождение видов”, где были провозглашены основные принципы эволюционной теории: наследственность (потомки сохраняют свойства родителей), изменчивость (потомки почти всегда не идентичны), естественный отбор (выживают наиболее приспособленные). Тем

самым было показано, какие механизмы могут приводить к эволюционному развитию флоры и фауны. Первые публикации, которые впоследствии привели к идее генетических алгоритмов, принадлежат Нильсу Баричелли (Nils Barricelli). Его работы [59–61] были направлены, прежде всего, на понимание природного феномена наследственности.

Основателем современной теории генетических алгоритмов считается Джон Холланд (John Holland). В его работе 1975 года [104] впервые вводится термин "генетический алгоритм" и предлагается схема классического генетического алгоритма (canonical genetic algorithm). Холланд заложил фундамент для дальнейших исследований в данной области и, в частности, для понимания основных законов функционирования эволюционных алгоритмов.

Важной вехой в развитии теории ГА стала диссертация Алана Де Йонга (Alan De Jong) [81]. В ней проведено исследование различных аспектов ГА с помощью компьютерных экспериментов. Набор тестов для проверки эффективности работы генетических алгоритмов, предложенный Де Йонгом, является множеством задач поиска экстремумов функций от простых унимодальных до мультимодальных. Подобные наборы тестов часто применяются для отладки работы алгоритмов.

В работе [77] Лоренс Девис (Lawrence Davis) впервые предложил использовать ГА для поиска расписаний. В отличие от методов локального поиска [17] и метода имитации отжига [52,111], основанных на манипуляции одним решением, ГА использует популяцию решений, давая им бóльшую устойчивость к преждевременной сходимости к локальным экстремумам. Для усиления данного свойства используются параллельные варианты генетических алгоритмов с несколькими подпопуляциями.

Также находят применение смешанные методы, в которых ГА комбинируется с другими алгоритмами. В работе [141] исследуются сочетания генетического алгоритма с методами, основанными на эвристиках упорядочения. Данный тип эвристик определяет правила формирования списка всех заданий, устанавливающего порядок их выполнения. Использование ГА для усовершенствования эвристик в различных эвристических методах описано в [102]. Ведутся исследования по объединению математического

аппарата нечеткой логики с генетическими алгоритмами для планирования выполнения заданий [93], где с помощью нечеткой логики происходит динамическая настройка значений вероятности применения генетических операторов.

Исследование алгоритмов планирования для РВС является достаточно молодым направлением. Только к концу 70-х — началу 80-х годов XX века сложилось достаточно целостное представление о границах применимости детерминированных методов теории расписаний к планированию параллельных вычислений [14, 21–23, 31, 34, 70, 105, 106, 161].

В книге [1] описываются точные ресурсно-зависимые методы решения задач минимизации длины расписания в вычислительных системах. Подробное изложение вопросов составления расписаний параллельных вычислений дается в работе [5]. Там же рассматривается задача составления расписания с прерываниями и приводится описание алгоритмов решения этих задач. В работе [135] предлагается классификация систем планирования заданий в РВС и формулируются требования, предъявляемые к этим системам.

Также необходимо отметить исследования в области стратегий планирования в РВС — выбор того или иного алгоритма для планирования в зависимости от наступивших событий. В работе [34] предлагается использование семейств стратегий распределения ресурсов, состоящих из опорных планов, и прогнозирования освобождения ресурсов на основе моделирования действий локальных систем планирования.

Помимо минимизации времени выполнения заданий, в работе [54] предлагается минимизировать вероятность нарушений в работе сети и функционировании узлов распределенной вычислительной системы. Такой подход, названный двуцелевым (biobjective), описывает два варианта алгоритма, в каждом из которых используются весовые характеристики для поддержания баланса между временем выполнения и вероятностью сбоев.

Использование ГА для нахождения расписания запуска заданий с учетом задержек на передачу данных между узлами вычислительного кластера обсуждается в [122]. В [55] описан ряд испытаний ГА и различных детерминированных методов составления расписаний. Они показали, что

использование генетических алгоритмов вместо детерминированных способно повысить эффективность эксплуатации вычислительного кластера.

Ведутся исследования по введению принципов экономики при распределении ресурсов в РВС между заданиями [72]. В этом случае определяется стоимость ресурсов и бюджет владельцев заданий. Затем происходит регуляция баланса между стоимостью и предложением. Введение стоимости выполнения задания мотивирует их владельцев “тратить” свой бюджет, подбирая оптимальный для них баланс между временем завершения расчетов и размером бюджета.

Целью диссертационной работы работы является разработка и исследование генетических алгоритмов составления расписания выполнения непараллельных и параллельных заданий в распределенных вычислительных системах, а также разработка и программная реализация алгоритма планирования непрерывно поступающих заданий на основе указанных генетических алгоритмов. В соответствии с целью были поставлены следующие задачи.

1. Разработка и исследование генетических алгоритмов для составления расписаний выполнения непараллельных и параллельных заданий в РВС.

2. Разработка алгоритма планирования выполнения непрерывно поступающих заданий в РВС.

3. Программная реализация системы планирования выполнения заданий в РВС типа Grid.

Объектом исследования служит теория и практика организации распределенных вычислений.

Предметом исследования являются методы планирования заданий в РВС.

Методы исследования. В процессе исследования использовались теория расписаний и методы эволюционного поиска. При решении поставленных задач применялись также методы системного и прикладно-

го программирования; методы объектно-ориентированного программирования; технология построения вычислительных кластеров и Grid.

Структура и объем диссертации. Диссертация состоит из введения, трех глав, заключения, списка литературы из 170 наименований, глоссария и приложений. Объём работы (за исключением глоссария и приложений) составляет 126 страниц в формате машинописного текста, в том числе 36 рисунков и две таблицы.

Краткое содержание работы. Во введении обосновывается актуальность исследуемой проблемы, приводится краткий обзор работ по методам составления расписаний и планированию в РВС. Излагаются предмет, объект и методы исследования, основные цели и пути их достижения, кратко описывается структура диссертационной работы.

Первая глава посвящена алгоритмам планирования заданий в распределенной вычислительной системе. *В первом разделе* главы определяются основные термины распределенных вычислительных систем. Вводятся такие понятия как ресурсы, процессы, виртуальные узлы и т.п. Кратко описываются основные инструментарии построения РВС. *Во втором разделе* рассматривается задача планирования выполнения заданий в РВС, где вводятся такие понятия как задание, политики планирования, целевая функция и т.п. Даётся математическая формулировка задачи планирования выполнения заданий в РВС. *Третий раздел* включает описание алгоритма планирования, основанного на последовательном составлении расписаний с применением генетических алгоритмов.

Вторая глава посвящена алгоритмам составления расписаний для РВС с применением ГА. *Первый раздел* главы содержит описание применения генетических алгоритмов составления расписаний. Указываются основные проблемы, возникающие при применении ГА к решению данной задачи. *Во втором разделе* детально описываются предлагаемые автором генетические алгоритмы для нахождения расписаний выполнения непараллельных и параллельных заданий. Представляются разработанные механизмы учета ресурсных ограничений и зависимостей между заданиями. Предлагается компактная форма представления хромосомы в оперативной памяти ЭВМ. *Третий раздел* включает описание численных эксперимен-

тов, проведенных с целью настройки основных параметров разработанных ГА. В качестве заданий для этих экспериментов используются задачи из набора тестов NAS Parallel Benchmark. В последнем разделе данной главы описываются численные эксперименты, предназначенные для сравнения эффективности составления расписания выполнения параллельных заданий генетическим алгоритмом с эффективностью алгоритма обратного заполнения.

Третья глава посвящена практической реализации алгоритмов планирования выполнения заданий в Grid. *В первом разделе* обсуждаются существующие глобальные и локальные менеджеры ресурсов в Grid и их взаимодействие. *Во втором разделе* описывается программная реализация разработанных в рамках диссертационной работы алгоритмов планирования в Grid. *Третий раздел* содержит результаты численных экспериментов применения разработанной системы планирования для вычислительных ресурсов ВЦ ДВО РАН. Рассматриваются вопросы масштабируемости системы при применении ее к планированию выполнения заданий с данными ресурсами. Проводится сравнение разработанного алгоритма с алгоритмом обратного заполнения на ресурсах ВЦ ДВО РАН.

В заключении диссертации формулируются основные результаты работы, намечаются направления дальнейших исследований.

Приложения включают основные понятия теории генетических алгоритмов, примеры конфигурационных файлов разработанной системы планирования, а также описание программного обеспечения, используемого на ресурсах Вычислительного центра ДВО РАН.

Научная новизна результатов диссертации заключается в разработке алгоритма составления расписания, позволяющего находить более эффективные по различным критериям расписания выполнения заданий в распределенной вычислительной системе по сравнению с широко используемыми в этой области алгоритмами. Разработанные генетические операторы позволяют учесть ресурсные ограничения на расписания, приоритеты и зависимости между заданиями, и при этом ГА показывает линейный рост производительности от размерности задачи составления расписания и числа параллельных подпопуляций. Анализ влияния значений основных

параметров ГА на скорость составления расписаний позволил выявить коридоры оптимальных значений параметров алгоритма.

Практическая значимость. Разработанное программное обеспечение позволяет повысить эффективность использования вычислительных ресурсов Grid. Полученные результаты являются основой для дальнейших исследований моделей эволюционных алгоритмов для планирования заданий в РВС. В процессе работы также разработано программное обеспечение для администрирования вычислительных кластеров, использующееся в Вычислительном центре ДВО РАН.

Личный вклад автора. Автором осуществлены постановка задач составления расписаний и непрерывного планирования выполнения заданий в РВС, разработка и реализация алгоритмов планирования выполнения параллельных и непараллельных заданий в РВС. Автор диссертации является автором системы планирования [40], реализующей разработанные в процессе выполнения диссертационной работы алгоритмы.

Положения, выносимые на защиту.

1. Применение генетических алгоритмов для планирования выполнения заданий в РВС является средством повышения эффективности использования вычислительных ресурсов.

2. Программная реализация разработанного алгоритма может быть использована для планирования выполнения заданий в РВС типа Grid.

Апробация. Результаты работы докладывались и обсуждались на следующих конференциях и семинарах.

1. Всероссийская научная конференция “Научный сервис в сети интернет: многоядерный компьютерный мир. 15 лет РФФИ.” (Новороссийск, 24 сентября – 29 сентября, 2007 г.).

2. Межрегиональная научно-практическая конференция “Информационные и коммуникационные технологии в образовании и научной деятельности” (Хабаровск, 21 мая – 23 мая, 2008 г.).

3. Международная научная конференция “Распределенные вычисления

и Grid-технологии в науке и образовании" (Дубна, 30 июня – 4 июля, 2008).

4. Региональная научная конференция "XXXIII Дальневосточная математическая школа-семинар имени академика Е.В. Золотова" (Владивосток, 29 августа – 4 сентября, 2008 г.).

5. Международная научная конференция "Параллельные вычислительные технологии" (Санкт-Петербург, 28 января – 1 февраля, 2008 г.).

6. Региональная научная конференция "XXXIV Дальневосточная математическая школа-семинар имени академика Е.В. Золотова" (Хабаровск, 25 июня – 30 июня, 2009 г.).

7. Межрегиональная научно-практическая конференция "Информационные и коммуникационные технологии в образовании и научной деятельности" (Хабаровск, 21 сентября – 23 сентября, 2009 г.).

8. Всероссийская "XI конференция молодых ученых по математическому моделированию и информационным технологиям" (Иркутск, 15 марта – 21 марта, 2010 г.).

9. Международная научно-практическая конференция "Суперкомпьютеры: вычислительные и информационные системы" (Хабаровск, 30 июня – 2 июля, 2010 г.).

10. Региональная научная конференция "XXXV Дальневосточная математическая школа-семинар имени академика Е.В. Золотова" (Владивосток, 31 августа – 5 сентября, 2010 г.).

11. International science conference "First Russia and Pacific Conference on Computer Technology and Applications" (Владивосток, 6 сентября – 9 сентября, 2010 г.).

12. Семинары Вычислительного центра ДВО РАН.

Публикации. По материалам диссертации опубликовано 19 печатных работ [26, 27, 30, 33, 37–50, 145], в том числе две статьи [30, 37] в

журналах, рекомендованных ВАК для опубликования научных результатов диссертации на соискание ученой степени доктора или кандидата наук. Из совместных работ с Пересветовым В.В., Сапроновым А.Ю., Смагиным С.И., Тарасовым А.Г. и Щербой С.И. в диссертацию включены только те результаты, которые получены автором лично.

Автор благодарит Пересветова В.В. и Смагина С.И. за руководство и помощь в процессе выполнения диссертационной работы.

Глава 1. Планирование выполнения заданий в распределенных вычислительных системах

Глава посвящена планированию выполнения заданий в РВС. Приводится описание концепции РВС, описываются алгоритмы планирования заданий для РВС. Даётся математическая постановка задачи планирования выполнения заданий в РВС. В основе алгоритма планирования лежит генетические алгоритмы, детальное описание которых приводится во второй главе диссертации.

1.1. Распределенные вычислительные системы

Термин *распределенные вычисления* или *метавычисления* (metacomputing) был введен в 1987 году Ларри Смааром (Larry Smarr) и Чарльзом Кэтлеттом (Charles Catlett) [155]. В основе данного подхода лежит интеграция вычислительных машин посредством сети передачи данных и специализированного системного программного обеспечения в распределенную вычислительную систему. Назначение такой системы — организация единообразного доступа пользователей к ее ресурсам.

Ресурсы — совокупность программных и аппаратных средств для выполнения процессов. Примеры ресурсов: процессор, среда передачи данных, прикладное программного обеспечение, система хранения данных и т.п. Под *процессом* понимается поток инструкций для процессора ЭВМ с единым адресным пространством, значениями регистров процессора, стеком, открытыми файлами, глобальными переменными и т.п.

Так как процессоры на узлах РВС часто управляют отдельными ресурсами (либо связанными только с этими процессорами, либо разделяемыми несколькими процессорами), то целесообразно рассматривать обобщенное понятие виртуального узла (ВУ). Таким образом, $ВУ$ — один вычисли-

тельный элемент (ядро процессора), рассматриваемый в связке с сопоставленными ему ресурсами. Если несколько процессоров используют одни и те же ресурсы, характеризуемые объемом (оперативная память, постоянная память и т. п.), то будем считать, что объем этих ресурсов равномерно делится между всеми виртуальными узлами, соответствующими данным процессорам. На рис. 1.1 схематически изображено соотношение ВУ и ресурсов в одном вычислительном узле.

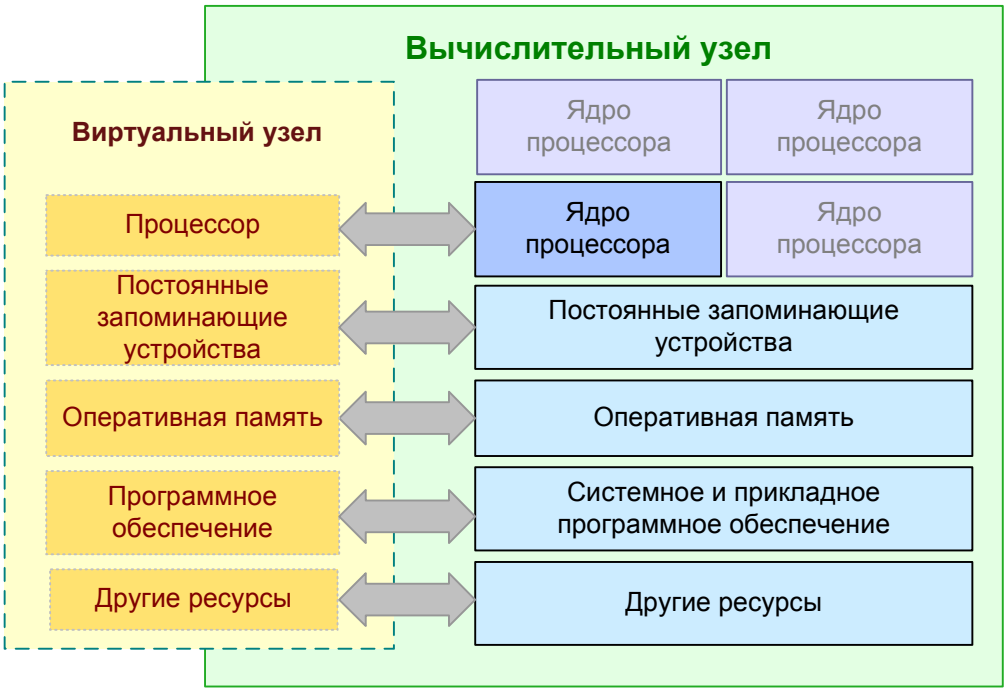

Рис. 1.1. Схема соотношения ресурсов виртуального и физического узлов распределенной вычислительной системы

Словосочетанием “Другие ресурсы” на рис. 1.1 обозначены дополнительные возможности вычислительных узлов. Например, наличие графических или сигнальных процессоров, специализированных электронных плат ускорения вычислений.

В РВС могут включаться вычислительные системы различной архитектуры. Описание особенностей современных вычислительных архитектур имеется в [2, 15, 16, 20, 32]. Особенности архитектуры отечественных семейств современных высокопроизводительных систем изложены в [15, 20].

Наиболее распространенными реализациями РВС являются системы, построенные по технологии Grid [9, 85, 86]. Определение термина Grid на данный момент не является устоявшимся. Будем придерживаться определения Яна Фостера (Ian Foster) и Карла Кесселмана (Carl Kesselman) [87]: "Grid — согласованная, открытая и стандартизованная среда, которая обеспечивает гибкое, безопасное и скоординированное разделение ресурсов в рамках виртуальной организации". Под виртуальной организацией понимается объединение пользователей и реальных организаций, которым доступно некоторое подмножество ресурсов в Grid. Целью введения такого объединения является необходимость разделения полномочий в Grid между пользователями.

В большинстве случаев в качестве сайтов Grid выступают вычислительные кластеры [71], расположенные удаленно и отличающиеся по своим характеристикам. В общем случае, *вычислительный кластер* — параллельная масштабируемая вычислительная система, включающая набор высокопроизводительных компьютеров (процессорных узлов кластера), объединенных коммуникационными сетями и управляемых единым диспетчером заданий [3, 34, 62]. При интеграции процессорных узлов в единый кластер межузловые коммуникации осуществляются посредством передачи сообщений между процессами заданий.

Помимо вычислительных кластеров в Grid могут быть включены и другие типы сайтов. Например, рабочие станции визуализации или сетевые хранилища данных. Термин Grid основывается на метафоре электрической сети (electrical power grid), которая предоставляет пользователям электричество по запросу без необходимости обладания знаниями — где и как оно генерируется.

Доступ к ресурсам в Grid обычно осуществлятся посредством *служб* [36], присутствующих на каждом из сайтов. Службы должны быть стандартизованы в рамках Grid, что позволяет единообразно получать доступ ко всем ресурсам, независимо от его вида и локальных политик на сайте. Стандарт архитектуры построения Grid сервисов OGSA[1] (Open Grid Services Architecture) унифицирует создание, наименование и взаимодей-

[1] http://forge.gridforum.org/sf/projects/ogsa-wg

ствие служб.

Фактическим стандартом построения Grid на базе архитектуры OGSA стал инструментарий Globus Toolkit[2]. Данный программный комплекс включает в себя службы управления заданиями (для мониторинга и координации удаленного выполнения заданий), службы сбора данных (основанные на использовании протокола *LDAP*), службы обеспечения безопасности *GSI* (Grid Security Infrastructure, компоненты которого основаны на технологии электронно-цифровых сертификатов стандарта *X.509*) и службы управления данными (позволяют пользователям получать доступ, передавать и управлять распределенными данными).

Компоненты среды выполнения инструментария Globus Toolkit включают внешние программные интерфейсы (API и SDK языков программирования *Java, Python* и C++), среду выполнения как стандартных, так и пользовательских служб. Для работы с сертификатами применяется, входящий в состав инструментария, пакет программ GSI-OpenSSH — модифицированная версия OpenSSH. Разработанная организацией National Center for Supercomputing Applications (NCSA), она может использоваться для доступа к удаленным системам и для передачи файлов без ввода пароля: все операции аутентифицируются сертификатами подсистемы GSI. Для этих целей используются модификации классических утилит sftp, scp и ssh: gsisftp, gsiscp и gsissh соответственно.

Помимо Globus Toolkit можно отметить и другие подобные ему инструментарии: gLite[3] [10], используемый в международном проекте EGEE и Legion[4] [98]. Последний представляет собой распределенную вычислительную платформу для построения Grid. Данный пакет системного программного обеспечения обладает практически всеми функциями, что и Globus Toolkit. В отличие от Globus Toolkit и Legion пакет gLite является законченным решением для Grid, объединяя как базовые компоненты системного уровня, так и ряд служб высокого уровня (например, подсистему планирования заданий и распределенную файловую систему). Инструментарий gLite включает в себя ряд компонентов Globus Toolkit. Отметим менее рас-

[2]http://www.globus.org/toolkit
[3]http://www.glite.org
[4]http://legion.virginia.edu

пространенные платформы для организации Grid, основанные на Globus Toolkit: *ARC* [84], *AliEn* [73], *DataGrid*[5] и *Unicore*[6].

Запускаемые в РВС приложения можно разделить на три класса: распределенные, параллельные и непараллельные.

Распределенные Grid приложения (в англоязычной литературе: multi-site Grid applications) — класс приложений, которые могут быть запущены на множестве сайтов Grid. Другими словами, процессы одного распределенного приложения могут быть запущены на нескольких вычислительных кластерах одновременно. Теоретическое рассмотрение подобного класса приложений проведено в работе [67].

Распределенные приложения практически не находят применения в Grid ввиду отсутствия поддержки со стороны Grid-инструментариев. Основная сложность здесь состоит в том, что вычислительные кластеры контролируются своими (локальными) системами управления ресурсами. Таким образом, контроль всех процессов распределенного между несколькими кластерами задания затрудняется.

Другой причиной малой распространенности данного класса приложений является то, что разница между скоростными характеристиками сети передачи данных между узлами одного кластера и между несколькими кластерами может сильно различаться. Учет данной характеристики сетей передачи данных усложняет алгоритм распределенного приложения.

Также препятствием к развитию сферы распределенных приложений в некоторых случаях может стать разнородность архитектуры процессоров в составе сайтов Grid. При запуске приложения полезна компиляция исходного кода под заданную архитектуру процессора уже после составления расписания и запуска задания на узле Grid. Такая компиляция в ряде случаев позволяет значительно ускорить приложение за счет оптимизации бинарного кода под данную архитектуру процессора. Очевидно, что в этом случае распределенное приложение должно компилироваться и запускаться в нескольких различающихся копиях, что усложняет конструирование подобного рода приложений.

[5] http://www.datagrid.org
[6] http://www.unicore.org

В подавляющем большинстве случаев имеется возможность провести декомпозицию распределенного алгоритма на несколько параллельных или непараллельных. После завершения вычислений множества таких заданий, соответствующих одной распределенной задаче, полученные данные собираются и обрабатываются отдельно.

Параллельные приложения относятся к другому классу. В отличие от распределенных, приложения данного класса расчитаны для выполнения на однородном вычислительном кластере или многопроцессорной системе с общей памятью.

Параллельной будем называть программу с числом процессов $N > 1$, одновременно выполняемых на N однородных процессорах. Процессы параллельной программы потенциально могут обмениваться данными между собой как по сети передачи данных, так и используя механизмы взаимодействия процессов в рамках одной операционной системы (IPC, Inter-Process Communication). Все процессы параллельной программы должны быть спланированы на одновременный запуск. Такой вид планирования называется совместным (синонимы: связанный, комплектный, gang-scheduling).

В общем случае *параллельное задание* — это указание пользователя запустить одну параллельную программу с определенными требованиями к ресурсам, передав ей все необходимые для расчета файлы данных. В планировании параллельных заданий для Grid важным является то, что процессы одного приложения не выходят за рамки одной группы ВУ — сайта Grid. Примером параллельного задания может служить выполнение MPI-приложения на узлах вычислительнного кластера.

Непараллельной будем называть программу, включающую лишь один вычислительный процесс. *Непараллельным заданием* будет являться указание пользователя выполнить одну или несколько непараллельных программ, между которыми отсутствуют взаимодействия. Часто такое задание включает запуск множества копий одной программы с различными входными параметрами.

В непараллельных приложениях часто используется управляющая программа, собирающая результаты расчетов и выдающая новую порцию входных данных. Такой вид архитектуры носит название Поставщик-

Потребитель [162]. Изолированные вычислительные процессы часто облегчают разработку и поддержку приложений. Примерами алгоритмов с изолированными процессами могут служить *Map-Reduce* [82] и семейство алгоритмов k-средних [63], которые находят применение в методах анализа данных, математической статистике и других областях знаний.

1.2. Задача планирования выполнения заданий в распределенной вычислительной системе

В разделе осуществляется постановка задачи планирования выполнения непараллельных и параллельных заданий в РВС. Описывается разработанный алгоритм планирования, основанный на составлении расписаний генетическими алгоритмами.

Важной задачей при организации РВС является планирование выполнения заданий. Процесс планирования включает резервирование ресурсов для задания, для чего необходимо найти подходящие ресурсы, выбрать необходимое их количество и найти оптимальное время запуска соответствующего приложения. Если запрашиваемые заданиями ресурсы превышают имеющиеся, это ведет к конфликтам, которые должны быть разрешены алгоритмом планирования.

Система планирования в Grid должна быть готова принимать поток заданий, генерировать расписания и в соответствии с ними ставить задания на выполнение. Задания могут поступать для планирования одновременно. При постановке задания в очередь требуется файл описания этого задания (синоним: паспорт задания). Различные системы планирования могут работать с разными форматами файлов описания.

Описание задания обычно включает информацию следующих категорий:

- Информация о пользователе. Она может быть использована для определения приоритета задания и доступных для пользователя ресурсов. Часто информация о пользователе поступает в систему планирования исходя из данных о соответствующей пользователю системной учетной записи.

- Требования задания к ресурсам. Эти данные конкретизируют ресурсы, запрашиваемые заданием пользователя. Часто данная категория включает необходимое число и тип процессоров, объем оперативной памяти, а также другие типы ресурсов аппаратного и программного обеспечения, необходимые для выполнения задания. Часть информации данной категории может быть определена приближенно, как, например, длительность выполнения задания.

- Цели планирования. Информация данной категории в некоторых случаях помогает алгоритму составления расписания находить лучшее расписание. К примеру, пользователь может указать, что ему нужен результат выполнения задания к определенному сроку, таким образом подсказывая, что данные ему не понадобятся ранее. В случае, если пользователи получают платный доступ к ресурсам, в данной категории может указываться, что пользователь готов заплатить за ускорение расчета.

Для некоторых систем планирования важной для составления расписания информацией являются требуемое количество ресурсов и предполагаемая длительность выполнения. Так, алгоритм обратного заполнения использует длительности выполнения заданий для определения порядка их запуска. В процессе составления расписания система планирования может масштабировать значение предположительной длительности выполнения в соответствии с характеристиками процессоров, на которых задание будет запущено. Для масштабирования данного значения системой планирования, пользователь должен указать параметры процессоров, для которых было получено значение длительности выполнения.

Также в файле описания задания может быть указана дополнительная информация, такая как имя задания, имя файла стандартного вывода и ошибок, расположение файлов со входными данными. Может быть указано явное требование задания запускать все его вычислительные процессы одновременно.

При разработке и исследовании алгоритмов в данной работе предполагалось, что после запуска задание не будет прервано. Также предполага-

лось, что миграции заданий с одних ресурсов на другие после их запуска запрещены. Требования задания к ресурсам также фиксируются и не могут быть изменены после того, как задание было спланировано на запуск. Альтернативная модель задания может предполагать, что задания приспособлены к перезапуску и самостоятельно (либо средствами операционной системы) способны восстановить свое состояние, сохраненное на момент прерывания.

Существует множество форматов описания заданий, но стандартным считается формат *JSDL* (Job Submission Description Language), а также его расширение для высокопроизводительных систем — *JSDL-WG*. Стандарты разработаны некоммерческой организацией OGF[7] (Open Grid Forum), представляющей сообщество пользователей, разработчиков программного обеспечения и организаций, участвующих в непосредственном развитии и внедрении в различных сферах деятельности высокопроизводительных вычислений. Далее под стандартом JSDL-WS подразумевается стандарт версии 1.0, основанный на языке описания данных XML. Преимуществом данного формата является то, что как структуру, так и типы данных в XML-документе можно проверить на соответствие заданной схеме. Также XML данные сравнительно легко могут восприниматься и изменяться человеком.

В общем случае система планирования включает [113] следующие компоненты.

- Политики планирования.
- Целевая функция.
- Алгоритм составления расписания.

Политики планирования — это правила, устанавливающие очередность выделения ресурсов заданиям. Часто ресурсов оказывается недостаточно для удовлетворения потребностей всех заданий немедленно. В таких случаях политика планирования призвана разрешить данный конфликт. При планировании заданий в PBC будем придерживаться правила приоритетов: процессы, принадлежащие заданиям с приоритетом выше должны

[7] http://www.ogf.org

быть запущены не позже процессов с приоритетом ниже, если это возможно.

Для оценки расписания необходим метод, обеспечивающий однозначное их сравнение. *Целевая функция* (в ГА — функция пригодности) — функция оценки расписания по заданному критерию. Таким образом, имеется возможность ранжировать расписания по их эффективности. Часто целевая функция возвращает для конкретного расписания числовое значение в диапазоне $(0, 1]$, где значение 1 достигается в случае оптимального расписания. Целевая функция может содержать в себе как один, так и несколько критериев [154]. В основе целевой функции может лежать длина расписания, среднее время выполнения всех заданий, среднее время ожидания запуска заданий, загруженность ресурсов и другие [91].

Алгоритм составления расписания — ключевой компонент системы планирования, который на основании информации о ресурсах и заданиях, политиках планирования и целевой функции строит непротиворечивое расписание. Основными свойствами указанного алгоритма являются: эффективность получаемых расписаний; время составления расписания; количество вычислительных ресурсов, необходимое самому алгоритму для составления одного расписания.

Математическая формулировка задачи планирования. Разделим отрезок времени T, в течении которого поступают задания пользователей, на $Q + 1$ последовательных отрезков (периодов) планирования: $T = \bigcup T_q$, где q — порядковый номер периода планирования, $q = 0, 1, \ldots, Q$. В каждый из периодов T_q накапливается некоторое количество заданий. Каждое задание включает в себя запрос на выполнение одного или нескольких процессов.

Пусть $U_q = \{u_{iq} : \ i = 1, 2, \ldots, N_q\}$ — множество процессов, принадлежащих заданиям, планируемым в период T_q, где N_q — число процессов, принадлежащих всем заданиям, планируемым в период T_q. Для выполнения задания пользователь заказывает количество требуемых ресурсов определенных видов. Ресурсами являются: тип и версия операционной системы, архитектура и серия процессора, частота процессора, оперативная память и так далее. Если пользователь не указал для задания требование

к ресурсу определенного вида, то для данного вида ресурса заказывается значение по умолчанию, заданное администратором РВС.

Пусть $\Phi = \{\phi_m : \ m \in M\}$ — неизменяемое в течении всего отрезка времени T множество всех виртуальных узлов в РВС, где M — множество номеров всех виртуальных узлов в РВС. Тогда *расписание* S_q выполнения процессов $u_{iq} \in U_q$, планируемых в период T_q в РВС, определяется как множество отрезков времени (*слотов*) s_{imq}, зарезервированных для выполнения этих процессов на ВУ $\phi_m \in \Phi$:

$$S_q = \{s_{imq} : \ i \in I_q, \ m \in M_q\}, \tag{1.1}$$

где I_q — множество номеров процессов, которые будут выполняться в РВС согласно расписанию S_q, а M_q — множество номеров ВУ, на которых данные процессы будут запущены.

Обозначим через r^e_{imq} величину, характеризующую ресурс с номером e, который пользователь заказывает для выполнения процесса u_{iq} на некотором ВУ ϕ_m. Например, величина r^e_{imq}, относящаяся к оперативной памяти, обозначает требуемый на ВУ ϕ_m объем данной памяти, а относящаяся к программному обеспечению, отражает наличие ($r^e_{imq}=1$) или отсутствие ($r^e_{imq} = 0$) данного программного обеспечения на ВУ ϕ_m.

Величину, характеризующую ресурс с номером e, который доступен ВУ $\phi_m \in \Phi$ в момент времени $t \in T$, обозначим через $R^e_m(t)$. Тогда ресурсы, доступные для ВУ ϕ_m в этот момент времени, характеризуются множеством $\{R^e_m(t) : \ e = 1, 2, \ldots, N_r\}$, где e — номер ресурса, N_r — число различных видов ресурсов в РВС.

Расписание S_q заданий для РВС является *допустимым*, если выполняются условия достаточности имеющихся ресурсов для их выполнения:

$$r^e_{imq} \le R^e_m(t), \quad \forall t \in s_{imq}, \quad \forall i \in I_q, \quad \forall m \in M_q, \quad e = 1, 2, \ldots, N_r. \tag{1.2}$$

Также важным свойством расписания является его *непротиворечивость*. Под непротиворечивостью расписания S_q понимается выполнение следующих условий.

1. Каждому процессу из множества U_q, который может быть запущен в РВС, должен быть сопоставлен один слот в расписании S_q. Пусть

N'_q — число слотов в расписании S_q, соответствующих планируемым в период T_q процессам, N_q — число процессов всех заданий, планируемых в период T_q, которые могут выполняться в РВС. Тогда должно быть справедливо равенство:

$$N'_q = N_q.$$

2. Никакие два слота $s_{imq} \in S_q$ и $s_{jmq} \in S_q$ $(i \neq j)$ не должны быть сопоставлены одному ВУ ϕ_m одновременно:

$$|s_{imq} \bigcap s_{jmq}| = 0,$$

где $|\cdot|$ — длина соответствующего отрезка времени.

Для составления расписания *параллельных* заданий также необходимо брать в расчет еще два условия непротиворечивости. Для описания данных условий дополнительно введем разбиение множества Φ всех ВУ в РВС на непересекающиеся подмножества $\Phi_g = \{\phi_m : \ m \in M_g\}$ — группы ВУ:

$$\Phi = \bigcup_{g=1}^{N_g} \Phi_g, \quad \Phi_g \bigcap \Phi_p = \varnothing, \quad g, p = 1, 2, \ldots, N_g, \quad g \neq p, \tag{1.3}$$

где N_g — количество групп ВУ в РВС, M_g — множество номеров ВУ, принадлежащих группе Φ_g, $\varnothing$ — пустое множество.

Пусть $U_{pq} = \{u_{iq} : i \in I_{pq}\}$ — множество процессов, принадлежащих p-му заданию, поступившему в период T_q, где I_{pq} — множество номеров процессов этого задания. При этом

$$\bigcup_{p=1}^{P_q} U_{pq} = U_q, \quad U_{lq} \bigcap U_{pq} = \varnothing, \quad l, p = 1, 2, \ldots, P_q, \quad l \neq p, \tag{1.4}$$

где P_q — количество заданий, поступивших в период планирования T_q.

Тогда дополнительные условия непротиворечивости раписания S_q выполнения процессов, принадлежащих *параллельным* заданиям следующие.

3. Все процессы *параллельного* задания должны быть спланированы на выполнение в пределах одной группы ВУ. Пусть M_g — множество ВУ g-й группы, тогда в расписании S_q для каждого слота s_{imq}, сопоставленного с i-м процессом из множества U_{pq}, должно выполняться условие $m \in M_g$.

4. Все процессы *параллельного* задания должны иметь одинаковый момент времени запуска и завершения. Для всех $i, j \in N_{pq}$ в расписании S_q должно выполняться равенство $s_{imq} = s_{jkq}, \;\; m, k \in M_q$. Слотам s_{imq} и s_{jkq} соответствуют процессы одного параллельного задания.

В большинстве случаев для одного множества процессов U_q можно составить множество расписаний $\widehat{S}_q = \left\{S_{\alpha q} : \; \alpha \in A'_q\right\}$, где A'_q — множество номеров всех возможных расписаний для процессов из множества U_q.

В процессе планирования для номера периода планирования $q \neq 0$ расписания $S_{\alpha q} \in \widehat{S}_q$ должны удовлетворять условию сопряжения. Пусть длина $\tau_{\alpha\beta q}$ отрезка времени ожидания всех ВУ в РВС между завершением слотов расписания $S_{\beta(q-1)}$ и началом слотов расписания $S_{\alpha q}$ задается следующим образом:

$$\tau_{\alpha\beta q} = \sum_{\forall m \in M_q} (t_{m\alpha q} - t_{m\beta(q-1)}),$$

где $t_{m\alpha q}$ — момент времени начала выполнения первого слота в расписании $S_{\alpha q}$ на ВУ ϕ_m, а $t_{m\beta(q-1)}$ — момент времени завершения выполнения последнего слота в расписании $S_{\beta(q-1)}$ на ВУ ϕ_m; $\alpha \in A'_q$, $\beta \in A'_{q-1}$. Тогда расписание $S^*_q \in \widehat{S}_{\alpha q}$ будет удовлетворять условию сопряжения с расписанием $S_{\beta(q-1)}$, если:

$$\tau^*_{\beta q} = \min_{\alpha \in A'_q} \tau_{\alpha\beta q}. \tag{1.5}$$

Пусть $\widetilde{S}_q \subseteq \widehat{S}_q$ — подмножество расписаний, удовлетворяющих условию сопряжения. Введем для расписаний $S_{\alpha q} \in \widetilde{S}_q$ целевую функцию $f : \widetilde{S}_q \to R_+$, где R_+ — множество действительных положительных чисел. Оптимальным будем считать расписание $S^*_q \in \widetilde{S}_q$, удовлетворяющее условию:

$$f(S^*_q) = \min_{\alpha \in A'_q} f(S_{\alpha q}).$$

В основе целевой функции могут также лежать среднее время выполнения всех заданий, среднее время ожидания запуска заданий, загруженность ресурсов и другие критерии [91]. Важно выбирать такие целевые функции, которые позволяли бы учесть приоритеты процессов и равномерно загрузить работой ВУ в РВС.

В связи с тем, что задача составления расписания в общем случае является NP-полной, на практике приходится ограничиваться поиском субоптимального решения [13]. Сформулируем задачу планирования выполнения заданий в распределенной вычислительной системе.

Задача планирования. Найти для каждого периода планирования T_q субоптимальное расписание S_q вида (1.1) выполнения процессов $u_{iq} \in U_q$. При этом S_q должно удовлетворять условиям ресурсных ограничений (1.2), условию сопряжения (1.5), а также условиям непротиворечивости 1–2 (для непараллельных заданий) или 1–4 (для параллельных заданий).

1.3. Алгоритм планирования выполнения заданий в распределенной вычислительной системе

В реальных условиях эксплуатации вычислительные системы рассчитаны на непрерывное поступление заданий. Два последовательно найденных расписания должны удовлетворять условию сопряжения. Другими словами, между выполнением заданий этих расписаний не должно быть лишнего времени простаивания вычислительных ресурсов, если того не требуют сами задания.

На следующем рисунке показан пример сопряженных расписаний выполнения *непараллельных* заданий.

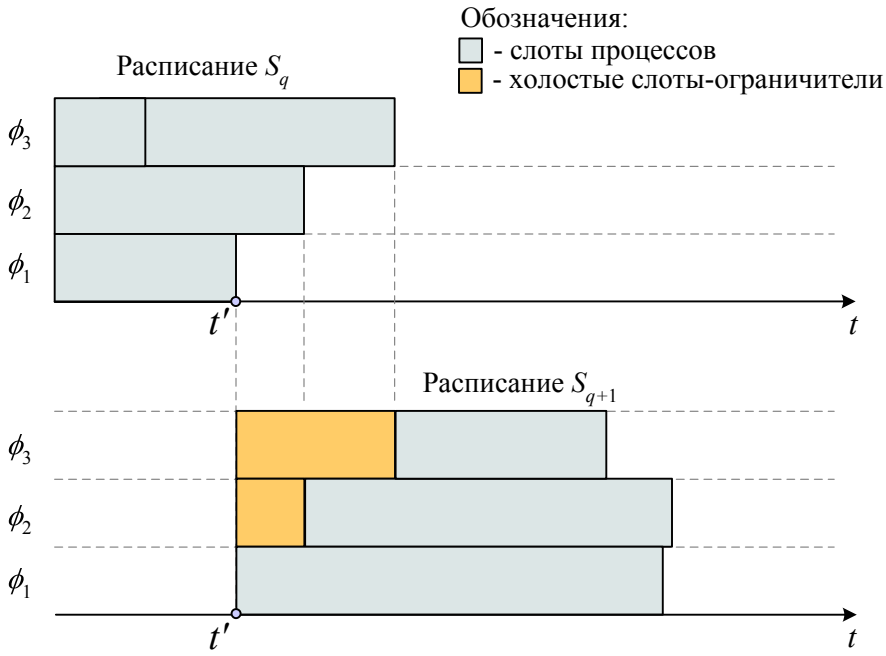

Рис. 1.2. Пример двух сопряженных расписаний S_q и S_{q+1} запуска непараллельных заданий

На рис. 1.2 схематически изображены два сопряженных расписания S_q и S_{q+1} для системы из трех ВУ, которые обозначены как ϕ_1, ϕ_2 и ϕ_3. Для каждого расписания S_{q+1}, на основании предыдущего расписания S_q, формируются холостые слоты-ограничители, размещаемые в самом “начале”. На указанные слоты алгоритм составления расписаний не должен оказывать влияния. В результате эти слоты ограничивают по времени запуска процессы U_q, обеспечивая максимально плотную состыковку двух последовательных расписаний. За момент времени t' на данном рисунке обозначено начало участка расписания S_q, на основании которого создаются холостые слоты-ограничители в расписании S_{q+1}.

На рис. 1.3 показан пример сопряженных расписаний выполнения *параллельных* заданий. Для каждого нового расписания S_{q+1} на основании предыдущего расписания S_q формируется множество холостых слотов-ограничителей аналогично ситуации с непараллельными заданиями.

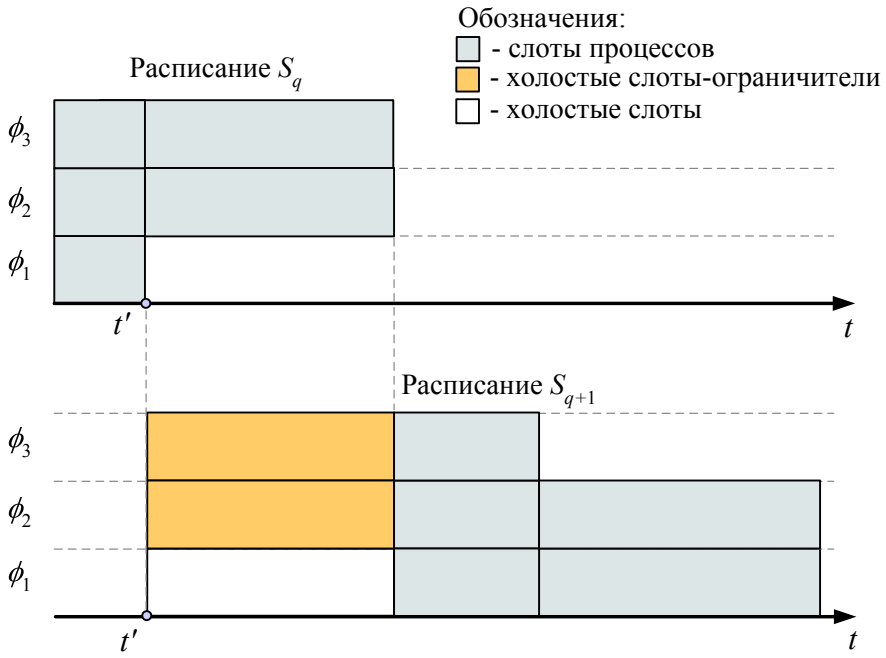

Рис. 1.3. Пример двух сопряженных расписаний S_q и S_{q+1} выполнения параллельных заданий

Схема планирования выполнения заданий с составлением расписаний генетическим алгоритмом для РВС изображена на рис. 1.4.

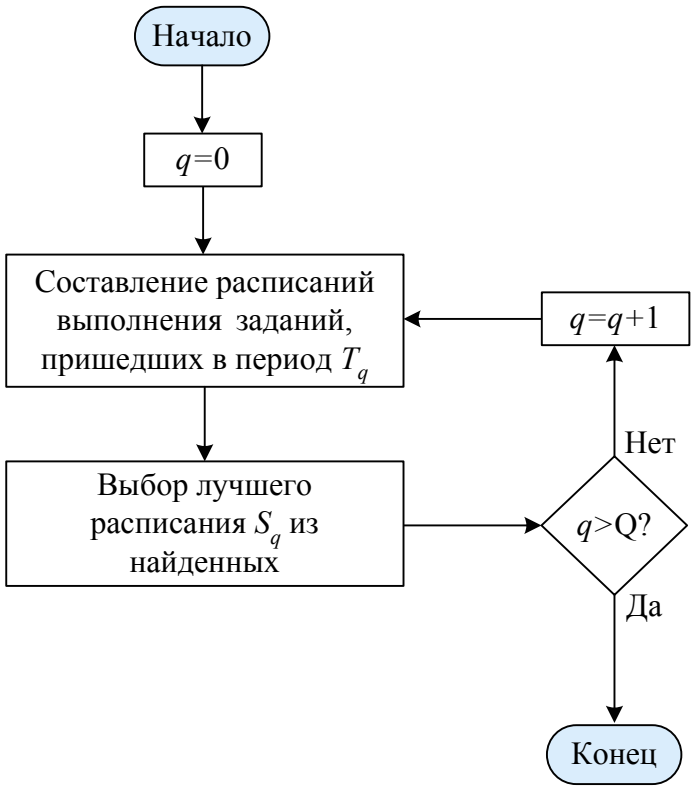

Рис. 1.4. Схема планирования заданий в РВС

Формальное описание алгоритма планирования, схематически изображенного на рис. 1.4, следующее.

1. Задаётся начальный номер периода планирования $q = 0$.
2. По истичении отрезка времени T_q составляется множество непротиворечивых допустимых расписаний для заданий, поступивших в этот период. Для этого запускается генетический алгоритм с передачей ему на вход множества процессов U_q и множества Φ всех ВУ в РВС. Запущенный ГА завершает поиск расписаний только по окончанию периода планирования T_{q+1}.
3. Из множества найденных генетическим алгоритмом расписаний выбирается лучшее расписание S_q согласно целевой функции.
4. Если $q > Q$, то работа алгоритма завершается.
5. Задаётся новый номер периода планирования $q = q + 1$.
6. Осуществляется переход на шаг 2. ■

Система планирования непрерывно накапливает задания, поступившие к ней от пользователей. В конце каждого периода T_q множество заданий, поступивших в этот период, фиксируется, и для них составляется расписание. Все последовательно составленные расписания $\{S_q\}$ помещаются в общий пул расписаний, образуя единый план выполнения заданий. Система планирования должна следить за изменениями пула расписаний и запускать задания на узлах РВС в определенный соответствующим расписанием момент времени.

Так как для каждого множества процессов U_q можно составить несколько расписаний, то введение целевой функции позволяет ранжировать полученные для каждого периода планирования T_q расписания согласно их эффективности.

Часто в качестве критериев оценки расписаний используются критерий длины расписания и среднего взвешенного времени прохождения процессов. Пусть для одного множества процессов U_q можно составить

несколько расписаний: $\widehat{S}_q = \{S_{\alpha q} : \alpha \in A_q\}$. Под длиной $l_{\alpha q}$ расписания $S_{\alpha q}$ понимается следующая величина:

$$l_{\alpha q} = \bigcup_{s_{im\alpha q} \in S_{\alpha q}} s_{im\alpha q}.$$

Тогда лучшим расписанием S_q^* будет расписание с минимальным значением его длины:

$$f_q^* = \min_{\alpha \in A_q} l_{\alpha q}.$$

Целевая функция с критерием среднего взвешенного времени прохождения процессов в расписании S_q определяется следующим образом:

$$f_q = \sum_{i=1}^{N_q} \lambda_{imq} \left(t_{imq} + |s_{imq}| \right),$$

где λ_{imq} — вес процесса $u_{iq} \in U_q$ при запуске его на ВУ ϕ_m, t_{imq} и $|s_{imq}|$ — начало и длина слота $s_{imq} \in S_q$ соответственно, $m \in M_q$, M_q — множество номеров ВУ, с которыми сопоставлены слоты в расписании S_q, N_q — число слотов в расписании S_q.

Обе эти целевые функции имеют существенные достоинства и недостатки при применении в качестве функции пригодности в генетических алгоритмах для составления расписания в РВС. Длина расписания учитывает лишь последние по времени завершения процессы и плохо “справляется” с уплотнением остальных процессов. Критерий среднего взвешенного “справляется” с уплотнением расписания. Тем не менее, значение целевой функции с этим критерием существенно зависит от порядка следования слотов на ВУ. Для равнозначных расписаний это ведет (при большом числе сильно различающихся по длине слотов) к существенно различным значениям целевой функции, что мешает работе генетических алгоритмов.

Для устранения вышеописанных недостатков, а также в целях учета приоритета заданий в разработанных автором алгоритмах используется следующая целевая функция:

$$f_\alpha = \sqrt[G]{\prod_{m \in M_\alpha} \sum_{i \in I_{m\alpha}} \lambda_{im\alpha} |s_{im\alpha}|}, \tag{1.6}$$

где M_α — множество номеров ВУ, которым сопоставлены слоты из расписания S_α; $I_{m\alpha}$ — множество номеров слотов, сопоставленных с ВУ ϕ_m; G — число всех виртуальных узлов в РВС; $|s_{im\alpha}|$ — длина слота $s_{im\alpha} \in S_\alpha$; $\lambda_{im\alpha}$ — вес процесса u_i при запуске его на ВУ ϕ_m. Вес $\lambda_{im\alpha}$ процесса u_i на ВУ ϕ_m необходим для учета его приоритета: $\lambda_{im\alpha} = (p_{i\alpha})^{d_{im\alpha}}$, где $p_{i\alpha}$ — приоритет процесса u_i, $d_{im\alpha}$ — порядковый номер следования слота $s_{im\alpha}$ на ВУ ϕ_m. Приоритет процесса равен установленному администратором приоритету для каждой из очередей заданий, с которой задание данного процесса сопоставлено.

В программной реализации целевой функции при больших значениях d_{imq} вес процесса λ_{imq} может превысить максимальное число, которое можно представить типом данных. Решением данной проблемы может стать масштабирование d_{imq}. Так, в исследуемых программных реализациях алгоритмов значение d_{imq} масштабировалось как $(1 + 0.1d_{imq})$.

Глава 2. Генетические алгоритмы составления расписаний для распределенных вычислительных систем

Глава посвящена применению генетических алгоритмов к составлению расписания выполнения заданий в распределенной вычислительной системе. Описывается предлагаемые автором генетические алгоритмы для составления расписания запуска непараллельных и параллельных заданий в РВС. Приводятся результаты численных экспериментов, призванных выявить оптимальные значения параметров разработанных ГА.

2.1. Методы кодирования расписаний в генетических алгоритмах

Существует множество задач составления расписаний в различных сферах человеческой деятельности [95]. Для их решения находят применение стохастические алгоритмы. Генетический алгоритм — представитель данного класса алгоритмов, положительно зарекомендовавший себя применительно к поиску расписания с ограничениями.

Важным вопросом здесь является метод представления решения в виде строки (хромосомы). Варианты подобного представления можно разделить на косвенные и прямые. В косвенном кодируются указания для последующего построения расписания. Прямое представление подразумевает “жесткое” кодирование последовательности выполнения заданий на тех или иных элементарных вычислительных машинах. При составлении расписания для РВС элементарной вычислительной машиной служит виртуальный узел.

В случае косвенного представления гарантируется, что последующее построение расписания из полученных в закодированной строке ин-

струкций приведет к непротиворечивому расписанию. Примером косвенного представления может служить закодированный относительный порядок следования заданий. В нём несколько различных хромосом могут соответствовать одному и тому же расписанию.

При прямом кодировании расписание однозначно кодируется в хромосому. Примером прямого представления может служить хромосома, кодирующая время планируемого запуска заданий на заданном множестве ВУ. Тем не менее, у большинства прямых представлений не каждая хромосома из возможных соответствует непротиворечивому расписанию. Для использования данного представления часто применяют механизм восстановления.

Одним из вариантов представления закодированного для ГА расписания в виде битовой строки является булева матрица упорядочения заданий. Последовательность из N заданий можно описать матрицей $m = [N \times N]$. Если элемент матрицы $m_{ij} = 1$, то i-ое задание следует запустить перед j-ым, $m_{ii} = 0$, $i, j = 1, 2, \ldots, N$.

Для хромосом в виде матриц упорядочения вводятся специальные генетические операторы. Так, оператор пересечения (intersection operator) получает новую матрицу упорядочения на основе двух родительских в результате применения к каждой матрице логического оператора конъюкции &, таким образом, что $m'_{ij} = m^1_{ij}$ & m^2_{ij}, где m^1 и m^2 — "родительские" матрицы для новой матрицы m'. После применения & необходимо добавить к m' все недостающие значения элементов одной из "родительских" матриц.

Другим генетическим оператором, применяемым к подобному представлению расписания, является оператор объединения (*union operator*). Данный оператор близок к классическому оператору скрещивания, но исключает нарушение непротиворечивости расписаний. Оператор включает следующие шаги.

1. Разделение множества генов на два непересекающихся подмножества S_1 и S_2.

2. Создание двух новых матриц: первой — m^1, содержащей все элементы матрицы первого родителя, соответствующих S_1 и второй — матрицы

m^2, включающей все элементы матрицы второго родителя, которые соответствуют S_2.

3. Применение логического оператора дизъюнкции: $m'_{ij} = m^1_{ij} \mid m^2_{ij}$.

Как для оператора пересечения, так и для оператора объединения, результирующая матрица кодирует непротиворечивое расписание. Это происходит ввиду того, что подмножества S_1 и S_2 не пересекаются. Описанная форма хромосомы используется и в других задачах, в которых требуется найти оптимальную последовательность некоторых действий, например, в задачах поиска кратчайшего пути [92, 164].

При составлении расписания для РВС необходимо учитывать, что процессов может быть достаточно много. Каждая хромосома, представленная в виде матрицы $[N \times N]$, будет занимать объем памяти N^2 бит, где N — число вычислительных процессов в расписании. Для небольших РВС использование подобных матриц вполне оправдано, так как они будут занимать малый объем оперативной памяти. В противном случае, популяция может занять достаточно большой объем памяти, что негативно скажется на производительности всего алгоритма. Таким образом, автор диссертации считает, что представление расписания в ГА для РВС в виде матрицы упорядочения не целесообразно.

Для составления расписания классический оператор скрещивания может использоваться с последующим восстановлением результирующей хромосомы [126]. Рассмотрим представление расписания в виде направленного ациклического графа $\Gamma = \langle V, C \rangle$. Здесь V — множество заданий, C — множество дуг, представляющих собой последовательность выполнения заданий на одной элементарной вычислительной машине. Данный граф однозначно определяет запуск заданий в расписании для РВС.

При прямом кодировании расписания граф Γ можно представить в виде двоичной строки (хромосомы) длиной $MN(N-1)/2$, где M — число машин, N — число заданий. Для составления хромосомы каждая ветвь графа кодируется в хромосоме последовательно. Нулевым битом описывается каждая дуга, принадлежащая множеству дуг C в случае, если задание v_i следует перед v_j на элементарной вычислительной машине, сопоставлен-

ной с указанной веткой графа. В противном случае (если задание v_i следует после v_j) дуга кодируется ненулевым битом.

К подобному представлению хромосомы применяют как одноточечный, так и многоточечный классический оператор скрещивания. В результате его применения расписание может терять свою непротиворечивость. Таким образом, необходимо будет использовать восстановление хромосомы. Алгоритм восстановления получил название алгоритма гармонизации (harmonization algorithm) [127]. Данный алгоритм включает локальную и глобальную гармонизации. В процессе локальной гармонизации исключаются конфликты заданий на каждой машине в отдельности. В результате глобальной гармонизации из расписания исключаются конфликты заданий на всех машинах.

Недостатком гармонизации является то, что в результате могут исключаться из популяции особи хоть и неверные с точки зрения ограничений на расписание, тем не менее, полезные для формирования особей последующих поколений. Другим недостатком использования механизма гармонизации является увеличение сложности алгоритма. Для РВС с большим числом узлов гармонизация может существенно замедлить работу ГА.

Другой формой хромосомы может служить представление расписания как решения задачи упорядочения (например, в задаче коммивояжёра). В этом случае хромосомой будет являться матрица последовательности заданий для всех машин. Данная форма хромосомы легла в основу представления расписания в разработанном автором ГА (см. раздел 2.2).

В матрице упорядочения элементы одной строки представляют собой последовательность выполнения заданий на одной элементарной вычислительной машине. К подобного рода матрицам можно применить скрещивание с заменой подпоследовательностей (subsequence exchange crossover). Пусть даны две хромосомы в виде матриц упорядочения m^1 и m^2. Два множества генов (из m^1 и из m^2) будут взаимозаменяемыми тогда и только тогда, когда они содержат одинаковый набор генов и процессы, соответствующие данным генам будут выполняться на одной машине. Описываемый оператор ищет взаимозаменяемые множества генов для каждой из элементарной вычислительной машины и меняет их местами, генерируя тем

самым новую матрицу упорядочения. После скрещивания с заменой подпоследовательностей часто необходимо восстановление расписания ввиду возможной утраты непротиворечивости.

Косвенные представления также широко используются при поиске расписания заданий с помощью ГА. Так, формой хромосомы может служить последовательность генов, не сгруппированных по признаку принадлежности той или иной элементарной вычислительной машине, а лишь упорядоченная по некоторому правилу [69]. При преобразовании хромосомы в конкретное расписание, процессы, соответствующие генам, последовательно назначаются на выполнение соответствующим машинам, при этом учитываются ограничения, налагаемые на расписание. В результате из одной хромосомы может получиться два или более различных расписания.

Для косвенных форм представления также разработан ряд генетических операторов. Так, оператор скрещивания с исключением генов (precedence preservative crossover) [68] полностью соблюдает порядок следования генов в родительских хромосомах. Дополнительно используется битовая строка, определяющая какая родительская особь будет следующим источником очередного гена. Соответствующий ген исключается из другой родительской хромосомы. Данный шаг повторяется каждый раз с первого гена родительских хромосом до тех пор, пока хромосомы не будет иметь нулевую длину.

Принцип исключения генов стал базисом для разработанного алгоритма генетического оператора скрещивания. Вместо битовой строки для определения родительской особи в нём используется селекция методом рулетки [104]. Точки разрыва при этом выбираются строго между генами, соответствующими различным группам ВУ (например, вычислительным кластерам). В результате такого выбора точек разрыва исключаются ситуации, когда процессы одного параллельного задания назначаются различным группам ВУ. Затем гены копируются в дочернюю хромосому согласно точкам разрыва попеременно с первого и второго родителя. Гены, которые уже были скопированы, пропускаются (см. раздел 2.2).

Недостатком использования косвенных форм представления является необходимость частого преобразования хромосомы в расписание для опре-

деления значения функции пригодности. Такие преобразования могут негативно сказаться на производительности алгоритма составления расписания для больших РВС.

2.2. Генетические алгоритмы составления расписания для распределенной вычислительной системы

В разделе описаны детали разработанных генетических алгоритмов составления допустимых непротиворечивых субоптимальных расписаний выполнения параллельных и непараллельных заданий в РВС. Представлены разработанные автором генетические операторы, описан механизм учета ресурсных ограничений, налагаемых на расписания, а также механизм учета в генетических операторах зависимостей между заданиями.

При составлении расписания непараллельных заданий вычислительные процессы одного задания могут быть спланированы на запуск в различные моменты времени. Предполагается, что все непараллельные задания в очереди включают процессы, не обменивающиеся между собой данными.

Для параллельной программы важен одновременный запуск всех ее процессов. Параллелизм часто необходим для синхронизации процессов одной группы между собой, например, при обмене сообщениями по протоколу MPI. В случае рассинхронизации данного типа процессов возникает вероятность простаивания или нарушения работы алгоритма, реализованного в параллельной программе. Подобная ситуация может произойти, если процесс находится в ожидании сообщений от другого процесса, который в настоящий момент не запущен. По этой же причине планируемая длительность выполнения всех процессов одного задания должна совпадать.

При составлении расписаний разработанными генетическими алгоритмами не ведется учет времени доставки ПО и данных на узлы РВС перед выполнением. В связи с этим обычно применяется механизм предварительного резервирования ресурсов [147, 148].

На следующем рисунке представлена объединенная схема разрабо-

танных генетических алгоритмов составления расписаний выполнения как непараллельных, так и параллельных заданий. Пунктиром обозначен оператор уплотнения, применяемый только в случае с параллельными заданиями.

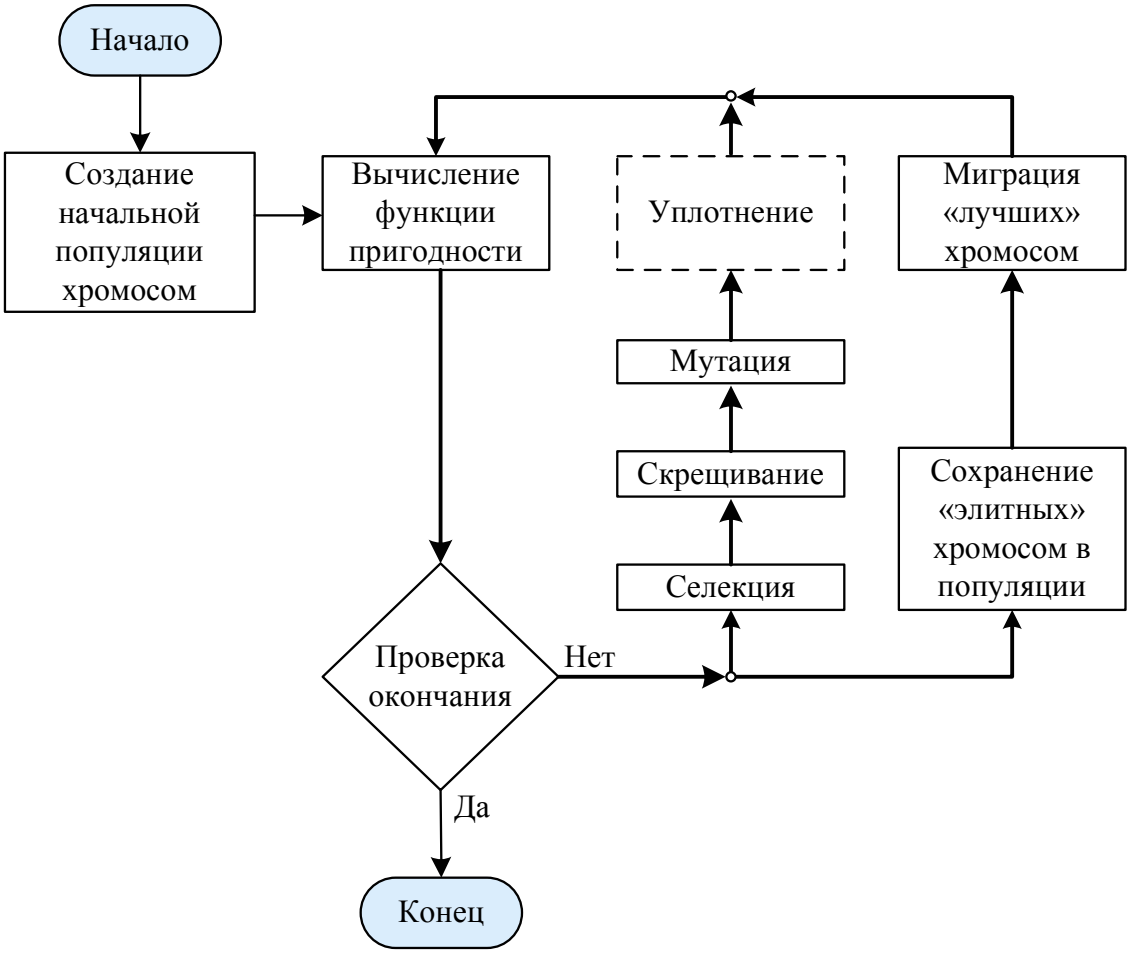

Рис. 2.1. Схема разработанных генетических алгоритмов составления расписаний выполнения заданий в распределенной вычислительной системе

Под сохранением “элитных” хромосом подразумевается сохранение подмножества хромосом в популяции перед применением генетических операторов с последующей заменой “худших” хромосом в популяции на сохраненные.

Очевидно, что с увеличением числа процессов и ресурсов в РВС, растут и вычислительные затраты на составление расписания, а с ними и время от постановки пользователем задания в очередь до начала выполнения этого заданий. Для уменьшения данного времени используется островная модель ГА с миграцией хромосом с минимальным значением функции

пригодности между изолированными подпопуляциями через определенное число поколений [112]. Эти подпопуляции развиваются независимо друг от друга в течении числа I_m поколений ГА (число поколений изоляции). По истечении I_m происходит распространение лучших хромосом между всеми подпопуляциями. Миграция хромосом позволяют подпопуляциям совместно использовать генетический материал.

Далее будем опускать номер периода планирования q, так как он *не* требуется для понимания работы алгоритма составления *одного* расписания. Для описания деталей алгоритма будем придерживаться следующих понятий и обозначений.

Назовем хромосомой множество $X_\alpha = \{s_{im\alpha} : \ i \in I_\alpha, m \in M_\alpha\}$, где $\alpha \in A$, A — множество номеров расписаний, кодируемых хромосомами в популяции генетического алгоритма, I_α — множество номеров процессов, которые будут выполняться в РВС согласно расписанию S_α, M_α — множество всех ВУ, с которыми сопоставлены слоты расписания S_α. Элемент $s_{im\alpha}$ (слот) хромосомы X_α в терминах ГА является геном. Через $X = \{X_\alpha : \ \alpha \in A\}$ обозначим множество хромосом, составляющих популяцию генетического алгоритма.

Входными параметрами описываемых ГА являются следующие: p_c — вероятность применения оператора скрещивания; p_m — вероятность применения оператора мутации; E — количество элитных хромосом в популяции; V — множество заданий; T — отрезок времени, на протяжении которого выполняется генетический алгоритм; Ф — множество ВУ в распределенной вычислительной системе. Предлагаемый алгоритм в общем виде следующий.

1. *Создание начальной популяции.* С помощью модифицированного алгоритма обратного заполнения в случае параллельных заданий или алгоритма FCFS в случае непараллельных заданий из заданий из множества V создается начальная популяция: $X = \{X_\alpha : \ \alpha \in A\}$, где X_α — хромосомы.

2. *Вычисление значений* функции пригодности f_α по формуле (1.6) для всех хромосом из X.

3. *Проверка окончания.* Если выделенный для составления расписания отрезок времени T завершен, то работа алгоритма прекращается.

4. *Сохранение элитных хромосом.* Из множества X формируется множество Y, состоящее из E хромосом с наименьшими значениями f_α.

5. *Селекция.* Создается подмножество $X' \subseteq X$ хромосом, выбранных методом рулетки [104].

6. *Скрещивание.* Для каждой хромосомы из подмножества X' случайным образом выбирается число на отрезке $[0, 1]$, и если это число меньше или равно p_c, то к данной хромосоме применяется оператор скрещивания. Новые хромосомы, полученные в результате действия на них оператора скрещивания, заменяют в X' хромосомы с максимальными значениями f_α.

7. *Мутация.* Для каждой хромосомы из подмножества X' случайным образом выбирается число на отрезке $[0, 1]$, и если это число меньше или равно p_m, то к данной хромосоме применяется оператор мутации. Каждая хромосома, полученная в результате действия оператора мутации, заменяет в X' исходную хромосому.

8. *Уплотнение.* К каждой хромосоме из подмножества X' применяется оператор уплотнения. Новые хромосомы, полученные в результате действия оператора уплотнения, заменяют в X' исходные хромосомы. Данный оператор уменьшает отрезки времени простаивания ВУ в расписаниях, соответствующих данным хромосомам.

9. *Восстановление элитных хромосом.* Хромосомы с максимальными значениями f_α в подмножестве X' заменяются на хромосомы из множества Y.

10. Текущей популяцией становится X': $X = X'$. Осуществляется переход на шаг 2. ■

Элемент $s_{im\alpha}$ хромосомы X_α в терминах ГА является геном и соответствует одному слоту. Длина $|s_{im\alpha}|$ (продолжительность) слота $s_{im\alpha}$

рассчитывается исходя из параметров, указанных пользователем в файле описания соответствующего задания. При составлении расписания выполнения заданий для гетерогенной РВС необходимо масштабировать длину слотов в соответствии с фактически используемым типом процессора. Новое значение длины слота вычисляется по следующей формуле:

$$|s_{im\alpha}| = |s_{ih\alpha}|\frac{\kappa_h}{\kappa_m}, \tag{2.1}$$

где κ_h и κ_m — коэффициенты производительности процессоров, которые соответствуют ВУ ϕ_h и ϕ_m. Данные коэффициенты расчитываются на основании параметров, влияющих на производительность процессоров, например, числа конвейеров, эффективности работы процессора с различными типами памяти, тактовой частоты и т.п.

Для увеличения производительности программной реализации алгоритма необходимо рассчитывать все возможные варианты длительностей заданий *перед* началом работы ГА. Для этого заполняется таблица, в которой по одному направлению откладываются номера процессов заданий, а по другому — номера ВУ. В каждой ячейке такой таблицы хранится вычисленная по формуле (2.1) длина слота, соответствующего процессу p-го задания на ВУ $\phi_m \in \Phi$, $m \in M$, M — множество номеров всех ВУ в РВС, $p = 1, 2, \ldots, P$, P — число заданий для планирования в период T_q.

2.2.1. Форма представления расписания

Каждое расписание $X_\alpha \in X$ в ГА должно быть закодировано в форме хромосомы. Форма хромосомы, применяемая автором для составления расписания непараллельных и параллельных заданий следующая. Для описания одного слота в хромосоме X_α необходимы параметры: ВУ $\phi_m \in \Phi$, номер слота i и длина $|s_{im\alpha}|$ слота $s_{im\alpha} \in X_\alpha$ на данном ВУ. Здесь Φ — множество всех ВУ в РВС. Хромосомой X_α тогда может служить последовательное расположение кортежей $< \phi_m, i, |s_{im\alpha}| >$ в порядке запуска соответствующих процессов на ВУ.

Автор предлагает выносить номера ВУ за пределы области кодирования слотов в отдельный вектор X'_α (“заголовок” хромосомы). Каждый

элемент данного вектора есть число $N_{m\alpha}$ слотов на ВУ ϕ_m в "теле" хромосомы X_α. Тогда сами слоты будут описываться кортежами $< i, |s_{im\alpha}| >$ и храниться во второй части хромосомы — в векторе X''_α ("тело" хромосомы). Таким образом, $X'_\alpha = \{N_{m\alpha}\}$, $X''_\alpha = \{< i, |s_{im\alpha}| >\}$, а вся хромосома: $X_\alpha =< X'_\alpha, X''_\alpha >$. На рис. 2.2 схематически показана одна хромосома.

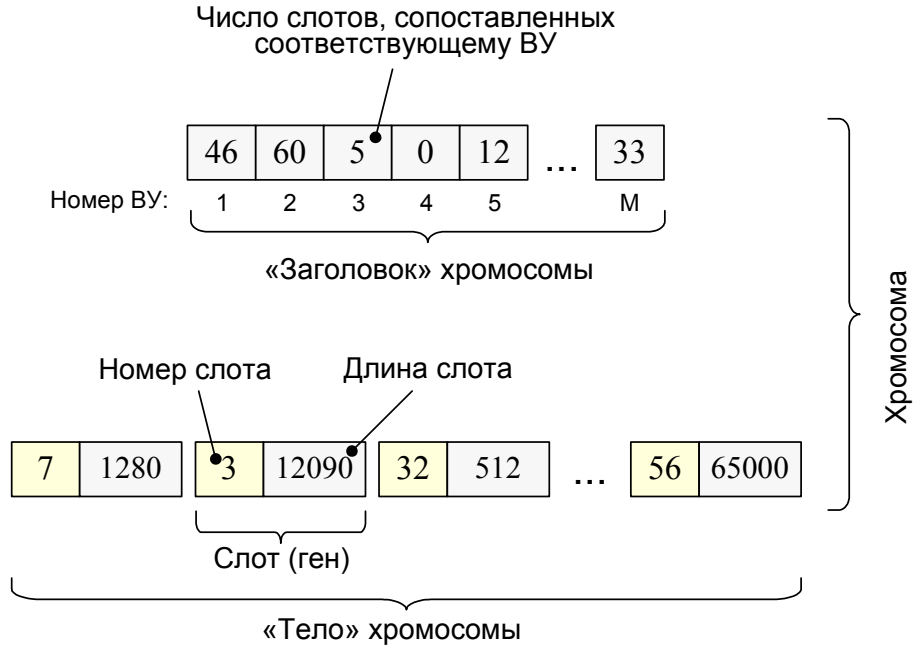

Рис. 2.2. Форма представления хромосомы в оперативной памяти ЭВМ

Помимо экономии используемой под хромосомы оперативной памяти при составлении расписания с задействованием большой популяции, такая форма позволяет оставлять слоты одного ВУ расположенными физически последовательно в памяти после применения оператора мутации. Если бы номера ВУ были расположены непосредственно в кортеже слота, то в результате обмена двух слотов местами, необходимо было бы либо менять номера ВУ местами, что нарушало бы непрерывность расположения в памяти слотов одного ВУ, либо производить копирование слотов, которые в данном случае состоят из трех компонент, что замедлило бы работу программы. Сохранение непрерывного расположения слотов дает также возможность генетическому оператору скрещивания копировать в дочернюю хромосому за раз непрерывные участки оперативной памяти родительских

хромосом, что позволяет существенно ускорить процесс скрещивания.

Несмотря на то, что автор рекомендует в программной реализации хранить хромосомы в форме, описанной в данном разделе, при описании алгоритмов для простоты повествования будем работать с хромосомами в форме множества слотов. Очевидно, что описанные формы хромосом эквивалентны для генетических алгоритмов.

2.2.2. Функция пригодности

Роль функции пригодности в ГА выполняет целевая функция. Для составления расписания выполнения заданий функция пригодности тогда принимает следующий вид:

$$f_\alpha = \sqrt[G]{\prod_{m \in M_\alpha} \sum_{i \in I_{m\alpha}} \lambda_{im\alpha} |s_{im\alpha}|},$$

где M_α — множество номеров ВУ, которым соответствуют слоты в хромосоме X_α, $I_{m\alpha}$ — множество номеров слотов, соответствующих одному ВУ ϕ_m в хромосоме X_α, G — число всех ВУ в РВС, $|s_{im\alpha}|$ — длина слота $s_{im\alpha} \in X_\alpha$, $\lambda_{im\alpha}$ — вес процесса u_i при запуске его на ВУ ϕ_m, согласно расписанию, кодируемому хромосомой X_α. Вес $\lambda_{im\alpha}$ процесса u_i на ВУ ϕ_m необходим для учета его приоритета: $\lambda_{im\alpha} = (p_{i\alpha})^{d_{im\alpha}}$, где $p_{i\alpha}$ — приоритет процесса u_i, показатель степени $d_{im\alpha}$ — порядковый номер следования слота $s_{im\alpha}$ на ВУ ϕ_m.

Приоритет процесса равен установленному администратором приоритету для каждой из очередей, с которой задание данного процесса сопоставлено. В результате, расписания, в которых более приоритетные процессы завершаются раньше, чем менее приоритетные, имеют более низкое значение данной функции. Чем ниже значение этой функции, тем ГА считает расписание более пригодным. Так происходит учет приоритетов при составлении расписания — более приоритетные задания (в лучших расписаниях в популяции) ставятся на расчет раньше менее приоритетных заданий.

2.2.3. Учет зависимостей между заданиями

Между заданиями могут существовать зависимости, возникающие при ожидании заданием результатов выполнения других заданий. В общем случае такие зависимости можно представить в виде направленного ациклического графа. В диссертации рассматриваются ситуацию, когда каждое задание может зависеть только от одного задания, что в большинстве случаев достаточно. Если i-е задание зависит от j-го, то должно выполняться условие $t_{im\alpha} \geq t_{jm\alpha} + |s_{jm\alpha}|$, где $t_{im\alpha}$ и $t_{jm\alpha}$ — моменты времени запуска соответствующих заданий, $|s_{jm\alpha}|$ — длительность выполнения процесса j-го задания на ВУ $\phi_m \in \Phi$, $m \in M_\alpha$, M_α — множество номеров ВУ, которым соответствуют слоты в расписании, кодируемом хромосомой X_α, $i, j \in I_\alpha$, I_α — множество номеров слотов в хромосоме X_α.

Для учета зависимостей между заданиями при мутации необходимо соблюдать коридор допустимых значений для отрезка времени выполнения процессов i-го задания в расписании, кодируемом хромосомой X_α: $\left[t^{\min}_{im\alpha},\ t^{\max}_{im\alpha}\right]$. Коридор мутации определяет отрезок времени с минимальным значением момента запуска $t^{\min}_{im\alpha}$ и максимальным значением момента завершения $t^{\max}_{im\alpha}$ выполнения процессов соответствующего задания. В рамках указанного отрезка времени слоты, соответствующие процессам этого задания, могут быть смещены генетическими операторами.

Пусть i-ое задание должно быть завершено до момента начала j-го задания, а k-ое задание должно начать выполняться не ранее момента завершения j-го задания. Тогда границы коридора мутации для j-го задания задаются как $t^{\min}_{jm\alpha} = t_{im\alpha} + |s_{im\alpha}|$ и $t^{\max}_{jm\alpha} = t_{km\alpha}$, где $t_{im\alpha}$ и $t_{km\alpha}$ — моменты начала выполнения i-го и k-го заданий, а $|s_{im\alpha}|$ — длина слота i-го задания, $m \in M_\alpha$. Данный коридор используется при создании хромосом начальной популяции.

В алгоритмах генетических операторов также приходится проверять, могут ли слоты $s_{im\alpha}$ и $s_{jn\alpha}$ быть заменены друг на друга в расписании без нарушения коридоров мутации, $m, n \in M_\alpha$, $i, j \in I_\alpha$. Обозначим через $t_{im\alpha}$ и $t_{jn\alpha}$ — моменты начала слотов $s_{im\alpha}$ и $s_{jn\alpha}$, а $t^{\max}_{jm\alpha}$ и $t^{\max}_{in\alpha}$ — моменты завершения коридоров мутации этих слотов для ВУ ϕ_m и ϕ_n соответственно.

Тогда данные слоты могут быть заменены друг на друга в хромосоме X_α, если выполняются следующие условия:

$$s_{in\alpha} \subseteq [t_{jn\alpha}, t^{\max}_{in\alpha}], \quad s_{jm\alpha} \subseteq [t_{im\alpha}, t^{\max}_{jm\alpha}]. \tag{2.2}$$

2.2.4. Учет ресурсных ограничений

Под ресурсными ограничениями понимаются правила, определяющие возможность процесса выполняться на данном ВУ. При составлении расписания для РВС такими ограничениями являются: объем оперативной памяти, запрашиваемый для задания на вычислительном узле; соответствие архитектуры процессора узла архитектуре процессора, для которого предназначено задание, и т. д.

Важная задача при применении ГА к составлению расписания для РВС — разработка механизма учета указанных ограничений. Представляемый алгоритм является ресурсно-зависимым — т. е. учитываются ограничения (1.2) на ресурсы, необходимые заданиям. Его идея заключается в сокращении пространства выбора ВУ перед применением генетических операторов.

Используется дополнительная структура данных — маска ресурсов. Алгоритм создания маски ресурсов для слота $s^*_{im\alpha} \in X_\alpha$ приведен ниже, здесь $m \in M_\alpha$, M_α — множество номеров ВУ, которым соответствуют слоты в расписании, кодируемом хромосомой X_α, $i \in I_\alpha$, I_α — множество номеров слотов из хромосомы X_α.

1. Обозначим через $B^e_{i\alpha} = \{b^e_{in\alpha} : n \in M\}$ булевый вектор, значение компоненты $b^e_{in\alpha}$ которого равно 1, если для номера ресурса e, ВУ ϕ_n и слота $s^*_{im\alpha}$ выполняется неравенство (1.2), и $b^e_{in\alpha} = 0$ — в противном случае. Элемент $b^e_{in\alpha}$ данного вектора устанавливает возможность сопоставить слот $s^*_{im\alpha}$ с ВУ ϕ_n без нарушения ресурсного ограничения для ресурса номер e.

2. Вычисляется булевый вектор $B_{i\alpha}$, устанавливающий возможность сопоставить слот $s^*_{im\alpha}$ с тем или иным ВУ во всей РВС с учетом всех

ресурсных ограничений:

$$B_{i\alpha} = \left\{ b_{im\alpha} = \bigwedge_{e=1}^{N_r} b^e_{im\alpha} : \ m \in M \right\}, \ b^e_{im\alpha} \in B^e_{i\alpha},$$

где $\bigwedge$ — логический оператор коньюкции, а N_r — число всех типов ресурсов в РВС. ■

В результате элемент $b_{im\alpha} \in B_{i\alpha}$ определяет следующую возможность:

$$\text{если } b_{im\alpha} = \begin{cases} 1, & \text{то слот } s^*_{im\alpha} \text{ может быть сопоставлен с ВУ } \phi_m, \\ 0, & \text{то слот } s^*_{im\alpha} \text{ не может быть сопоставлен с ВУ } \phi_m. \end{cases}$$

Описанный механизм учета ресурсных ограничений позволяет принимать во внимание и другие ограничения, не рассматриваемые в данной работе, например, ограничения доступа заданий к ресурсам в РВС по определенным администратором организационным правилам.

2.2.5. Создание начальной популяции

Для поиска расписания выполнения *непараллельных* заданий хромосомы начальной популяции можно создавать случайным образом. Но для увеличения эффективности алгоритма вероятность $p_{in\alpha}$ выбора ВУ ϕ_n для сопоставления очередного слота $s_{in\alpha}$ может вычисляться по следующей формуле:

$$p_{in\alpha} = 1 - w_n \left(\sum_{m=0}^{M_{i\alpha}} w_m \right)^{-1}, \tag{2.3}$$

где $M_{i\alpha}$ — число ВУ, для которых $b_{im\alpha} = 1$, $b_{im\alpha} \in B_{i\alpha}$, $n \in M$, а w_n — загруженность ВУ ϕ_n, которая вычисляется следующим образом:

$$w_n = \sum_{i=0}^{N_{n\alpha}} |s_{in\alpha}|,$$

где $N_{n\alpha}$ — число слотов, сопоставленных с ВУ ϕ_n в хромосоме X_α, а $|s_{in\alpha}|$ — длина слота $s_{in\alpha}$.

Алгоритм заполнения хромосомы X_α начальной популяции при поиске расписания *непараллельных* заданий совпадает с алгоритмом *FCFS*

с одним отличием: при выборе очередного ВУ необходимо использовать вероятность выбора по формуле (2.3) согласно маске ресурсов.

Для создания хромосом начальной популяции применялся модифицированный алгоритм обратного заполнения (Backfill), часто используемый при составлении расписаний как для вычислительных кластеров [34], так и для РВС типа Grid [11] и на сегодня эталонный при составлении расписания параллельных заданий для РВС.

Входными данными алгоритма являются: $V = \{v_p : \ p = 1, 2, \ldots, P\}$ — множество заданий, удовлетворяющих условию (1.4), P — число заданий, для которых составляется расписание; Ф — объединение групп ВУ вида (1.3). Результат работы алгоритма — хромосома $X_\alpha \in X$, где X — множество хромосом создаваемой популяции.

Под холостым подразумевается слот, кодирующий отрезок времени простаивания ВУ. Под термином окно в алгоритме обратного заполнения подразумевается множество равных холостых слотов на ВУ одной группы.

Опишем алгоритм обратного заполнения с учетом зависимостей между заданиями и ресурсных ограничений.

1. Элементы $v_p \in V$ сортируются по убыванию приоритетов заданий, образуя новый отсортированный список $V' = \{v_i : \ i = 1, 2, \ldots, P\}$.

2. Пусть хромосома X_α на данном шаге не содержит ни одного слота. Тогда для каждого $i = 1, 2, \ldots, P$ последовательно выполняются следующие действия:

2.1. полагается $v^* = v_i$; если для задания v^* ранее уже были добавлены слоты в хромосому X_α и $i \neq P$, то $i = i+1$ и осуществляется повторное выполнение текущего пункта алгоритма;

2.2. если $i = P$, то работа алгоритма завершается;

2.3. если существует такое задание $v_j \in V'$, что задание v^* может быть запущено не ранее завершения v_j (v^* зависит от v_j), то:

2.3.1. устанавливается нижний временной предел t^* запуска задания v^*:

$$t^* = t_{jm\alpha} + |s_{jm\alpha}|, \tag{2.4}$$

где $t_{jm\alpha}$ и $|s_{jm\alpha}|$ — начало и длина слотов для задания v_j в хромосоме X_α, $m \in M$, M — множество номеров всех ВУ в РВС;

2.3.2. полагается $v^* = v_j$ и осуществляется переход на начало шага 2.3;

2.4. определяется множество окон $W_\alpha = \{w_{k\alpha} : k = 1, 2, \ldots, N_\alpha\}$. Здесь $w_{k\alpha} = \{s_{kn\alpha} : n \in M\}$, где N_α — число окон в расписании, кодируемом хромосомой X_α;

2.5. создается маска ресурсов B^*_α для задания v^*;

2.6. пусть $W^*_\alpha \subseteq W_\alpha$ — множество окон, для N^* слотов $s_{kn\alpha}$ которых $b_{n\alpha} = 1$, где $b_{n\alpha} \in B^*_\alpha$, $k \in K^0_\alpha$, $n \in M^0_\alpha$, K^0_α — множество номеров холостых слотов из X_α, M^0_α — множество номеров ВУ, с которыми сопоставлены эти холостые слоты, N^* — число процессов задания v^*. Тогда случайным образом из W^*_α выбирается окно w^* с минимальным моментом времени начала;

2.7. к хромосоме X_α добавляются слоты для процессов задания v^* с моментом начала, равным началу холостых слотов окна w^*. Длина холостых слотов этого окна уменьшается на длину слотов задания v^* либо эти холостые слоты удаляются из хромосомы X_α, если их длина совпадает с длиной вставляемых слотов. Если на шаге 2.3 по формуле (2.4) был установлен нижний предел планирования t^* для задания v^*, то соответствующие слоты должны начинаться в момент t^*. Для получившихся отрезков времени простаивания ВУ в хромосоме X_α создаются холостые слоты. ■

2.2.6. Оператор скрещивания

Для сохранения непротиворечивости при скрещивании, слоты исключаются из обеих родительских хромосом, чтобы не попасть в дочернюю хромосому повторно. На рис. 2.3 схематически показан процесс “сборки” дочерней хромосомы из двух родительских (у слотов $s_{im\alpha}$ опущен индекс m (номер ВУ)).

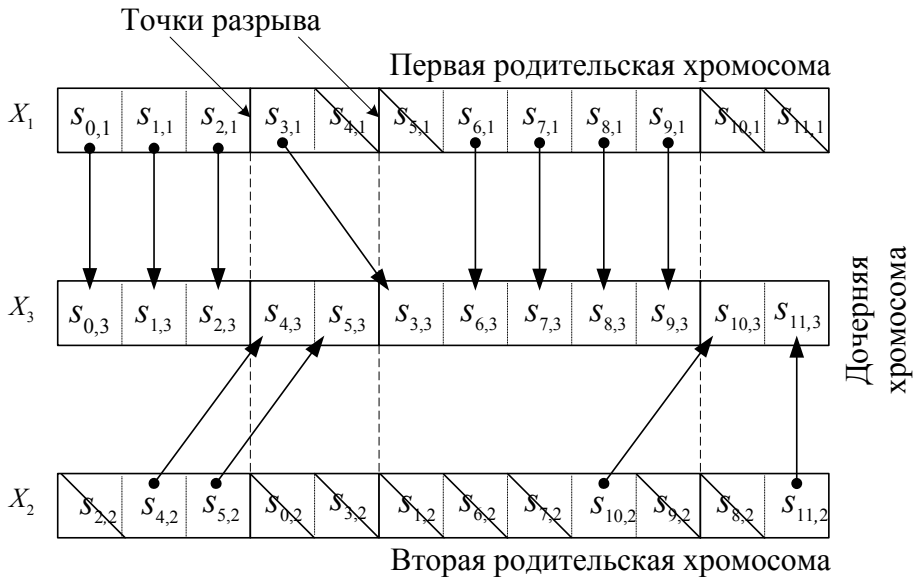

Рис. 2.3. Схема скрещивания двух хромосом

Точки разрыва в случае поиска расписания непараллельных заданий выбираются произвольно. В случае параллельных заданий они выбираются строго между слотами, соответствующими различным группам ВУ. В результате такого выбора точек разрыва исключаются ситуации, когда процессы одного параллельного задания назначаются различным группам ВУ.

Входными данными алгоритма оператора скрещивания являются хромосомы $X_1 \in X$ и $X_2 \in X$, где X — множество хромосом в популяции. В результате работы данного алгоритма создается новая (дочерняя) хромосома X_3. Алгоритм оператора скрещивания следующий.

1. Множества слотов X_1 и X_2 разбиваются на непересекающиеся подмножества

$$X_\beta = \bigcup_{g=1}^{N_g} X_{g\beta}, \quad X_{g\beta} = \{s_{im\beta} : \ m \in M_g, \ i \in I_{g\beta}\}, \quad \beta = 1, 2,$$

где M_g — множество номеров ВУ из g-й группы, $I_{g\beta}$ — множество номеров слотов, сопоставленных с ВУ из g-й группы согласно расписанию, кодируемому хромосомой X_β соответственно, N_g — количество групп виртуальных узлов в РВС.

2. Для каждого номера группы $g = 1, 2, \ldots, N_g$ в хромосому X_3 добавляются слоты из множества $(X_{g1} \backslash X_3)$, если g — четное, либо из множества $(X_{g2} \backslash X_3)$, если g — нечетное. ■

2.2.7. Оператор мутации

Данный оператор осуществляет обмен слотов, что позволяет сохранить неизменным число всех слотов в хромосоме. Автором диссертации разработано два варианта оператора мутации: для составления расписания непараллельных и параллельных заданий.

При составлении расписания выполнения *непараллельных* заданий при однократной мутации два слота в хромосоме меняются местами. При этом длины слотов масштабируются в соответствии с формулой (2.1), так как после обмена соответствующие слоты могут быть сопоставлены ВУ с отличными от прежних характеристиками.

В случае составления расписания выполнения *непараллельных* заданий для увеличения вероятности равномерного распределения процессов по ВУ при выборе первого и второго слота для обмена автор предлагает использовать различные вероятности их выбора. В результате процессы чаще сопоставляются менее загруженным вычислительной работой ВУ.

Для определения вероятностей выбора слотов для обмена представим хромосому в следующем виде. Подмножества слотов хромосомы, относящиеся к одному ВУ, “выравниваются” до максимального числа слотов по всем ВУ путем вставки в конец данного подмножества заполняющих холостых слотов. Такая форма хромосомы дает возможность изменения числа слотов, относящихся к тому или иному ВУ, в то время как при обычном обмене двух слотов, их число на том или ином ВУ не изменяется. На рис. 2.2.7 показан пример подобной формы хромосомы, включающей для наглядности всего два ВУ.

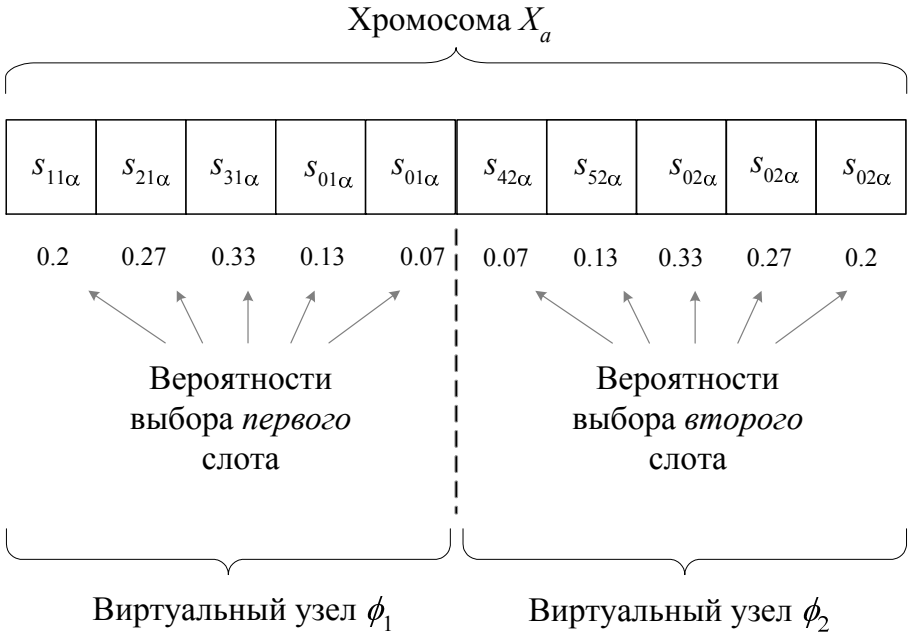

Рис. 2.4. Пример распределения вероятностей выбора слотов для мутации; холостые слоты обозначены как $s_{0m\alpha}$, где $m = 1, 2$

Для упрощения рассуждений предположим, что первый и второй слоты будут выбраны из частей хромосомы, относящихся к ВУ ϕ_1 и ϕ_2 соответственно. При выборе слотов для мутации обменом в данном случае необходимо оперировать двумя различными значениями вероятностей p_{m_1} и p_{m_2}, которые вычисляются для обычных и заполняющих холостых слотов по следующим формулам:

$$p_{m_1} = (\delta_1 + m)/\sigma_n, \qquad p^0_{m_1} = (m + m_0 - \delta_1 + 1)/\sigma_n, \tag{2.5}$$

$$p_{m_2} = \kappa_c \delta_2/\sigma_n \qquad p^0_{m_2} = \kappa_c(2m + m_0 - \delta_2 + 1)/\sigma_n, \tag{2.6}$$

$$\sigma_n = \sum_{n=1}^{m+m_0} n = (m + m_0 + 1)(m + m_0)/2,$$

где m и m_0 — здесь число обычных и заполняющих холостых слотов, δ_1 и δ_2 — порядковые номера следования слотов на ВУ, где произошел первый и второй выбор соответственно, κ_c - нормированный коэффициент производительности, вычисляемый по формуле 2.1. В результате вычисления вероятностей по данным формулам для первого выбора слоты получают возрастающую вероятность для обычных слотов и убывающую для запол-

няющих холостых, в то время как для второго выбора вероятности распределяются обратно первому.

Алгоритм оператора мутации в ГА составления расписания *непараллельных* заданий в РВС для хромосомы $X_\alpha \in X$ следующий.

1. Выбирается некоторый порядковый номер слота $1 \leq j \leq N_\alpha$ с вероятностью, определенной по формулам (2.5), где N_α — длина хромосомы X_α.

2. Создаётся для слота $s_{jm\alpha}$ маска ресурсов $B_{j\alpha} = \{b_{jn\alpha}\}$, $n, m \in M$, j — номер слота, выбранный на предыдущем шаге алгоритма.

3. Находится множество $X'_\alpha \subseteq X_\alpha$ слотов, принадлежащих таким ВУ ϕ_n, для которых $b_{jn\alpha} = 1$, где $b_{jn\alpha} \in B_{j\alpha}$, $n \in M_\alpha$, M_α — множество номеров таких ВУ, с которыми сопоставлены слоты в хромосоме X_α.

4. Если $X'_\alpha = \varnothing$, то работа алгоритма завершается.

5. Для каждого слота $s_{kn\alpha} \in X'_\alpha$ составляется маска ресурсов $B_{k\alpha} = \{b_{kh\alpha}\}$, $n, h \in M$, $1 \leq k \leq N_\alpha$.

6. Определяется множество $Y_\alpha \subseteq X'_\alpha$ слотов, таких, чьи маски ресурсов $B_{k\alpha}$ включают элемент $b_{km\alpha} = 1$, где $m \in M_\alpha$ — конкретный номер ВУ первоначально выбранного слота $s_{jm\alpha}$. Таким образом, Y_α — множество слотов, с которыми выбранный на первом шаге слот $s_{jm\alpha}$ может быть заменен без нарушения определяемых неравенством (1.2) ресурсных ограничений на кодируемое хромосомой X_α расписание.

7. Если $Y_\alpha = \varnothing$, то работа алгоритма завершается.

8. Пусть $\hat{Y}_\alpha \in Y_\alpha$ — множество слотов, соответствующих заданиям, которые зависят от задания слота $s_{jm\alpha}$, а $\check{Y}_\alpha \in Y_\alpha$ — множество слотов, соответствующих заданиям, от которых зависит задание слота $s_{jm\alpha}$. Вычисляется $Y_\alpha = Y_\alpha \setminus \left(\hat{Y}_\alpha \cup \check{Y}_\alpha\right)$.

9. Если $Y_\alpha = \varnothing$, то работа алгоритма завершается.

10. Определяется множество $Z_\alpha \subseteq Y_\alpha$, состоящее только из слотов $s_{in\alpha}$ таких, что для $s_{jm\alpha}$ и $s_{in\alpha}$ выполняются неравенства (2.2).

11. Если $Z_\alpha = \varnothing$, то работа алгоритма завершается.

12. Выбирается некоторый слот $s^* \in Z_\alpha$ с вероятностью, определенной по формулам (2.6).

13. Осуществляется обмен слотом s^* и выбранным на первом шаге слотом $s_{jm\alpha}$ в хромосоме X_α с вычислением новых значений их длин согласно формуле (2.1). ■

Для составления расписания *параллельных* заданий в операторе мутации для обмена выбираются не отдельные слоты, а множества слотов двух заданий. В случае, если новое расположение слотов во времени занято другими слотами, то происходит смещение расписания, как показано на рис. 2.5. Здесь t' — момент времени, по которому будут начинаться перемещаемые оператором мутации слоты.

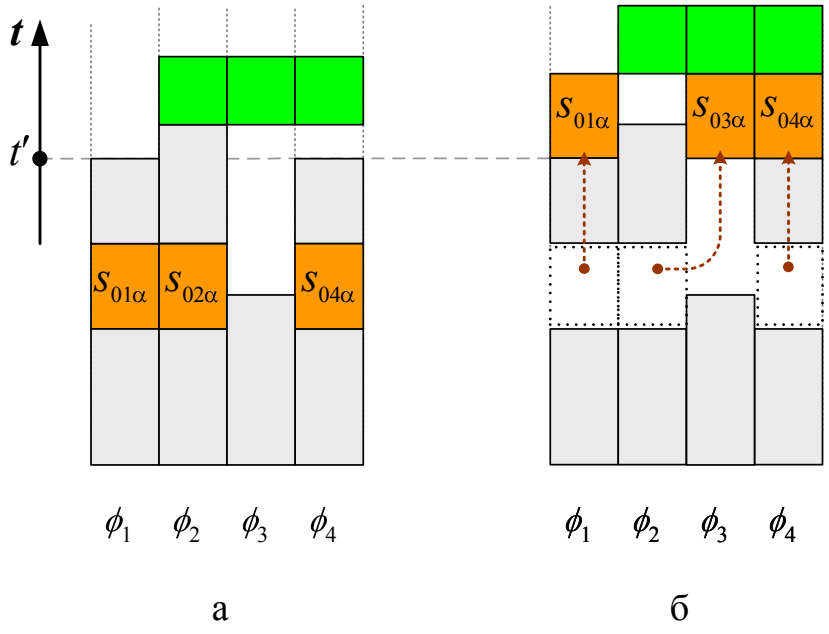

Рис. 2.5. Схема смещения части расписания до (а) и после (б) перемещения слотов оператором мутации при составлении расписания выполнения параллельных заданий

Ниже описан алгоритм оператора мутации в ГА составления расписания выполнения параллельных заданий в РВС, разбитый на две части. В

первой части рассмотрена последовательность действий для выбора слотов двух заданий, во второй — описываются действия по обмену слотов этих заданий.

Часть первая. Алгоритм оператора мутации применяется к хромосоме $X_\alpha \in X$.

1. Создается маска ресурсов $B_{j\alpha} = \{b_{jn\alpha} : \ n \in M\}$ для слота $s_{jm\alpha}$, где номер $j \in I_\alpha$ слота выбирается случайным образом, I_α — множество номеров слотов, которые присутствуют в хромосоме X_α, M — множество номеров всех виртуальных узлов в РВС.

2. Определяется множество $X'_\alpha \subseteq X_\alpha$ слотов, принадлежащих ВУ ϕ_n, для которых $b_{jn\alpha} = 1$, где $b_{jn\alpha} \in B_{j\alpha}$, $n \in M$.

3. Если $X'_\alpha = \varnothing$, то работа алгоритма завершается.

4. Для всех слотов из множества X'_α создаются маски ресурсов $B_{k\alpha} = \{b_{kh\alpha} : \ h \in M\}$, где $k = 1, 2, \ldots, N'_\alpha$, N'_α — число слотов из множества X'_α.

5. Определяется подмножество $Y_\alpha \subseteq X'_\alpha$ слотов, чьи маски ресурсов $B_{k\alpha}$ включают элемент $b_{km\alpha} = 1$, где m — номер ВУ первоначально выбранного слота $s_{jm\alpha}$. Таким образом, Y_α — подмножество слотов, на которые выбранный на первом шаге слот $s_{jm\alpha}$ может быть заменен без нарушения определяемых неравенством (1.2) ресурсных ограничений.

6. Если $Y_\alpha = \varnothing$, то работа алгоритма завершается.

7. Пусть $Y_{1\alpha} \subset Y_\alpha$ — подмножество слотов заданий, которые должны быть завершены до начала слота $s_{jm\alpha}$, $Y_{2\alpha} \subset Y_\alpha$ — подмножество слотов заданий, которые должны начинаться не ранее момента завершения слота $s_{jm\alpha}$. Определяется подмножество $Y'_\alpha = Y_\alpha \setminus (Y_{1\alpha} \bigcup Y_{2\alpha})$.

8. Если $Y'_\alpha = \varnothing$, то работа алгоритма завершается.

9. Определяется подмножество $Z_\alpha \subseteq Y'_\alpha$, состоящее только из таких слотов $s_{in\alpha}$, что для $s_{jm\alpha}$ и $s_{in\alpha}$ соблюдаются условия коридора мутации (2.2).

10. Если $Z_\alpha = \varnothing$, то работа алгоритма завершается.

11. Пусть $X_{1\alpha} \subset X_\alpha$ — подмножество слотов, соответствующих процессам того же задания, что и слот $s_{jm\alpha}$. Пусть $N_{1\alpha}$ — число слотов из множества $X_{1\alpha}$, а $N'_{i\alpha}$ — число процессов в задании, которому принадлежит соответствующий слоту $s_{im\alpha} \in Z_\alpha$ процесс, M_j — число ВУ в группе,

соответствующей слоту $s_{jm\alpha}$, а M'_i — число ВУ в группах $\Phi_i \in \Phi$, соответствующих слотам $s_{im\alpha} \in Z_\alpha$. Определяется подмножество Z'_α слотов $s_{in\alpha} \in Z_\alpha$, для которых $N_{1\alpha} \leq M'_i$ и $N'_{i\alpha} \leq M_j$.

12. Если $Z'_\alpha = \varnothing$, то работа алгоритма завершается.

13. Выбирается случайным образом слот $s^* \in Z'_\alpha$.

Первая часть алгоритма позволяет выбрать два слота: $s_{jm\alpha} \in X_\alpha$ (далее — слот первого задания) и $s^* \in Z'_\alpha$ (далее — слот второго задания) для мутации обменом без нарушения ограничений, налагаемых на расписание выполнения параллелных заданий в РВС. Во второй части алгоритма осуществляется такое переупорядочивание слотов хромосомы X_α, которое заменяет одно множество слотов (слоты первого задания) на другое (слоты второго задания) и наоборот.

Часть вторая. Пусть $X_{1\alpha}$ и $X_{2\alpha}$ — множества слотов, соответствующих первому и второму выбранным в первой части описываемого алгоритма заданиям, $B_{1\alpha}$ и $B_{2\alpha}$ — маски ресурсов для процессов этих заданий, а $t_{1\alpha}$ и $t_{2\alpha}$ — моменты времени начала слотов из множеств $X_{1\alpha}$ и $X_{2\alpha}$ соответственно в расписании, кодируемом хромосомой X_α.

1. Определяется такое подмножество $X'_{1\alpha} \subseteq X_\alpha$ слотов $s'_{im\alpha}$, что $s_{1m\alpha} \subseteq s'_{im\alpha}$ для всех таких m, что $b_{2m\alpha} = 1, b_{2m\alpha} \in B_{2\alpha},\ m \in M,\ i \in I'_{1\alpha}$, где M — множество номеров всех ВУ в РВС, $I'_{1\alpha}$ — множество номеров слотов из подмножества $X'_{1\alpha}$.

2. Определяется подмножество слотов $X'_{2\alpha} \subseteq X_\alpha$ аналогично предыдущему шагу алгоритма.

3. Случайным образом выбираются $N_{1\alpha}$ неповторяющихся слотов $s'_{2m\alpha} \in X'_{2\alpha}$, где $N_{1\alpha}$ — число слотов, соответствующих первому заданию. Пусть $t_{1n\alpha}$ и $t'_{2m\alpha}$ — начала слотов $s_{1n\alpha} \in X_{1\alpha}$ и $s'_{2m\alpha}$ соответственно, а $|s_{1n\alpha}|$ — длина слота $s_{1n\alpha}$. Для всех $s_{1n\alpha} \in X_{1\alpha}$ выполняются следующие действия:

3.1. слот $s_{1n\alpha}$ сопоставляется с ВУ ϕ_m, где ϕ_m — ВУ, с которым сопоставлен слот $s'_{2m\alpha}$;

3.2. вычисляется новое значение начала слота $s_{1m\alpha}$: $t_{1m\alpha} = t'_{2m\alpha}$;

3.3. вычисляется новое значение начала слота $s'_{2m\alpha}$: $t'_{2m\alpha} = t_{1m\alpha} + |s_{1m\alpha}|$;

3.4. пусть $s^* = s'_{2m\alpha}$, $\phi^* = \phi_m$, а J^* — пустое множество. Тогда выполняются следующие действия;

3.4.1. в J^* добавляются номера слотов, которые соответствуют ВУ ϕ^* и начинаются позже слота s^*;

3.4.2. в J^* добавляются номера слотов $s_{jn\alpha} \in X_\alpha$, принадлежащих заданиям, которые соответствуют добавленным на предыдущем шаге слотам, $j \in I_\alpha$, $n \in M$;

3.4.3. для каждого слота $s_{jn\alpha}$, номер которого был добавлен в J^* на предыдущем шаге, рекурсивно выполняются шаги 3.4.1–3.4.3 ($s^* = s_{jn\alpha}$, $\phi^* = \phi_n$);

3.5. $\forall j \in J^*$ вычисляются новые значения начала слотов $s_{jm\alpha} \in X_\alpha$: $t_{jm\alpha} = t_{jm\alpha} + |s_{1m\alpha}|$.

4. Производятся аналогичные предыдущему шагу алгоритма вычисления для слотов, принадлежащих подмножеству $X_{2\alpha}$.

5. Так как переупорядочивание слотов множеств $X_{1\alpha}$ и $X_{2\alpha}$ в хромосоме X_α может привести к смещению подмножества слотов в расписании, то образованные в результате этого отрезки времени простаивания ВУ кодируются в хромосоме X_α холостыми слотами. ■

В данном варианте оператора мутации возможны частые вставки слотов в хромосому. Для увеличения производительности алгоритма оператора мутации необходимо минимизировать число подобных вставок. В программной реализации данного оператора предварительно вычисляются все перемещения и вставки слотов и только после этого все слоты хромосомы копируются в новый участок памяти и попутно производятся необходимые замены и вставки. Данная техника реализована в разработанной автором системе планирования, описанной в третьей главе диссертации, и существенно экономит машинное время, которое может тратиться на лишние операции копирования.

2.2.8. Оператор уплотнения

В результате переупорядочивания слотов в хромосоме оператором мутации могут возникать множества холостых слотов. Часть хромосом в популяции после мутации можно сжимать, уменьшая длину холостых слотов

или исключая эти слоты, если их длину можно уменьшить до нуля. На рис. 2.6 показана схема сжатия расписания в алгоритме оператора уплотнения. На данном рисунке в расписании, полученном в результате действия оператора мутации (см. рис. 2.5.б), моменты времени начал подмножества слотов изменяются на меньшие.

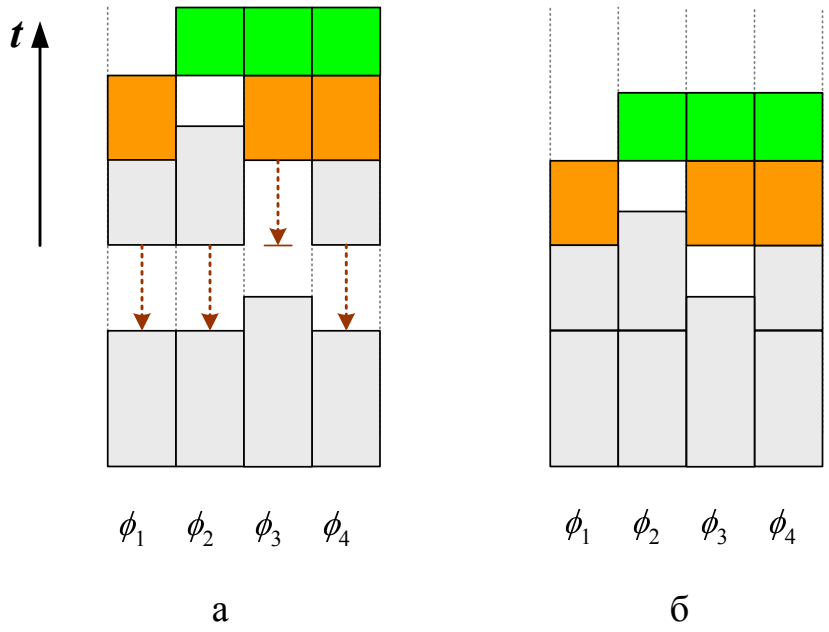

Рис. 2.6. Схема работы оператора уплотнения до (а) и после (б) сжатия расписания

Входным параметром алгоритма оператора уплотнения является хромосома $X_\alpha \in X$. Оператор уплотнения описывается в следующем виде.

1. Пусть $X_{p\alpha} = \{s_{pm\alpha} : \ m \in M_{p\alpha}\}$ — множество слотов задания $v_{p\alpha}$ из хромосомы X_α, а $M_{p\alpha}$ — множество номеров ВУ, с которыми сопоставлены слоты этого задания из X_α. Определяется множество $V_\alpha = \{v_{p\alpha} : \ p \in P\}$, где P — множество таких номеров заданий, что для каждого их слота выполняется следующее равенство:

$$t_{pm\alpha} = t^0_{im\alpha} + |s^0_{im\alpha}|, \quad m \in M, \quad i \in I^0_\alpha, \tag{2.7}$$

где $t^0_{im\alpha}$ и $|s^0_{im\alpha}|$ — начало и длина холостого слота $s^0_{im\alpha} \in X_\alpha$, $t_{pm\alpha}$ — момент начала задания $v_{p\alpha}$ на ВУ ϕ_m, I^0_α — множество номеров холостых слотов из хромосомы X_α, M — множество всех ВУ в РВС. Холостые слоты, удовлетворяющие равенству (2.7) формируют множество Y_α.

2. $\forall v_{p\alpha} \in V_\alpha$: если $t^{\min}_{pm\alpha} \le t^0_{pm\alpha}$, где $t^{\min}_{pm\alpha}$ — начало коридора допустимых значений для $v_{p\alpha}$, создаётся множество Y'_α из слотов $s^0_{pm\alpha}$. Иначе все слоты $s^0_{pm\alpha} \in Y_\alpha$ заменяются в хромосоме X_α двумя слотами $s^0_{1pm\alpha}$ и $s^0_{2pm\alpha}$: $|s^0_{1pm\alpha} \bigcap s^0_{2pm\alpha}| = 0$, $s^0_{1pm\alpha} \bigcup s^0_{2pm\alpha} = s^0_{pma}$, и создаётся множество Y'_α, состоящее из слотов $s^0_{2pm\alpha}$, которые начинаются раньше соответствующих слотов $s^0_{1pm\alpha}$.

3. Выбирается случайное целое число j, $1 \le j \le N_\alpha$, где N_α — число элементов из множества V_α. Тогда $M_{j\alpha} \subseteq M$ — подмножество номеров ВУ, с которыми сопоставлены слоты для задания $v_{j\alpha} \in V_\alpha$.

4. Определяется подмножество $V'_\alpha \subset V_\alpha$ заданий, слоты которых *не* сопоставлены с ВУ из Φ_j.

5. Пусть $Z_{j\alpha} = \{s_{jm\alpha} : \ j \in J_\alpha, \ m \in M_{j\alpha}\}$ — множество слотов для задания $v_{j\alpha}$, где J_α — множество номеров слотов из X_α, соответствующих этому заданию. Тогда для всех слотов $s_{jm\alpha} \in Z_{j\alpha}$: слоты $s_{jm\alpha}$ и $s^0_{jm\alpha} \in Y'_\alpha$ обмениваются моментами времени их начала.

6. $\forall v_{i\alpha} \in V'_\alpha$: для всех слотов, соответствующих заданию $v_{i\alpha}$ выполняются действия, аналогичные описанным на предыдущем шаге алгоритма.

7. Осуществляется переход на шаг 1. ■

В результате работы данного алгоритма возможно увеличение числа слотов в хромосоме ввиду того, что при перемещении слота в новую позицию в расписании холостой слот в данной позиции может быть заменен двумя холостыми слотами (см. п. 2 алгоритма уплотнения). В программной реализации данного оператора (также как и в операторе мутации) предварительно вычисляются все перемещения слотов. И только после этого все слоты хромосомы копируются в новый участок памяти, и попутно производятся необходимые замены слотов и вставки новых.

2.3. Параметры генетических алгоритмов

Алгоритм был программно реализован на языке программирования *C++* в двух версиях: для составления расписания параллельных и непараллельных заданий. Проведены численные эксперименты с целью настройки основных параметров: вероятности применения оператора мутации p_m, ве-

роятности применения оператора скрещивания p_c, размера популяции Z и процента элитных хромосом E в популяции.

Для формирования очередей использованы задачи из набора тестов NAS Parallel Benchmark [57] (NPB v.3.3). Тесты NPB состоят из ряда задач, являющихся фрагментами реальных приложений, требующих интенсивных параллельных вычислений. Каждый из тестов NPB может производить вычисления в нескольких классах сложности. В таблице 2.1 представлены использованные нами в численных экспериментах задачи из NPB для классов сложности "W", "A", "B", "C" и количества процессов 4 и 2 при применении технологии параллельных вычислений MPI.

Таблица 2.1. Тесты NPB с их характеристиками "оперативная память / время решения"

	W		A		B		C	
Тест	4	2	4	2	4	2	4	2
bt	6/2	8/4	35/45	50/80	100/190	200/340	420/750	80/1375
cg	7/1	11/1	15/1	30/2	110/60	220/80	300/170	600/220
ep	3/1	3/2	3/8	3/13	3/27	3/55	3/105	3/210
ft	10/1	20/1	115/4	230/6	450/55	900/80	-	-
is	3/1	6/1	25/1	50/1	100/3	200/4	400/5	770/12
lu	6/5	8/8	17/34	35/56	50/175	100/327	190/1160	360/1400
mg	20/1	35/1	120/3	230/4	115/12	230/14	880/105	1700/115
sp	10/8	15/12	26/75	50/100	100/360	330/430	350/1376	700/1680

В данной таблице указаны параметры, учитывающиеся алгоритмом составления расписаний: объем оперативной памяти в МБ и время счета в секундах. Данные параметры получены на ЭВМ с процессором Intel Core 2 Quad 6600 2.4 ГГц, ОС Linux CentOS 4.

Количество заданий в очереди может быть различным, поэтому программа составления расписаний должна надежно работать при любом их числе. Сначала изложим результаты проведенных экспериментов для составления расписания непараллельных заданий для различного количества процессов всех заданий N.

Для исследования алгоритма из табл. 2.1 были выбраны задания случайным образом так, чтобы общее число процессов достигало необходимого значения N. Точное решение задачи составления расписания методом перебора получить за приемлемое время невозможно, поэтому в дальнейшем принималось во внимание наилучшее решение, которое было получено в

результате проведенных экспериментов.

Рассматривалась РВС из пяти вычислительных кластеров, в общей сложности включающих 100 ВУ. Каждый кластер содержал по 20 ВУ, включающих процессор Intel Xeon, 2 ГБ оперативной памяти. Тактовые частоты процессоров на кластерах брались следующие (в ГГц): 1.8, 2.2, 2.8, 3.0, 3.2.

Численное решение рассматриваемых в разделе задач считалось найденным, если значение функции пригодности найденного расписания отличалось от наилучшего не более чем на 5%. Осуществлось 30 запусков поиска расписания с фиксированными параметрами ГА. Значение числа процессов всех заданий в расписании варьировалось от 600 до 1600 с шагом 200. Вычислялся коэффициент k_p ускорения поиска расписания генетическим алгоритмом:

$$k_p = \overline{T}_p \left[\min_{1 \le j \le J} \overline{T}_{pj} \right]^{-1}, \tag{2.8}$$

где $\overline{T}_p$ — среднее по 30 запускам время поиска расписания с текущим значением исследуемого параметра, $\overline{T}_{pj}$ — среднее по 30 запускам время поиска с j-ым значением исследуемого параметра, J — здесь число всех значений исследуемого параметра. Данный коэффициент позволяет наглядно оценить ускорение поиска расписания, т.к. одно из значений коэффициента на графике при каждом значении N всегда будет равно 1.

Получены зависимости коэффициента укорения k_p работы программы от значения уровня мутации p_m (рис. 2.7).

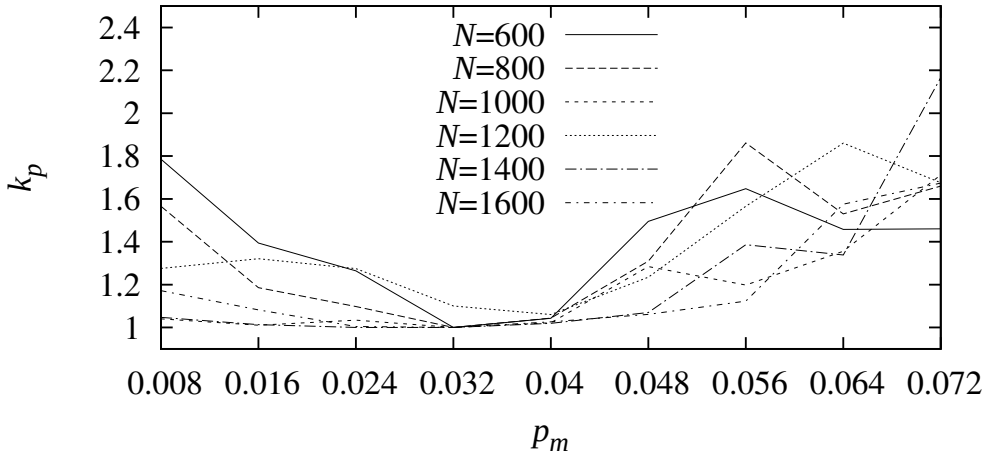

Рис. 2.7. Зависимость коэффициента k_p ускорения нахождения расписания непараллельных заданий от процента мутации p_m при числе всех процессов заданий N

Было установлено, что лучшего результата по времени алгоритм достигает в диапазоне значения p_m от 0.032% до 0.04%, остальные параметры: $p_c = 20\%$, $E = 10\%$, $Z = 30$. Явной зависимости k_p от числа процессов N на графике не наблюдается.

На рис. 2.8 показана зависимость коэффициента укорения k_p работы программы от значения процента скрещивания p_c. Число точек скрещивания — 4.

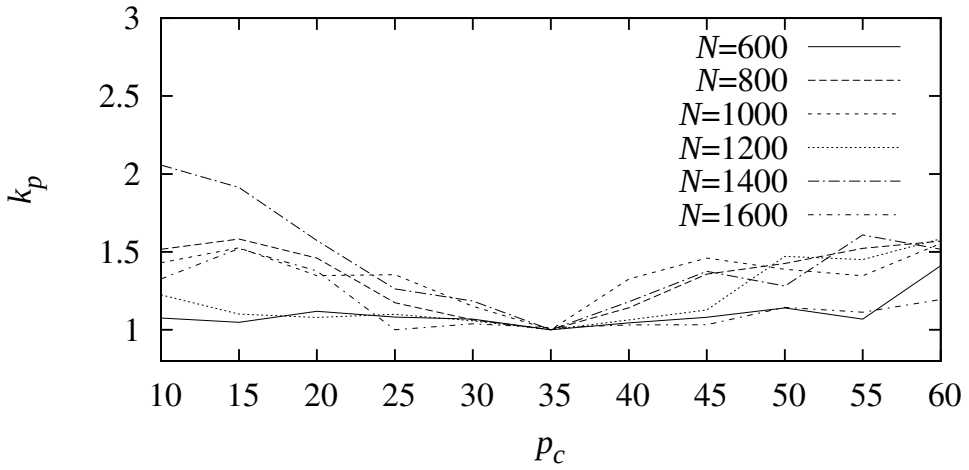

Рис. 2.8. Зависимость коэффициента k_p ускорения нахождения расписания непараллельных заданий от процента скрещивания p_c при числе всех процессов заданий N

Из графика видно, что значение процента скрещивания p_c оказывает меньшее влияние на ГА, чем p_m. В целом, наилучшие результаты получены при значении p_c в диапазоне от 25% до 35%, остальные параметры: $p_m = 0.032\%$, $E = 10\%$, $Z = 30$. Зависимости k_p от числа процессов N не наблюдается.

Важным параметром является число элитных хромосом E в популяции. На рис. 2.9 показана зависимость коэффициента ускорения алгоритма от процента элитных особей E в популяции.

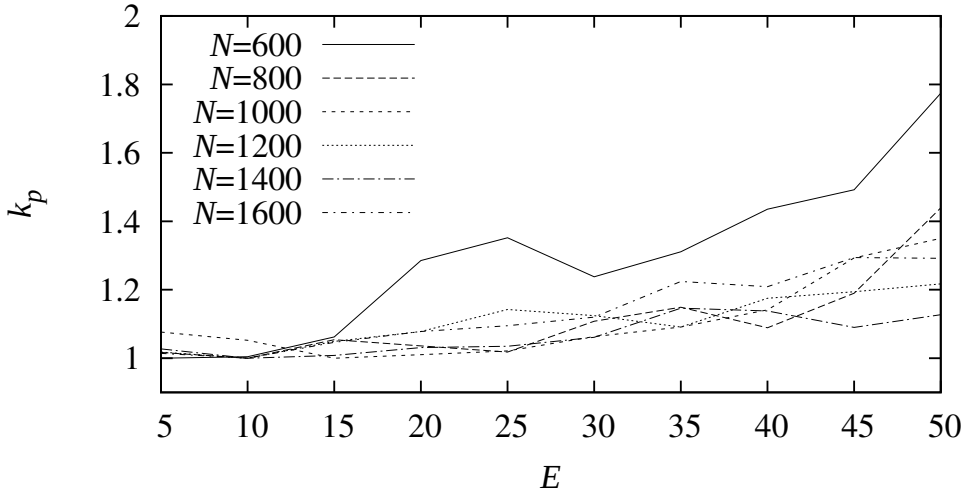

Рис. 2.9. Зависимость коэффициента k_p ускорения нахождения расписания непараллельных заданий от процента элитных особей в популяции E при числе всех процессов заданий N

Из графика видно, что высокие значения E не способствуют нахождению расписания непараллельных заданий. Лучшие результаты достигаются при $E = 10\%$ (в данном случае — 3 особи), остальные параметры: $p_m = 0.032\%$, $p_c = 30\%$, $Z = 30$.

На следующем рисунке изображена зависимость коэффициента ускорения алгоритма от размера популяции Z.

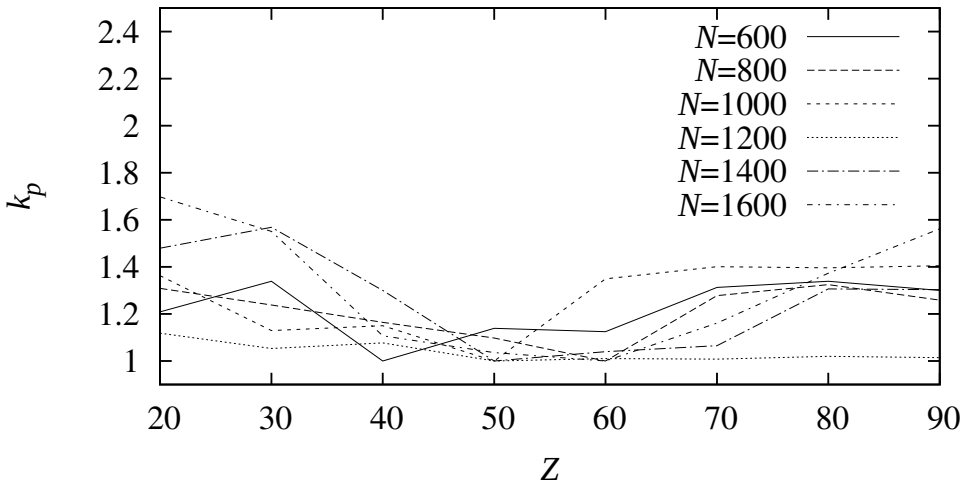

Рис. 2.10. Зависимость коэффициента k_p ускорения нахождения расписания непараллельных заданий от размера популяции Z при числе всех процессов заданий N

Лучшей скорости нахождения решения алгоритм при данной конфигурации достигал при значении Z в диапазоне от 40 до 60 особей, остальные параметры: $p_m = 0.032\%$, $p_c = 30\%$, $E = 10\%$.

Аналогичные испытания проведены для ГА поиска расписания параллельных заданий. Вычислялся коэффициент ускорения k_p по формуле (2.8). Число прогонов для фиксированного сочетания параметров равнялось 30. Задания брались из табл. 2.1.

На следующем рисунке показана полученная в ходе испытаний зависимость коэффициента ускорения поиска расписания параллельных заданий от процента мутации p_m.

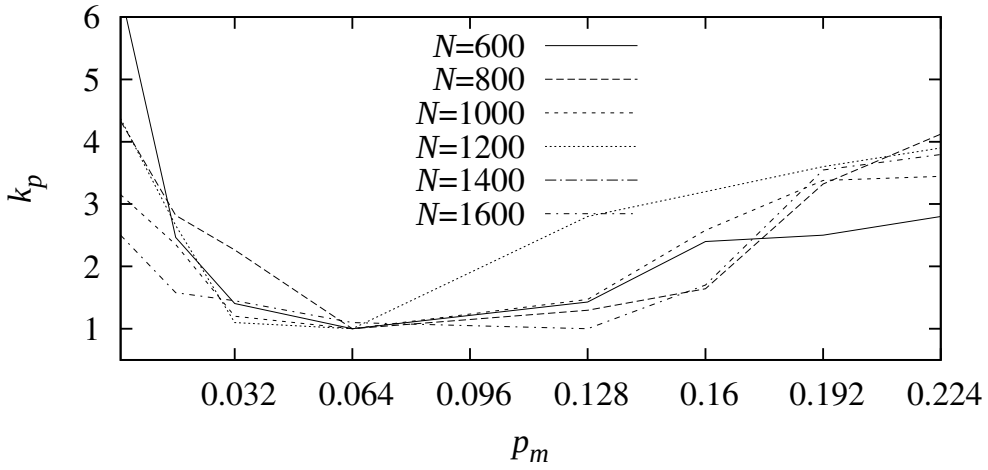

Рис. 2.11. Зависимость коэффициента k_p ускорения нахождения расписания параллельных заданий от процента мутации p_m при числе всех процессов заданий N

Из рис. 2.11 (а также из рис. 2.7) видно, что значения k_p, при которых алгоритмы поиска расписания непараллельных и параллельных заданий достигают лучших показателей, различны. Так, в случае поиска расписания параллельных заданий, значение p_m необходимо брать примерно вдвое большее, чем в случае непаралелльных заданий: лучшей скорости нахождения решения алгоритм достигал при значении p_m в диапазоне от 0.064% до 0.096%, остальные параметры $p_c = 20\%$, $E = 10\%$, $Z = 30$.

На следующем рисунке показана зависимость коэффициента ускорения поиска расписания параллельных заданий от процента скрещивания p_c.

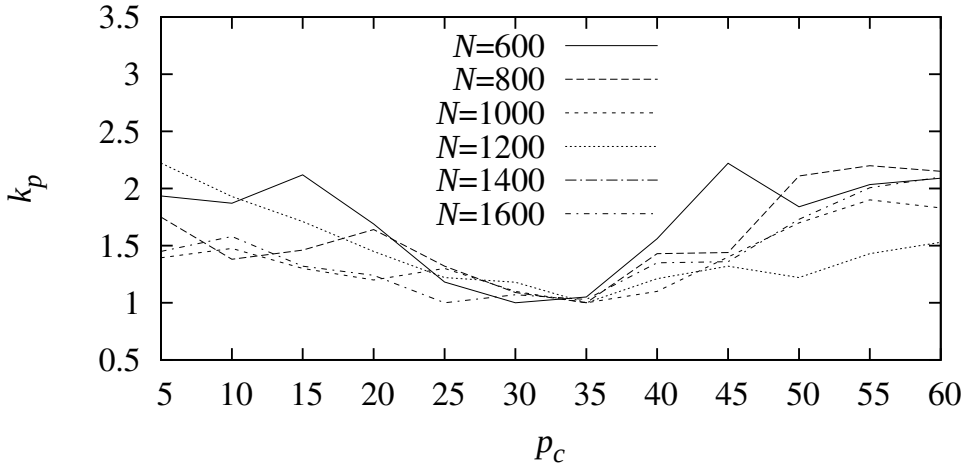

Рис. 2.12. Зависимость коэффициента k_p ускорения нахождения расписания параллельных заданий от процента скрещивания p_c при числе всех процессов заданий N

Из рис. 2.12 видно, что в случае с изменением процента скрещивания p_c генетический алгоритм поиска расписания паралельных заданий ведет себя схожим образом с ГА поиска расписания непараллельных заданий. Лучшие значения в данном случае достигаются при p_c в диапазоне 30% - 35%, остальные параметры $p_m = 0.065\%$, $E = 10\%$, $Z = 30$.

На рис. 2.13 показана зависимость коэффициента ускорения алгоритма от процента элитных особей E в популяции.

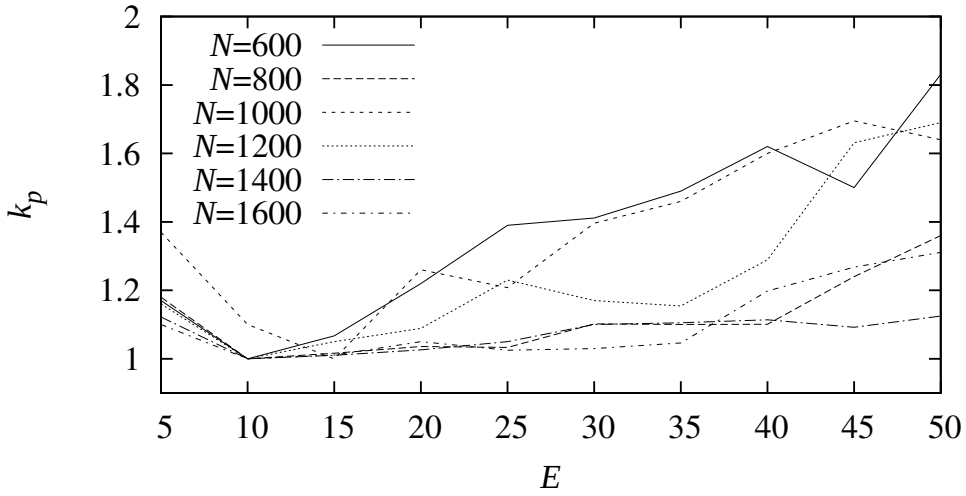

Рис. 2.13. Зависимость коэффициента k_p ускорения нахождения расписания параллельных заданий от процента элитных особей в популяции E при числе всех процессов заданий N

Из данного рисунка можно судить о том, что поведение тестируемых алгоритмов примерно совпадает при изменении значения параметра E. Экспериментально установлено (рис. 2.13), что процент элитных хромосом для предложенного ГА при решении рассматриваемой задачи должно быть небольшим: $E = 10\%$, остальные параметры $p_m = 0.065\%$, $p_c = 30\%$, $Z = 30$.

На следующем рисунке изображена зависимость коэффициента ускорения поиска расписания параллельных заданий от размера популяции Z.

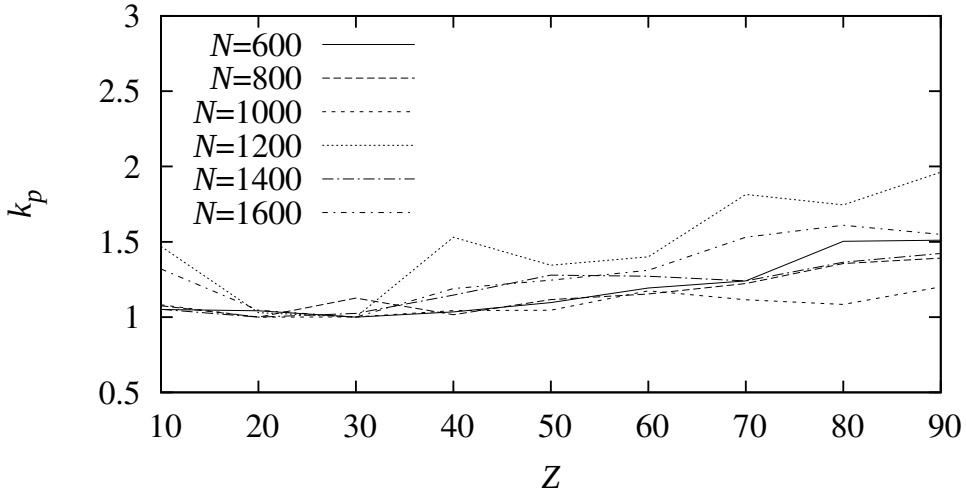

Рис. 2.14. Зависимость коэффициента k_p ускорения нахождения расписания параллельных заданий от размера популяции Z при числе всех процессов заданий N

Из рис. 2.14 видно, что лучшей скорости нахождения решения алгоритм достигает при значении Z в диапазоне от 20 до 30 особей, что примерно в 2 раза меньше, чем при поиске расписания запуска непараллельных заданий. Остальные параметры: $p_m = 0.065\%$, $p_c = 30\%$, $E = 10\%$.

Из полученных результатов численных экспериментов для задач различной сложности видно, что значения p_c и E, при которых реализации алгоритмов поиска расписания непараллельных и параллельных заданий находят приемлемые решения за наименьшее время, примерно совпадают. Значения p_m и Z, при которых алгоритмы показывают лучшие результаты, различны. Такое поведение можно объяснить существенно различными алгоритмами оператора мутации в данных ГА.

2.4. Сравнение генетического алгоритма составления расписания параллельных заданий с алгоритмом обратного заполнения

Алгоритм обратного заполнения (Backfill) широко применяется на практике для составления расписания выполнения параллельных заданий. Важно было проверить эффективность работы разработанного алгоритма планирования выполнения параллельных заданий с таким общеиспользуемым алгоритмом, как *Backfill*. Исследовались такие параметры, как выигрыш в загруженности w и сэкономленное время простаивания Δ вычислительных ресурсов.

Загруженность w_j — процентное соотношение времени всех ВУ под нагрузкой выполнения заданий к максимально возможному уровню загруженности для заданной длины расписания в поколении j генетического алгоритма:

$$w_j = \frac{1}{G l_j^{\max}} \sum_{m \in M} l_{mj}, \quad l_j^{\max} = \max_{1 \le n \le G} l_{nj}, \tag{2.9}$$

где l_{mj} — длина подрасписания для ВУ $\phi_m \in \Phi$ в поколении j, G — число всех ВУ в РВС, M — множество номеров всех виртуальных узлов в РВС.

Рассчитывалось значение разности между загруженностью вычислительных ресурсов, полученное в результате составления расписания с помощью алгоритма обратного заполнения (w_0) и загруженностью ресурсов согласно оптимизированному расписанию с помощью ГА (w_I): $w_I - w_0$, где I — последнее поколение ГА.

Среднее время простаивания вычислительных ресурсов Δ_j в поколении j генетического алгоритма есть величина, показывающая среднее время ожидания всех ВУ в РВС после выполнения всех запланированных расписанием процессов до момента времени, соответствующего длине расписания:

$$\Delta_j = \frac{1}{G} \sum_{m \in M} \left(l_j^{\max} - l_{mj} \right), \tag{2.10}$$

где l_{mj} — длина подрасписания на ВУ $\phi_m \in \Phi$ в поколении j. Тогда сэкономленное время простаивания есть разность $(\Delta_0 - \Delta_I)$, где Δ_0 — среднее время простаивания ресурсов в лучшем расписании, сгенерированным

алгоритмом обратного заполнения для популяции номер 0, Δ_I — среднее время простаивания ресурсов в лучшем расписании в последнем поколении ГА.

В тестировании использовались конфигурации РВС, включающие по два вычислительных кластера. Число ВУ в данных кластерах варьировалось от 64 до 512. Попарно перебирая данные значения. Были проведены тестовые оптимизации расписаний генетическим алгоритмом, изначально полученные алгоритмом обратного заполнения. Данный алгоритм применяется при создании начальной популяции, таким образом значения $(w_I - w_0)$ и $(\Delta_0 - \Delta_I)$ всегда были положительными.

Рассматривались задания случайной длительности от 12 до 168 часов, включающие случайное число параллельных процессов от 1 до 32. Перед численным экспериментом генерировались указанные задания в очереди до тех пор, пока общее число процессов всех заданий не достигнет необходимого (N). Осуществлялось 30 запусков алгоритма для каждой фиксированной пары параметров G и N. Для каждого прогона алгоритма задания генерировались заново.

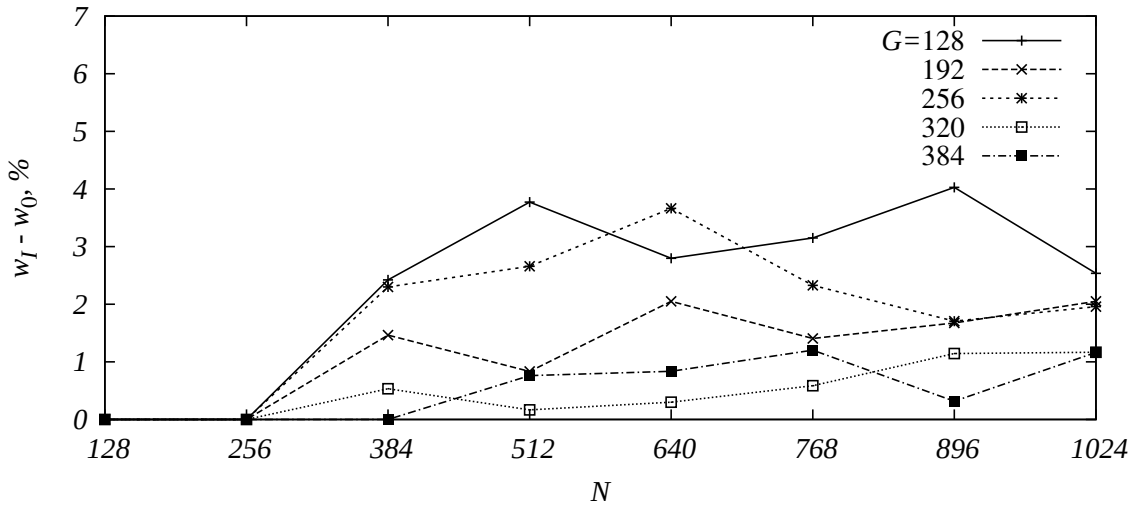

Рис. 2.15. Процент оптимизированной загруженности ресурсов после алгоритма обратного заполнения для числа всех ВУ (G) в РВС от 128 до 384

На рис. 2.15 показан выигрыш (средний по 30-ти прогонам) в загруженности модельных вычислительных ресурсов в процентном соотношении для тестируемого алгоритма при изначальной оптимизации расписания ал-

горитмом обратного заполнения. В данном случае рассматривались модели РВС, в сумме содержащие до 384 ВУ. При числе вычислительных процессов меньшим, либо равным числу ВУ выигрыша не наблюдается. Другими словами, при $D \leq 1$ разработанный алгоритм не способен оптимизировать очередь запуска параллельных заданий, где $D = N/G$ — размерность задачи планирования. С ростом D генетический алгоритм улучшает качество расписания, по сравнению с алгоритмом обратного заполнения, поддерживая выигрыш в использовании ресурсов от 0.5% до 4%.

Необходимо отметить, что на графике показаны средние значения загрузки по 30 прогонам, в то время как в некоторых случаях даже при $D > 1$ генетический алгоритм находил то же решение, что было получено алгоритмом обратного заполнения.

На рис. 2.16 показаны аналогичные зависимости, что и на рис. 2.15, но при числе ВУ в модельных РВС от 512 до 1024.

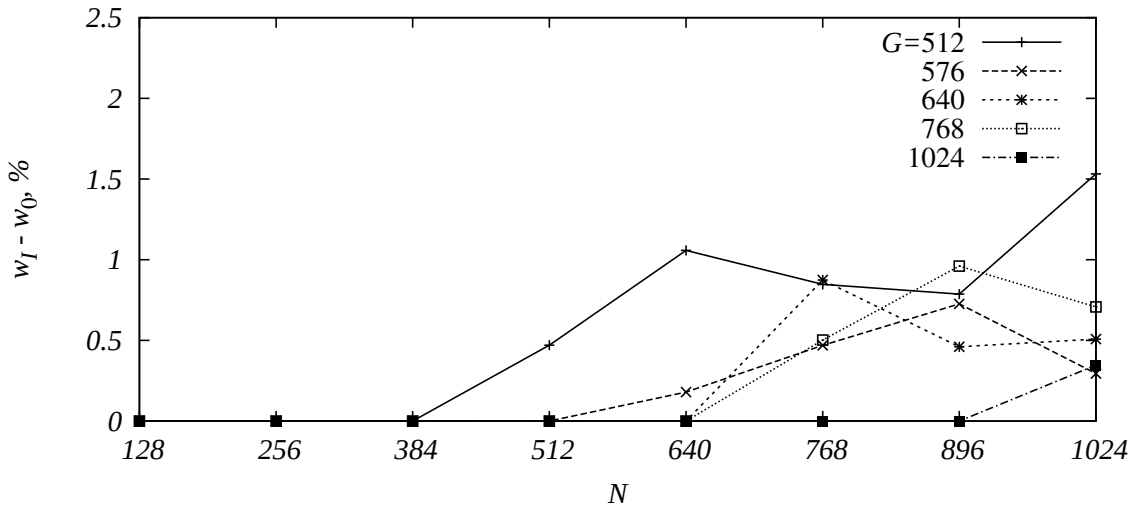

Рис. 2.16. Процент оптимизированной загруженности ресурсов после алгоритма обратного заполнения для числа всех ВУ (G) в РВС от 512 до 1024

Из рис. 2.16 видно, что при большем числе ВУ зависимость $(w_I - w_0)$ от D также сохраняется, как и при меньшем числе ВУ. При этом с каждым шагом увеличения G рост выигрыша в загрузке ресурсов становится все более пологим.

Практический выигрыш можно увидеть на зависимостях сэкономленного время простаивания $(\Delta_0 - \Delta_I)$ от G и N.

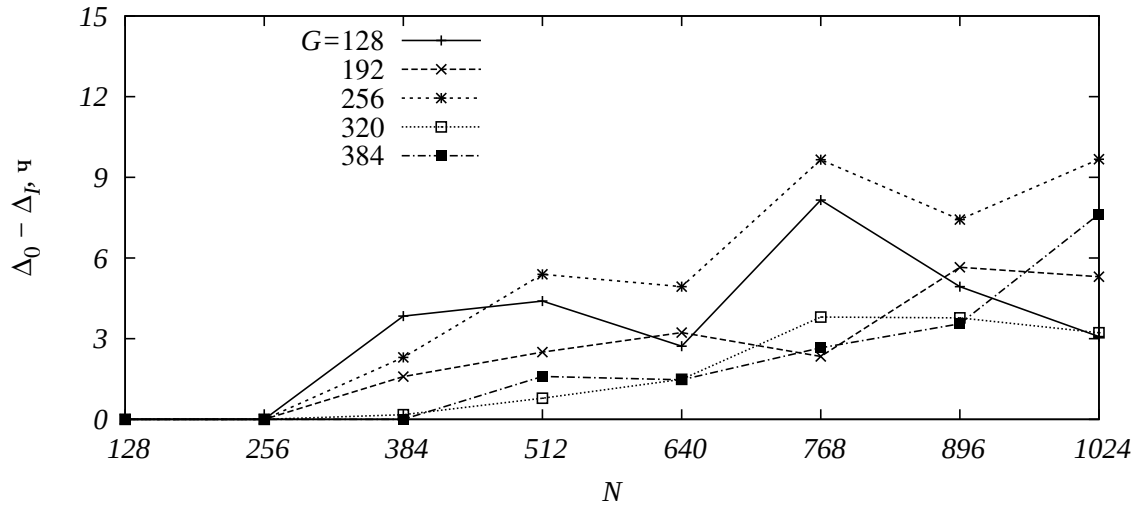

Рис. 2.17. Сэкономленное время простаивания $(\Delta_0 - \Delta_I)$, приходящееся на 1 ВУ для общего числа всех ВУ G в РВС от 128 до 384

Из рис. 2.17 видно, что $(\Delta_0 - \Delta_I)$ подчиняется схожим зависимостям, что и $(w_I - w_0)$. При числе процессов в очередях меньше, либо равное числу ВУ $(\Delta_0 - \Delta_I)$ остается нулевым. С ростом D генетический алгоритм уменьшает время простаивания, уплотняя расписание. Так, сэкономленное время простаивания в некоторых прогонах доходило до 12 часов на 1 ВУ. На рис. 2.18 показаны аналогичные зависимости, что и на рис. 2.17, но при числе ВУ от 512 до 1024. По нему видно, что при большем числе ВУ зависимость $(\Delta_0 - \Delta_I)$ от D также сохраняется, как и при меньшем их числе.

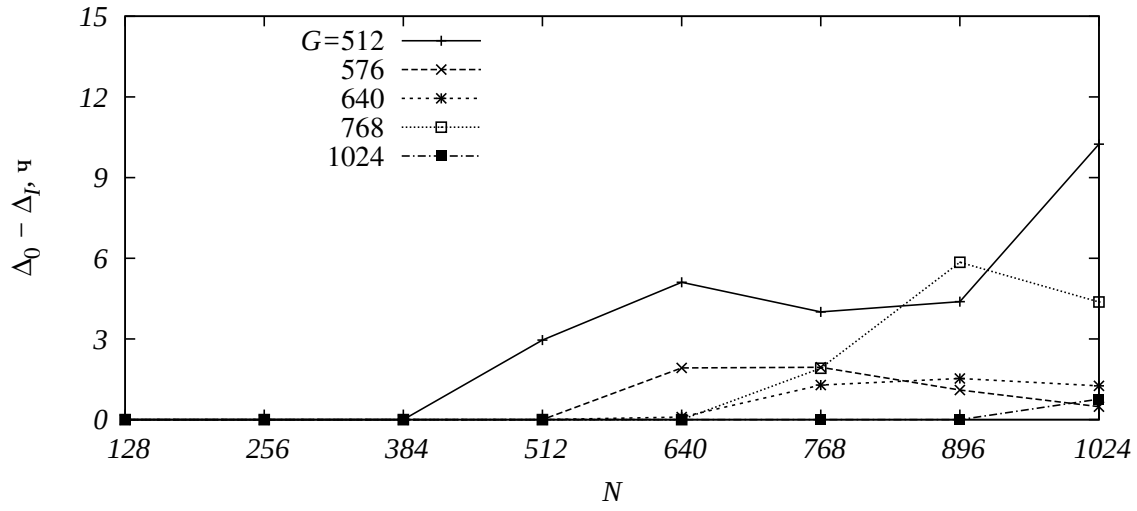

Рис. 2.18. Сэкономленное время простаивания $(\Delta_0 - \Delta_I)$ для числа всех ВУ (G) РВС от 512 до 1024

Глава 3. Практическая реализация алгоритма планирования

В данной главе рассматриваются существующие реализации систем планирования запуска заданий в Grid. Описывается созданная автором система планирования Geneur, в основу которой положены алгоритмы планирования выполнения непараллельных и параллельных заданий. Обсуждается применение разработанного программного комплекса планирования заданий для вычислительных ресурсов ВЦ ДВО РАН.

3.1. Реализации систем планирования запуска заданий в Grid

Создание системы планирования для Grid является достаточно сложной задачей. Ядром такой системы является менеджер ресурсов — подсистема, динамически составляющая расписание выполнения заданий.

Различают локальные и глобальные менеджеры ресурсов в Grid. К локальным относят пакетные менеджеры ресурсов вычислительного кластера. Наиболее востребованные в настоящее время локальные менеджеры ресурсов — это семейство PBS (Torque[1], PBS Pro[2]), Platform LSF[3], Load Leveler[4], NQE[5] (Net Queueing Environment), DQS[6] (Distributed Queueing System), SGE[7] (Sun Grid Engine) и Condor[8].

Глобальный менеджер ресурсов является единой точкой постановки заданий пользователей в очереди Grid. Он распределяет задания по вычислительным кластерам и другим вычислительным системам, являющимися сайтами Grid. Данные сайты часто управляются локальными ресурсами

[1] http://www.clusterresources.com/products/torque-resource-manager.php
[2] http://www.pbsgridworks.com
[3] http://www.platform.com
[4] http://www.ibm.com/systems/clusters/software/loadleveler
[5] http://www.cray.com
[6] http://scri.fsu.edu
[7] http://www.sun.com/software/sge
[8] http://www.cs.wisc.edu/condor

с помощью своих локальных менеджеров ресурсов. Операционные системы на узлах кластеров или отдельных вычислительных станциях в Grid управляют ресурсами конкретной вычислительной машины. Таким образом, планирование в Grid является многоуровневым.

Тем не менее, возможно объединение глобального и локального менеджеров ресурсов в единую систему для лучшего распределения ресурсов между заданиями. Так, глобальный менеджер ресурсов проекта Condor-G имеет возможность интегрироваться с локальным менеджером ресурсов Condor, что позволяет перераспределять ресурсы, перезапуская задания на другом вычислительном кластере.

Перечислим основные функции локальных менеджеров ресурсов:

- максимизировать общую загруженность всей системы, учитывая гетерогенность ресурсов;
- поддерживать различные типы технологий организации параллельных приложений, такие как MPI;
- предоставлять функциональность по удаленному выполнению заданий (постановка задания в очередь, прерывание выполнения, миграции, точки останова).

В основе любого менеджера ресурсов лежит алгоритм планирования. Таких алгоритмов может быть несколько, они могут взаимодействовать друг с другом для получения лучшего расписания, либо переключаться с одного на другой по требованию администратора. Так, система планирования GridWay поддерживает интерфейс к созданию своих алгоритмов планирования и позволяет переключаться с одного на другой в файле конфигурации. Описание алгоритмов планирования для Grid дается в разделе 1.2 диссертации.

Перечислим часто используемые системы планирования для Grid: Silver Grid Scheduler[9], GridWay[10], Condor-G[11] [158], Service Level Agreements[12], KB Metascheduler [114], Sun Grid Engine (SGE) scheduler[13].

[9] http://www.clusterresources.com/products/mgs
[10] http://www.gridway.org
[11] http://www.cs.wisc.edu/condor/condorg
[12] http://www.gridscheduling.org
[13] http://www.sun.com/software/sge

Отметим, что система планирования Silver Grid Scheduler основана на широко используемом в вычислительных кластерах планировщике Maui[14].

В целом распределенные вычисления можно организовывать и на базе некоторых локальных менеджеров ресурсов. Так PBS Torque позволяет создавать маршрутизирующие очереди, перенаправляющие запуск задания на другой вычислительный кластер. Тем не менее, данный вариант организации вычислений лишен гибкости, которой обладают Grid-инструментарии.

Для своего функционирования система планирования должна получать информацию о текущем состоянии Grid. Обычно такая информация запрашивается у сервиса Grid-инструментария. Так, например GridWay получает данную информацию от службы MDS (Monitoring and Discovery Service) инструментария Globus Toolkit.

Полный цикл обработки задания в Grid включает в себя следующие основные стадии.

- *Получение информации о ресурсах.* На данном этапе система планирования запрашивает информацию о доступных для данного пользователя в Grid ресурсах, основываясь на информации о ресурсном запросе пользователя.

- *Выбор ресурсов.* На основе информации о доступных ресурсах, полученной на предыдущем этапе, система планирования принимает решение о немедленном запуске задания, либо сохранении его в очереди.

- *Запуск.* Основываясь на принятом на предыдущем шаге решении, задание может быть доставлено и запущено на удаленном ресурсе, либо ресурс может быть зарезервирован до определенного момента в будущем для данного задания. При этом, помимо доставки приложения на удаленный ресурс, задание может "требовать" некоторой подготовки для своего запуска, как, например, доставку файлов данных, либо предварительную компиляцию под необходимую архитектуру вычислительной системы.

[14] http://supercluster.org/maui

- *Мониторинг.* На данной стадии пользователь может контролировать состояние задания. Система планирования также должна следить за состоянием для выполнения необходимых действий по окончанию или прерыванию выполнения задания.

- *Завершение.* Когда выполнение задания завершается, в том числе и по причине возникновения нарушения в его работе, система планирования выполняет ряд заданных администратором действий, таких, как отправка сообщения по электронной почте владельцу задания и возвращение пользователю файлов результата выполнения задания.

- *Очистка.* Данная стадия предполагает удаление данных, оставшихся на удаленных узлах и более не нужных для выполнения задания. Чаще всего для такого удаления пользователь должен явно указать информацию о том, какие файлы требуют удаления.

3.2. Система планирования Geneur

Система планирования Geneur основана на разработанных в диссертации алгоритмах планирования. РВС типа Grid, для которой создавалась данная система планирования обладает следующими свойствами:

- ресурсы полностью отчуждаемы;

- управление ресурсами централизовано;

- сайтами являются однородные многопроцессорные комплексы кластерного типа или вычислительные системы с общей памятью;

- прикладное программное обеспечение предустановлено на сайты;

- сайты считаются абсолютно надежными.

Генетические алгоритмы составления расписания выполнения заданий были распараллелены с применением островной модели [18, 19], которая подразумевает одновременную и независимую работу нескольких копий генетического алгоритма для различных подпопуляций. Через определенное число итераций ГА (число поколений изоляции) между данными

подпопуляциями осуществляется обмен (миграция) лучшими найденными на тот момент времени решениями (расписаниями). Такая модель может увеличивать эффективность поиска расписания, препятствуя преждевременной сходимости ГА и помогая более интенсивно исследовать пространство решений.

Система планирования имеет клиент-серверную архитектуру. Вычислительный модуль — запущенная копия на отдельной ЭВМ программной реализации ГА (“остров”), управляющая эволюцией одной подпопуляции. Сервер планирования (СП) — серверный компонент системы, принимающий от вычислительных модулей и распространяющий между всеми подпопуляциями лучшие решения в процессе миграции, а также осуществляющий иные действия для поддержания процесса планирования. СП, вычислительные модули, утилиты управления заданиями пользователей и набор скриптов для тестирования составляют пакет программ системы Geneur.

Вычислительные модули запускаются одновременно и ожидают команды от СП. При необходимости составить расписание, СП рассылает вычислительным модулям описание ресурсов в Grid и паспорта заданий. По завершению периода планирования СП останавливает поиск расписания и выбирает лучшее из найденных согласно целевой функции. Из найденных расписаний формируется единый список выполнения заданий, отсортированный по абсолютному времени запуска заданий. При наступлении запланированного для задания момента времени оно запускается в Grid.

На рис. 3.1 представлена диаграмма потоков данных описываемой системы. Цифрами в кругах внутри стрелок обозначен порядок “следования” данных.

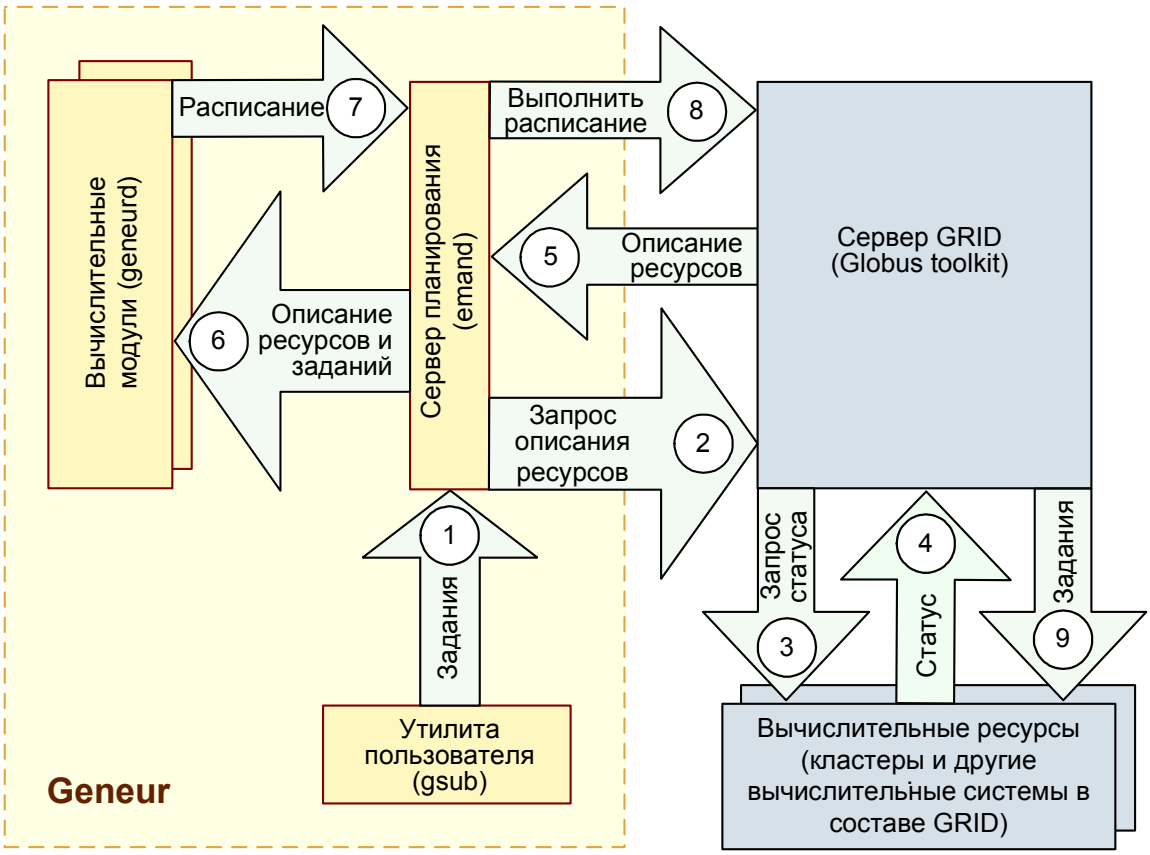

Рис. 3.1. Схема потоков данных при применении системы планирования Geneur в Grid на базе инструментария Globus Toolkit (цифрами обозначен порядок следования запросов)

После поступления файлов описания заданий от пользователей в СП (стрелка 1) осуществляется выяснение текущего состояния вычислительных ресурсов у сервера Grid (стрелка 2). Также СП поддерживает получение описания ресурсов из локальных файлов. В результате запроса описания ресурсов, сервер Grid производит опрос текущего статуса доступных вычислительных ресурсов (стрелка 3, 4). Данный этап реализуется средствами инструментария Globus Toolkit[15] и установленными на вычислительных узлах компонентами системы мониторинга Ganglia[16]. Затем полное описание доступных ресурсов возвращается серверу планирования (стрелка 5). Собрав к данному моменту всю необходимую информацию для составления расписания, СП рассылает ее вычислительным модулям (стрелка 6). Последние, после нахождения расписания заданий, возвращают его обратно СП (стрелка 7). На основании полученного расписания СП

[15] http://www.globus.org/toolkit
[16] http://ganglia.info

осуществляет рассылку заданий на соответствующие вычислительные узлы (стрелка 8, 9).

Необходимо отметить, что в описываемой системе предусмотрено принудительное обновление состояния ресурсов администратором РВС, путем посыла управляющей программе-демону (в терминах UNIX-подобных операционных систем) системного posix-сигнала[17] SIGHUP. Это же действие, будучи выполненное в отношении той или иной программы-демона вычислительного модуля, заставляет последнюю обновить параметры ГА из конфигурационного файла. Это позволяет проводить автоматическое тестирование алгоритмов с различными параметрами без перезапуска СП и вычислительных модулей, используя локальные файлы описания ресурсов РВС. Обмен данными между сервером планирования и вычислительными модулями происходит в отдельных потоках выполнения без прерывания работы ГА посредством сетевого протокола TCP.

Globus Toolkit предоставляет программный интерфейс к своим компонентам. Это позволило интегрировать систему планирования Geneur в инфраструктуру Grid посредством данного программного интерфейса. На рис. 3.2 показана схема взаимодействия Geneur при выполнении расписаний с такими Grid-службами инструментария как WS-MDS, GridFTP, RFT и WS-GRAM. Программный слой взаимодействия с Grid содержит основные функции для выполнения расписания и получения информации о ресурсах, что позволяет изменять связь с Grid-инструментарием, внося изменения лишь в данный промежуточный программный код.

[17] http://standards.ieee.org/regauth/posix

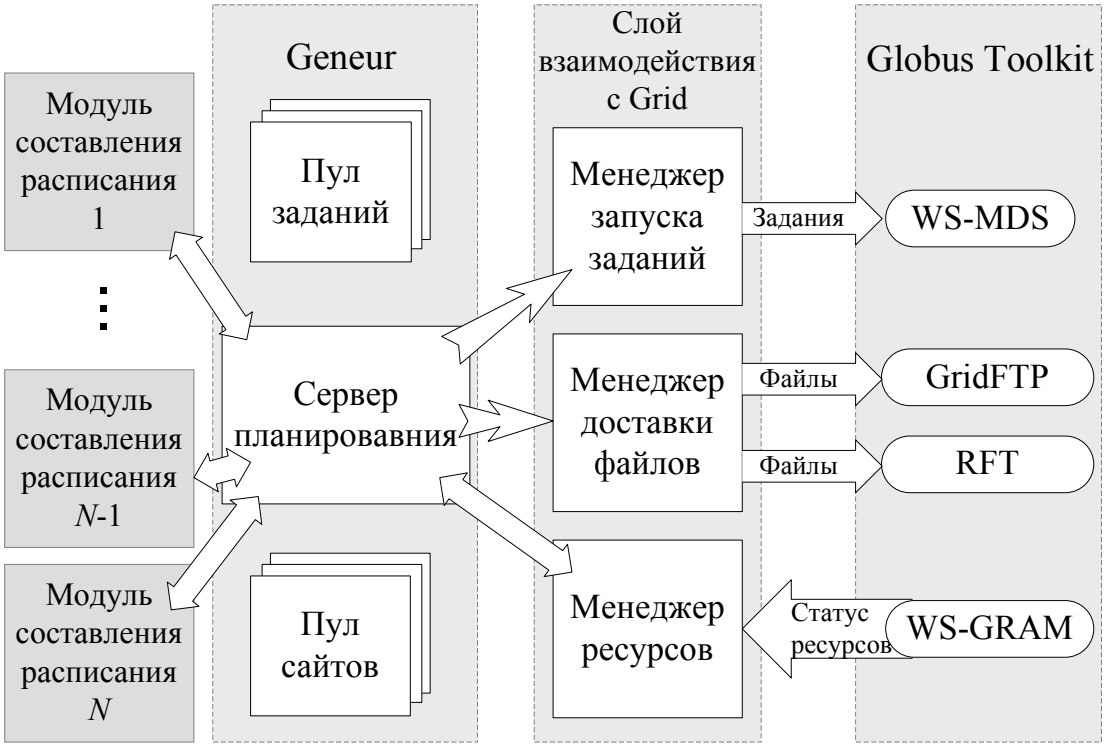

Рис. 3.2. Схема взаимодействия системы планирования Geneur и Grid через промежуточный программный слой

Установленное в Grid прикладное программное обеспечение, запускаемое заданиями, описывается в отдельных файлах в директории `./etc/software/`. В данных файлах определяются права доступа пользователей к ПО, пути к исполняемм файлам в файловой системе сайтов и другая служебная информация. Пример файла описания задания показан в приложении Б на листинге 1.

Для поиска расписания алгоритм нуждается в характеристиках планируемых заданий, которые указываются индивидуально для каждого задания в формате *JSDL-WG*. Данный язык является адаптированной версией языка *JSDL*[18], основан на стандарте *XML*[19] и служит для описания требований заданий к ресурсам распределенных вычислительных систем. Пример описания задания на языке *JSDL-WG* представлен в приложении Б на листинге 2. Обязательными значениями параметров задания являются следующие.

[18]https://forge.gridforum.org/sf/projects/jsdl-wg
[19]http://www.w3.org/XML

1. *ProcessCountLimit* — число вычислительных процессов задания.
2. *OperatingSystemName* — имя операционной системы, на которой может выполняться задание.
3. *OperatingSystemVersion* — версия операционной системы, на которой может выполняться задание.
4. *IndividualCPUTime* — время выполнения задания.
5. *CPUArchitectureName* — архитектура процессора, на котором замерялось значение параметра *IndividualCPUTime*.
6. *IndividualCPUSpeed* — частота процессора, на котором замерялось значение параметра *IndividualCPUTime*.
7. *IndividualCPUCount* — необходимое заданию число виртуальных узлов на каждом физическом вычислительном узле.
8. *TotalResourceCount* — необходимое заданию число вычислительных узлов, содержащих *IndividualCPUCount* виртуальных узлов.
9. *IndividualPhysicalMemory* — необходимое заданию количество оперативной памяти.

Время расчета задания *IndividualCPUTime* является важнейшей характеристикой. Существует ряд работ [8, 34, 148, 149], посвященных прогнозированию времени выполнения программ. Данные прогнозы строятся на основе статического или динамического анализа программы. В общем виде статический анализ состоит в предварительной фрагментации программы, статическом анализе данных фрагментов и последующей генерации вывода о длительности работы программы на той или иной архитектуре вычислительной системы.

В случае динамического анализа каждый фрагмент кода исследуется в реальных прогонах без компилирования программы из языка высокого уровня в машинный код. Для этого может быть использован *метод частотных счетчиков* [12], основная идея которого состоит во введении в код средств для подсчета числа выполнений фрагментов программы.

В простейшем случае, для вычисления параметра *IndividualCPUTime* достаточно провести один запуск программы на доступном процессоре с архитектурой *CPUArchitectureName* и частотой *IndividualCPUSpeed*. Параметры *IndividualCPUTime*, *CPUArchitectureName* и *IndividualCPUSpeed* необходимы для вычисления длительности выполнения процессов задания. При составлении расписания для гетерогенных систем происходит масштабирование длительности выполнения вычислительных процессов в соответствии с фактически используемым типом процессора по формуле (2.1) (см. раздел 2.2).

Такой вариант масштабирования времени выполнения заданий имеет смысл, если все процессоры в Grid принадлежат одному семейству (например, все – Xeon серий 5xxx). При существенно разнородных процессорах в Grid формула (2.1) становится недостаточно точной, что приводит к появлению дополнительных отрезков времени простаивания вычислительных ресурсов или к снятию заданий со счета при привышении рассчитанного времени. В таком случае имеет смысл использовать то свойство Grid, что прикладное ПО, запускаемое заданиями, заранее установлено на сайтах. В этом случае после нескольких экспериментальных запусков того или иного программного обеспечения администратор сайта может определить время счета при согласованных в пределах Grid некоторых стандартных параметрах для этого ПО. По результатам запусков определяется коэффициент ускорения относительно одного из сайтов. В системе планирования Geneur указанный коэффициент заносится в файл описания прикладного ПО (см. в приложении Б на листинге 1 параметр `kperf`).

Необходимо отметить, что данный вариант определения коэффициента ускорения также не всегда может быть достаточно точным. Пока не существует абсолютно точного общего алгоритма для определения времени работы программы на ЭВМ. Узнать степень точности вычисления коэффициентов ускорения для конкретного прикладного ПО в Grid с оперделенной конфигурацией можно лишь эмперически. Если для прикладного ПО не заданы коэффициенты ускорения, то в описываемой системе планирования используется формула (2.1).

На рис. 3.3 схематически изображена архитектура сервера планирования emand.

Рис. 3.3. Архитектура сервера планирования emand

Алгоритм управления планированием отмечен на рис. 3.3 как *основной цикл планирования*. В данном бесконечном цикле `emand` осуществляет в несколько потоков все действия, необходимые для управления программными демонами `geneurd`.

Подсистема доступа к описанию ресурсов предоставляет серверной подсистеме описание вычислительных ресурсов в Grid, получая их из локальных файлов, либо запрашивая данную информацию у web-сервиса *Globus Toolkit MDS* (Monitoring and Discovery System).

Рис. 3.4. Архитектура вычислительного модуля geneurd

На рис. 3.4 схематически изображена архитектура вычислительного модуля `geneurd`. Подсистема доступа к описанию ресурсов посредством сетевой компоненты клиента запрашивает при каждом планировании информацию о ресурсах у сервера планирования `emand`.

Подсистема сбора статистических показателей сохраняет в памяти на каждом поколении ГА такие параметры, как среднее и лучшее значения функции пригодности, время составления расписания и другие. После завершения составления расписания собранные статистические данные сохраняются в файлы в директории, указанной в файле конфигурации `geneurd.conf`. Серверная и клиентская подсистемы программных демонов `emand`, `geneurd`, а также утилиты управления заданиями `gsub` реализованы на базе сетевой библиотеки `Boost::Asio`[20].

[20] http://boost.org

3.3. Компиляция, установка и настройка Geneur

После получения исходных текстов[21] созданного в процессе выполнения диссертации программного комплекса Geneur, необходимо убедиться в следующих установленных в операционной системе библиотеках.

- `Boost C++ Libraries`[22] — набор библиотек, расширяющих язык программирования `C++`. Свободно распространяются по лицензии Boost Software License с исходным кодом.
- `ThreadPool`[23] — библиотека (надстройка над *Boost C++ Libraries*), предоставляющая механизмы управления пулом потоков.
- `LibXml2`[24] — библиотека, предоставляющая функции работы с XML-данными.

После компиляции и установки данных библиотек можно приступать к сборке пакета программ `Geneur` с помощью следующей последовательности команд, выполняемой в корневом каталоге программы:

```
./configure ––prefix=/opt/geneur ––with–geneurd
––with–emand ––with–utils

make

make install
```

В корневом каталоге также присутствуют следующие директории.

- *./etc* — директория файлов конфигурации;
- *./scripts* — набор shell- и python-скриптов для проведения тестов;
- *./src* — файлы исходных кодов на языке `C++`;
- *./test* — тестовые данные (наборы заданий, описания вычислительных ресурсов и др.);

[21] http://sourceforge.net/projects/geneur
[22] http://boost.org
[23] http://threadpool.sourceforge.net
[24] http://xmlsoft.org

- *./var* — директория для хранения временных данных;

В каталоге программы *./etc* расположены следующие файлы конфигурации:

- *geneur.conf* — файл с параметрами ГА и вычислительного модуля `geneurd`;
- *eman.conf* — файл с параметрами сервера планирования `emand`;
- *gsub.conf* — файл с параметрами утилиты `gsub` постановки заданий в очереди на планирование;
- *nodes.list* — список DNS-имен (или IP-адресов) вычислительных узлов, на которых будут запущены вычислительные модули;
- *queues.xml* — файл описания очередей для заданий;
- *conveyors.list* — таблица соответствий архитектуры процессоров, применяемых в Grid и числа конвейеров.

Основные параметры файла *geneur.conf* с комментариями показаны в приложении Б на листинге 3. Данный файл может быть скопирован на каждый узел, где будет работать вычислительный модуль, либо разделяться (например с помощью сетевой файловой системы NFS) несколькими вычислительными модулями. В случае их различных настроек появляется возможность ставить численные эксперименты с различными параметрами генетических алгоритмов, управляющими подпопуляциями.

Пример файла параметров *eman.conf* с комментариями показан в приложении Б на листинге 4. Основные параметры файла *gsub.conf* аналогичны параметрам *eman.conf* и *geneur.conf*.

Файл *queues.xml* содержит описание приоритета заданий в очередях и другие настройки очередей в формате стандарта XML. Пример файла *queues.xml* для двух очередей заданий (`verylong` и `short`):

```
<queues>
  <queue>
    <name> verylong </name>
    <priority> 1 </priority>
    <path> queues/verylong </path>
```

```
  </queue>

  <queue>
    <name> short </name>
    <priority> 2 </priority>
    <path> queues/short </path>
  </queue>
</queues>
```

Здесь параметр *path* указывает относительный путь к директории временного хранения файлов описания заданий, а *priority* — приоритет заданий в очереди.

Таблица соответствий типов архитектур процессоров, применяемых в Grid и числа конвейеров процессоров данных типов архитектур в файле *conveyors.list* содержит список следующего формата:

```
<имя архитектуры>    <число конвейеров>    # <комментарий>
```

Например, для процессоров xeon с четырьмя конвейерами строка в файле *conveyors.list* будет следующая:

```
xeon    4    # Intel Xeon
```

Названия типов архитектур не регламентируются и могут быть изменены администратором. Пользователи должны следовать данным названиям при составлении файла описания задания.

Формат файла расписания представляет собой XML-документ с описанием места (идентификатора ВУ) и времени запуска того или иного вычислительного процесса задания. Пример расписания для трех процессов, планируемых на 1 вычислительном кластере из двух ВУ:

```
<schedule>

  <resource>
    <hostname> cluster_sample </hostname>

    <cpu>
      <id> 0 </id>
      <job>
        <order> 1 </order>
        <jobid> 3 </jobid>
        <cputime> 2000 </cputime>
        <execline> /home/taras/NPB/sp.A.4 </execline>
      </job>
```

```
      <job>
        <order> 2 </order>
        <jobid> 2 </jobid>
        <cputime> 200 </cputime>
        <execline> /home/taras/NPB/ep.C.4 </execline>
      </job>
    </cpu>

    <cpu>
      <id> 1 </id>
      <job>
        <order> 1 </order>
        <jobid> 2 </jobid>
        <cputime> 1000 </cputime>
        <execline> /home/taras/NPB/ft.B.4 </execline>
      </job>
    </cpu>

  </resource>

</schedule>
```

Формат данного файла может быть адаптирован администратором для различных Grid-инструментариев ввиду открытости распространяемых исходных кодов системы планирования. Для этого необходимо изменить метод `xml()` класса `geneur::Schedule` в файле `./src/core/schedule.cpp`.

3.4. Планирование запуска заданий на вычислительных ресурсах ВЦ ДВО РАН

В разделе рассматривается исследование системы планирования Geneur на вычислительных ресурсах ВЦ ДВО РАН.

Первый вычислительный кластер ВЦ ДВО РАН был построен в 2004 году [26]. Он состоит из управляющего узла с RAID-массивом и 8 бездисковых узлов с процессорами Intel Pentium4 3 ГГц, объединенных коммуникационной сетью Gigabit Ethernet. Производительность по тесту HPL — 35.5 Gflops. В работе [27] описывается архитектура, конструкция, ПО и результаты экспериментального исследования производительности вычислительного кластера. Данный кластер построен с использованием распространенных комплектующих и относится к начальному уровню высоко-

производительных вычислительных систем. Он не позволяет существенно повысить производительность вычислений, однако является удобной экспериментальной базой для разработок в области системного и прикладного ПО. Данный кластер в настоящий момент используется в качестве экспериментальной площадки по исследованию новых программных технологий.

Второй вычислительный кластер, введенный в эксплуатацию в 2007 году, состоит из восьми узлов, каждый из которых содержит по два двухядерных процессора Intel Xeon 5060 3,2 ГГц и 4 Гб оперативной памяти типа ECC. Пиковая производительность данного кластера — 204 Gflops. Реальная производительность по тесту HPL — 144 Gflops. Результаты исследований производительности данного кластера приведены в работе [51].

Третий вычислительный кластер, введенный в эксплуатацию в 2009 году, состоит из пяти узлов, каждый из которых содержит по два четырехядерных процессора Intel Xeon 5450 3.0ГГц и 16 Гб оперативной памяти типа ECC. Пиковая производительность данного кластера составляет 480 Gflops. Кластер построен на платформе Sun Blade X6250 с возможностью дальнейшего расширения за счет резервных *blade*-узлов.

Исследование системы планирования Geneur. Количество заданий в очереди может быть различным, поэтому алгоритм должен надежно функционировать при любом их числе. Далее исследуются аспекты масштабируемости разработанных алгоритмов планирования в составе системы планирования Geneur на вычислительных ресурсах ВЦ ДВО РАН.

Обозначим разработанный алгоритм планирования выполнения *непараллельных* заданий в РВС как **A1**, а алгоритм планирования *параллельных* заданий как **A2**.

Численное решение нахождения расписания с использованием ГА считалось найденным, если оно отличалось от наилучшего решения не более чем на 5%. Для проведения тестирования была написана на языке программирования `C++` утилита *sgen* для генерирования заданий, лучшее расписание запуска которых является идеальным. Под идеальным здесь понимается расписание, последние процессы на каждом из ВУ которого заканчивают выполнение в один момент времени и в котором отсутствуют периоды простаивания ВУ.

Для таких заданий это расписание будет иметь минимальное (лучшее) значение целевой функции. Для описания алгоритма составления такого расписания введем понятие ВУ-блока. Это множество последовательно пронумерованных виртуальных узлов, находящихся под вычислительной нагрузкой равный отрезок времени (рис. 3.5).

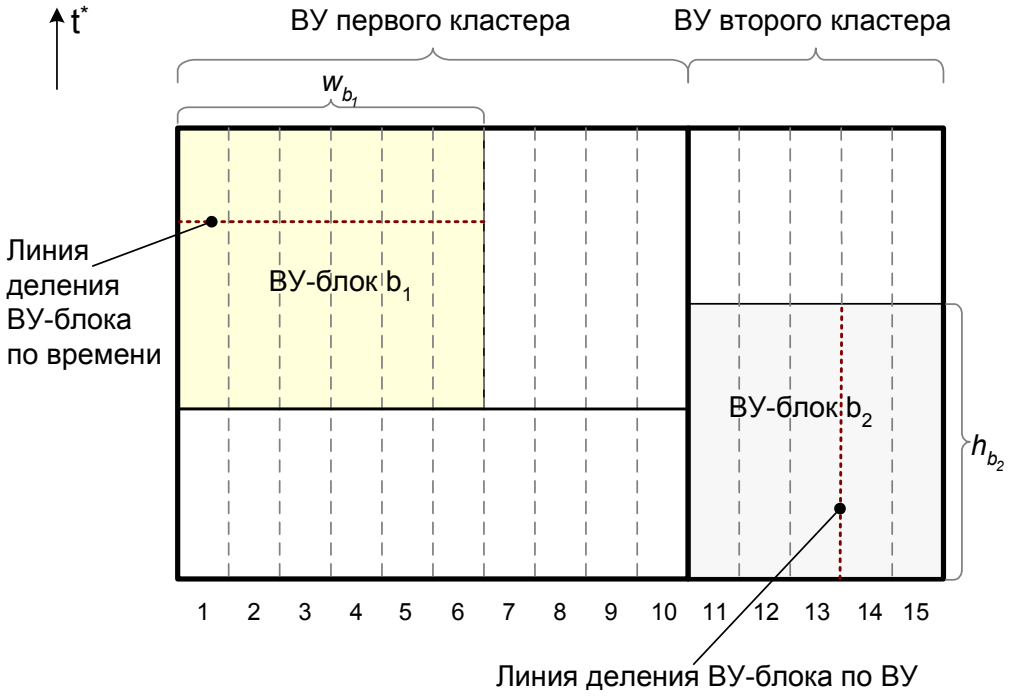

Рис. 3.5. Схема деления блоков идеального расписания s^* для получения множества планируемых заданий; t^* — шкала времени расписания s^*

Каждый ВУ-блок описывает одно задание. Ширина ВУ-блока (на рис. 3.5 обозначена как w_{b_1}) равна числу вычислительных процессов задания (по одному процессу на ВУ). Длина ВУ-блока (на рис. 3.5 обозначена как h_{b_2}) равна длительности процессов соответствующего задания по шкале времени расписания. Алгоритм составления идеального расписания s^*, реализованный в утилите *sgen*, следующий.

1. Каждому вычислительному кластеру ставится в соответствие один ВУ-блок с шириной w_b, равной числу виртуальных узлов этого кластера, высотой h_b, равной длине l^* расписания s^*.

2. Полученные ВУ-блоки добавляются во множество B с количеством элементов N_b.

3. Если величена N_b/w_b достигла достаточного числа процессов всех заданий в s^*, то работа алгоритма завершается.

4. Случайным образом выбирается блок $b_i \in B$.

5. Случайным образом выбирается метод деления ВУ-блока на две части: по времени или по ВУ (см. рис. 3.5).

6. В случае деления ВУ-блока по времени случайным образом выбирается момент времени в диапазоне значений $(\alpha_b, \alpha_b + h_b)$, где α_b — время начала ВУ-блока b_i в расписании s^*.

7. В случае разделения по ВУ случайным образом выбирается позиция между двумя соседними ВУ в ВУ-блоке b_i.

8. b_i делится на b_i^1 и b_i^2 выбранным методом.

9. b_i исключается из множества B.

10. Осуществляется переход к шагу 3. ■

При такой форме тестирования необходимо отключать механизм сохранения непрерывности двух последовательно найденных расписаний. Данный механизм может увеличивать длину расписания ввиду того, что задания могут быть спланированы на запуск не с начала времени выполнения расписания. Это объясняется тем, что условием останова при проведении численных экспериментов являлось нахождение решения, целевая функция которого на 95% совпадает с целевой функцией идеального расписания. Таким образом, с включенным механизмом сохранения непрерывности с каждым новым поиском длина расписаний будет увеличиваться (если только предыдущее расписание не на 100% совпадет с идеальным) и целевые функции будут все больше отличаться от целевых функций идеального расписания.

В качестве описания ресурсов РВС для тестирования были взяты два кластера Вычислительного центра ДВО РАН. Их краткие характеристики приведены в таблице 3.1.

Таблица 3.1. Краткие характеристики кластеров тестовой конфигурации Grid

№	Количество		Тип процессора	Число конвейеров на один ВУ	Оперативная память на один узел/один ВУ, Гб
	узлов кластера	ВУ			
1	8	32	Xeon 5060 3.2 ГГц	4	4/1
2	5	40	Xeon 5450 3.0 ГГц	4	16/2

Важным свойством разработанных алгоритмов составления расписаний является их полиномиальная сложность. Другими словами, время решения ограничено сверху некоторым полиномом от размерности D задачи планирования. Так как данный полином, как установлено далее, является полиномом первой степени, то можно говорить о линейной сложности алгоритма. Далее экспериментально показывается данное свойство разработанных алгоритмов для множества тестовых заданий идеального расписания.

Размерность задачи составления расписания вычисляется по формуле $D = N/G$, где N — число вычислительных процессов всех заданий в расписании, а G — число доступных виртуальных узлов в PBC. С повышением N будет расти и время поиска расписания T_p. Для исследования этой зависимости были проведены испытания алгоритма для множества значений D.

Число процессов N всех заданий в очереди для тестов алгоритма A1 — 144 процесса, для A2 — 216. Для каждой размерности осуществлялось 30 прогонов алгоритма A1, затем измерялась длительность каждого поиска расписания и вычислялось ее среднее значение (на рис. 3.6 обозначено как T_p).

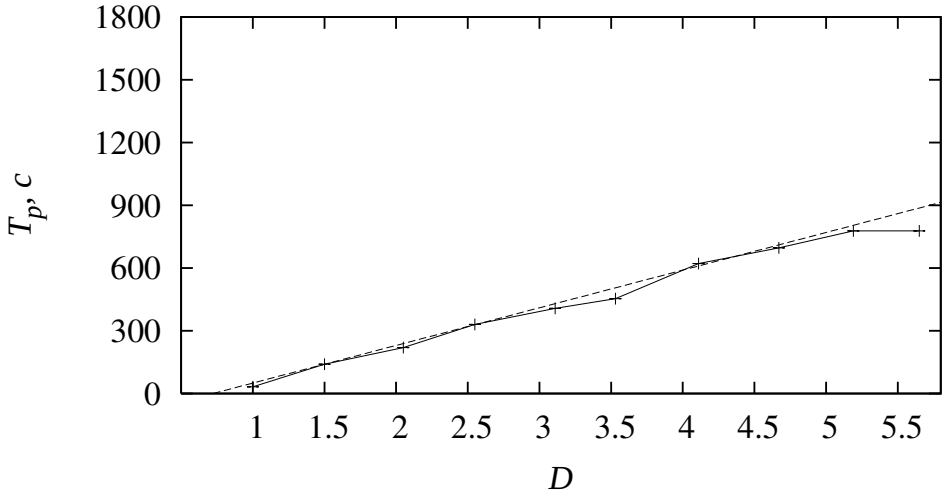

Рис. 3.6. Время T_p решения задачи составления расписания в зависимости от размерности D для алгоритма A1

Сплошной линией на рис. 3.6 обозначена зависимость времени решения T_p от размерности D, а пунктирной — аппроксимирующая линейная функция $T_p = 180D - 130$, по которой видно, что сложность алгоритма является линейной функцией от размерности задачи — $O(D)$. Вертикальными отрезками обозначены значения максимального и минимально времени поиска расписания по всем прогонам при данной размерности. Их значения также линейно возрастают с ростом D.

Аналогичная зависимость показана для алгоритма A2 на рис. 3.7. Расписания для начальной популяции генерировались случайным образом.

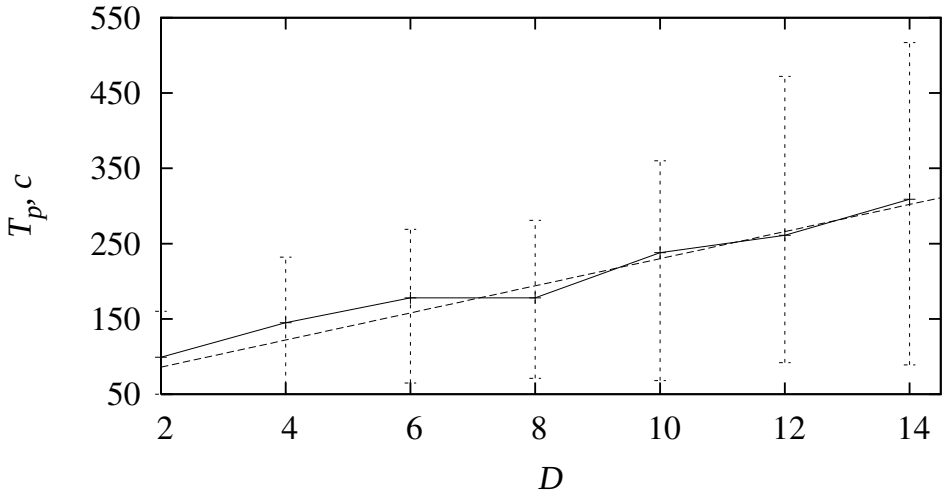

Рис. 3.7. Время T_p решения задачи составления расписания в зависимости от размерности D для алгоритма A2

Пунктирной линией на рис. 3.7 обозначена аппроксимирующая линейная функция $T_p = 18D{+}50$, по которой видно, что сложность алгоритма A2 также является линейной функцией от размерности задачи планирования — $O(D)$, но с меньшим множетелем. Вертикальными отрезками обозначены значения максимального и минимально времени поиска расписания по всем прогонам при данной размерности. Их значения также линейно возрастают с ростом D.

Далее приводятся результаты исследования эффективности реализации алгоритмов при фиксированном числе виртуальных узлов (таблица 3.1) и процессов, но с разными числами вычислительных модулей системы. Каждый вычислительный модуль (оперирующий своей подпопуляцией) запускался на отдельном узле вычислительного кластера (на кластере номер 1 из таблицы 3.1). Число параллельно работающих вычислительных модулей в испытании варьировалось от 1 до 8.

На рис. 3.8 представлена зависимость коэффициента ускорения k_p от числа P параллельных подпопуляций (каждая из которых содержала 18 особей и обрабатывалась своим вычислительным модулем).

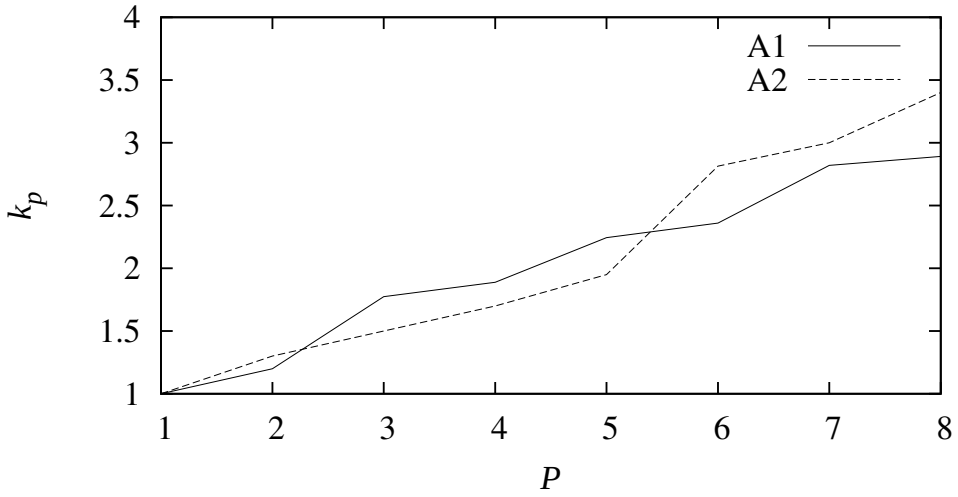

Рис. 3.8. Коэффициент ускорения k_p решения задачи в зависимости от числа P параллельных подпопуляций

Коэффициент ускорения в данном случае вычисляется по формуле $k_p = T_1/T_p$, где T_1 — время непараллельного поиска расписания, а T_p — время поиска для P вычислительных модулей в системе. Из данного ри-

сунка видно, что при росте числа подпопуляций P, не наблюдается заметного насыщения в ускорении. Экстраполируя полученные результаты на $P > 8$ можно сделать вывод, что алгоритмы будут давать некоторый прирост производительности и при дальнейшем увеличении числа узлов. В целом, рост производительности для 8 параллельно работающих вычислительных модулей (в сравнении с непараллельным поиском) был приблизительно троекратным.

Число I_m поколений изоляции — это число поколений ГА, проходящее до обмена найденными решениями между подпопуляциями. Осуществляется обмен решениями с наименьшими значениями целевой функции. Для исследования зависимости времени нахождения решения от значения I_m были использованы 8 подпопуляций (по 1-й на узел вычислительного кластера, на котором происходил поиск расписания). В процессе численного эксперимента были зафиксированы ресурсы (таблица 3.1) и число вычислительных процессов в расписании. Производилось по 30 запусков поиска расписания с равным значением I_m. Измерялось значение времени T_p нахождения расписания для $I_m = 25, 50, \ldots, 300$.

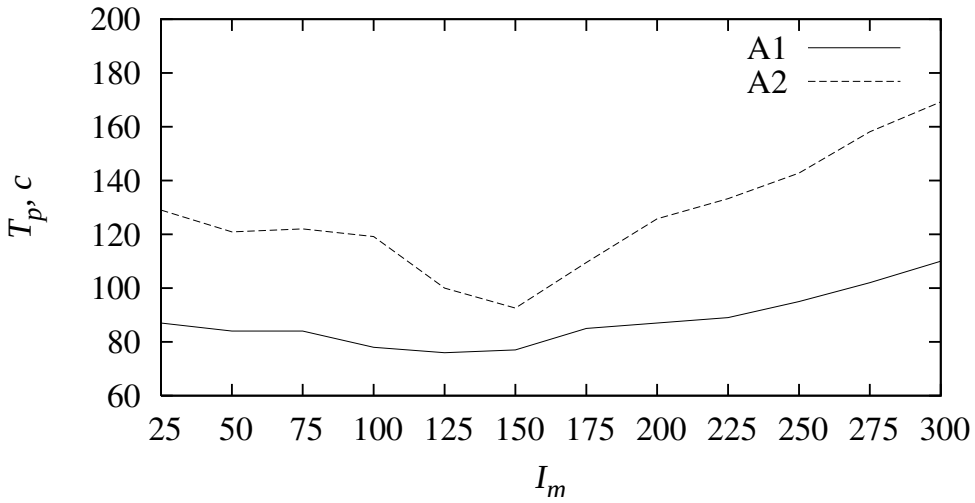

Рис. 3.9. Время T_p решения задачи составления расписания в зависимости от числа поколений изоляции I_m

По зависимостям на рис. 3.9 можно судить о том, что очень частые синхронизации (как и очень редкие) не способствуют нахождению решения алгоритмами. Оптимальное число I_m здесь находится в отрезке $[100, 150]$

для алгоритма A1 и в отрезке $[125, 150]$ для A2. Необходимо заметить, что на вычислительных кластерах с отличными от испытываемого характеристиками локальной сети передачи данных (скорость, латентность), возможно смещение диапазона оптимальных значений I_m.

Сравнение алгоритма поиска расписания параллельных заданий с алгоритмом обратного заполнения. На практике алгоритм обратного заполнения при планировании параллельных заданий с небольшим числом процессов является достаточно эффективным и быстрым. Генетические алгоритмы могут уступать ему в скорости. Важным при применении ГА является знание условий, при которых он может составлять более эффективные расписания по сравнению с общепринятым алгоритмом. В данном случае учет этих условий необходимо осуществлять при принятии решения: оптимизировать очередь заданий с помощью ГА или остановиться на более быстром алгоритме обратного заполнения.

Для оценки эффективности реализации разработанного алгоритма планирования параллельных заданий по сравнению с алгоритмом обратного заполнения проведен следующий численный эксперимент. Для фиксированной конфигурации Grid (таб. 3.1) рассматривались задания в очереди с различным числом процессов. Изменялось как максимальное число процессов каждого задания ($N_p = 10, 15, \ldots, 40$), так и общее число процессов всех заданий ($N = 128, 256, \ldots, 1024$).

Необходимо отметить, что значение N_p ограничивало лишь верхнюю границу числа процессов каждого задания, в то время как нижняя граница в экспериментах равнялась 1. Для каждого прогона задания генерировались заново случайным образом; число их процессов бралось от 1 до N_p. Учитывались средние значения времени поиска расписания по 30 прогонам для каждого сочетания факторов. Все задания имели одинаковый приоритет, равный 1.

На рис. 3.10 представлена зависимость выигрыша загруженности ресурсов $w_I - w_0$ от общего числа процессов N, где w_I — загруженность вычислительных ресурсов согласно оптимизированному с помощью разработанного автором алгоритма расписанию, а w_0 — аналогичная загруженность, согласно расписанию, полученному алгоритмом обратного заполне-

ния.

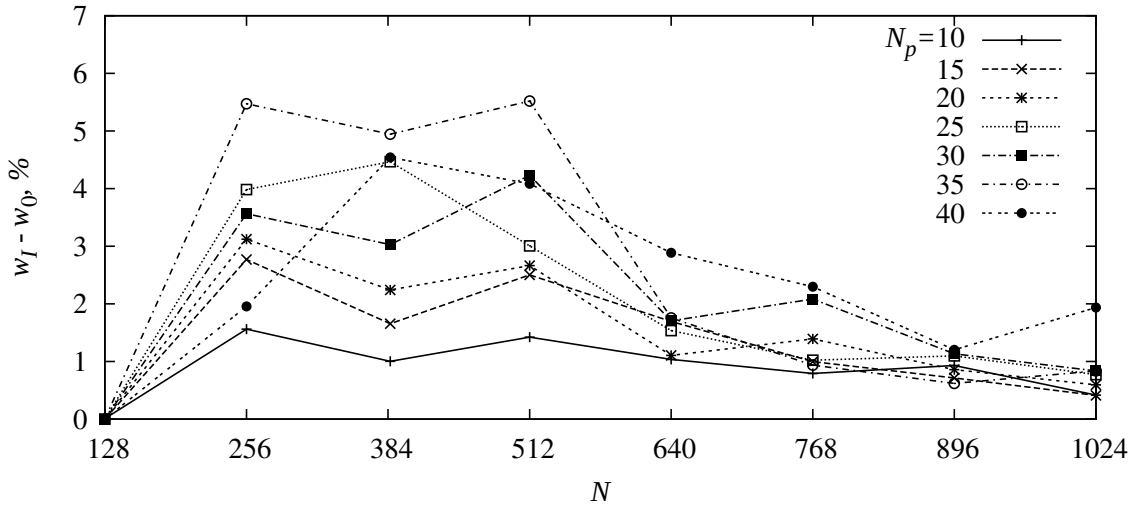

Рис. 3.10. Зависимость выигрыша загруженности $w_I - w_0$ вычислительных ресурсов от числа всех процессов N и максимального числа процессов одного задания N_p

По изображенным на рис. 3.10 зависимостям видно, что при $N \geq 128$ ГА может оптимизировать расписание, увеличивая загруженность рассматриваемых ресурсов. При этом больший выигрыш по загруженности наблюдается при относительно низких значениях общего числа процессов N. При увеличении N выигрыш стабилизируется в коридоре приблизительно от 0.5 до 1%.

Такой резкий скачек разницы между эффективностью двух алгоритмов объясняется тем, что при меньшем числе процессов алгоритм обратного заполнения имеет меньше возможностей по выбору окон. В то время как широкий выбор окон алгоритмом ведет к более равномерной загруженности ресурсов. С увеличением числа процессов N разница в загруженности вычислительных ресурсов в расписаниях, полученных данными алгоритмами, стабилизируется.

Значения загруженности ресурсов, вычисляемые по формуле (2.9) (см. раздел 2.4), рассматриваются безотносительно к длинам расписаний. Для сравнения зависимости выигрыша в загруженности от значения N_p заданий в очереди, удобнее рассматривать зависимость относительной загруженности $w_l = (w_I - w_0) \cdot l$ от N_p, где l — длина расписания.

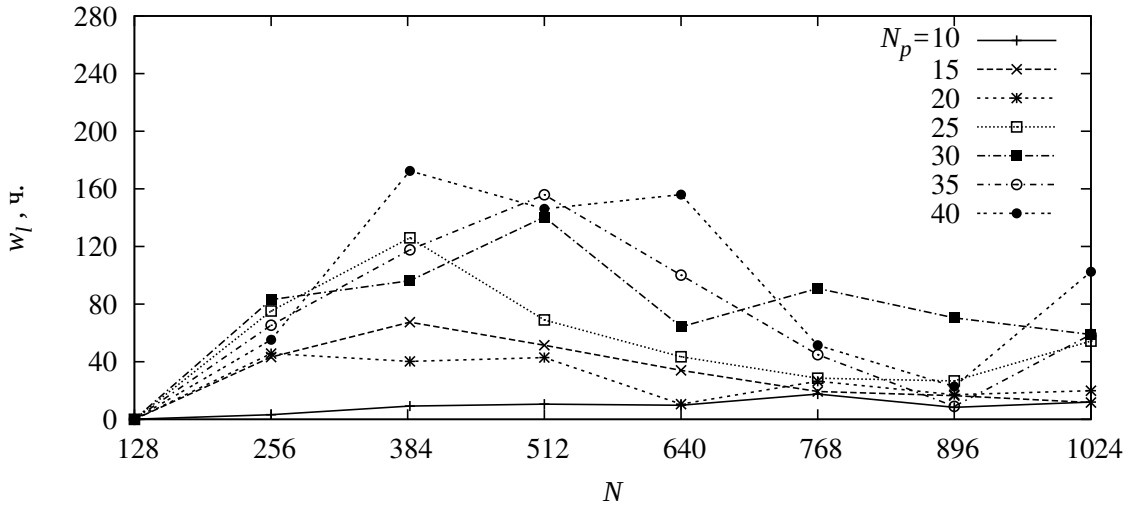

Рис. 3.11. Зависимость относительной загруженности w_l от максимального числа процессов одного задания N_p

На рис. 3.11 изображены графики зависимостей относительной загруженности w_l от значений N_p и N. Из данных зависимостей видно, что с увеличением "параллельности" заданий ГА в основном эффективнее справляется с загрузкой ресурсов, чем алгоритм *Backfill* для заданных вычислительных ресурсов. При малых значениях N_p выигрыш в загрузке не столь значителен.

Практический выигрыш можно увидеть на зависимости сэкономленного времени простаивания $(\Delta_0 - \Delta_I)$ (см. формулу (2.10)) от значений N и N_p на следующем рисунке.

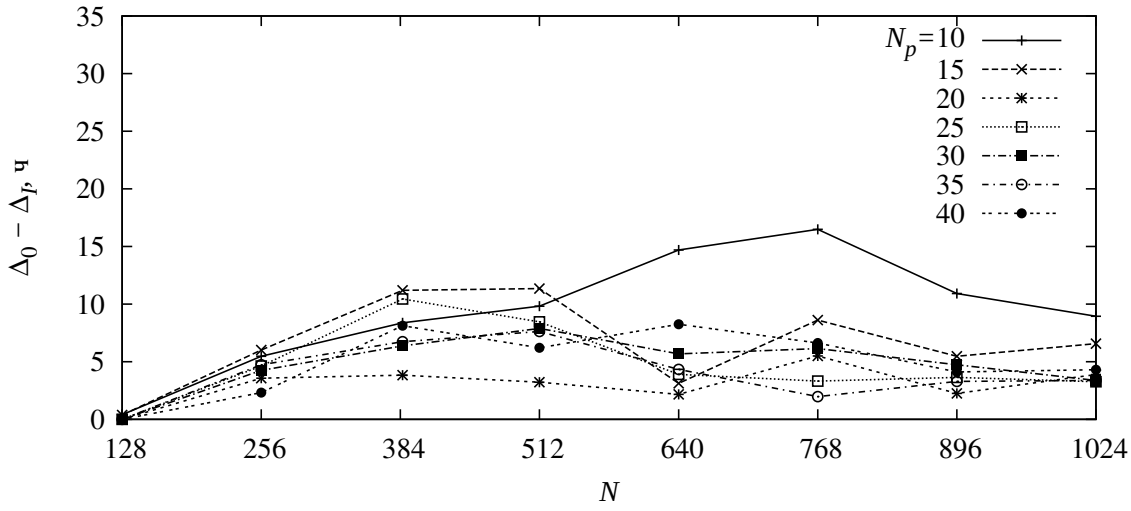

Рис. 3.12. Зависимость сэкономленного времени простаивания ($\Delta_0 - \Delta_I$) от числа процессов N и максимального числа всех процессов одного задания N_p

По изображенным на рисунке 3.12 зависимостям видно, что применяя разработанный ГА к задаче составления расписания выполнения параллельных заданий на указанных вычислительных ресурсах можно сэкономить порядка 8-12 часов процессорного времени на один ВУ (длина идеального расписания при этом равнялась 1000 часов). Однако при $N \leq 128$ включать оптимизацию расписания, уже полученного с помощью алгоритма обратного заполнения, не целесообразно.

Значения длин l расписаний, возникающих при различном значении N_p можно оценить по зависимости, изображенной на рис. 3.13.

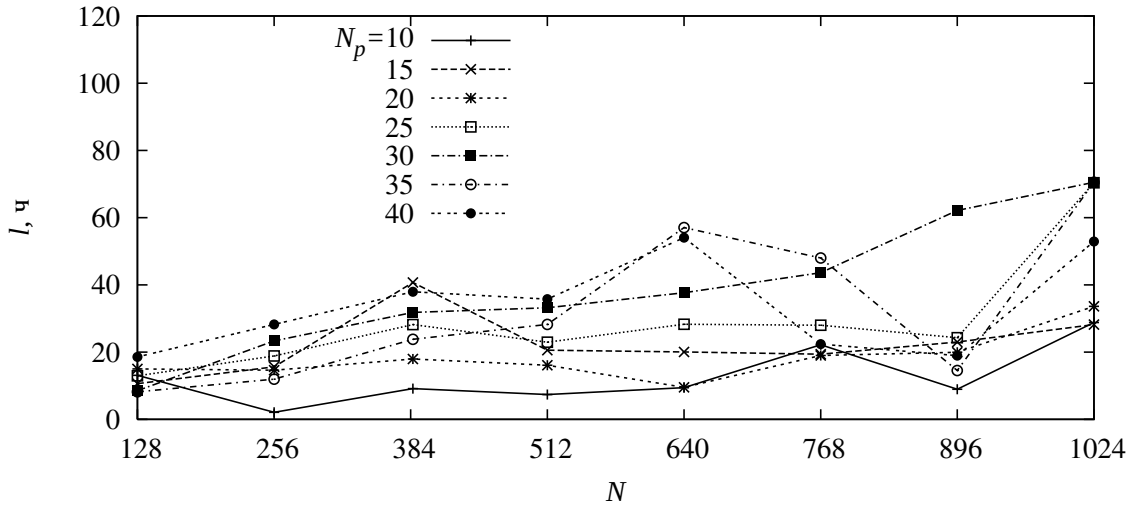

Рис. 3.13. Зависимость длины l расписания от числа всех процессов N и максимального числа процессов одного задания N_p

С ростом значений N_p возрастает и длина расписания. Это происходит ввиду увеличения отрезков времени простаивания ВУ, которые возникают из-за условия одновременности запуска всех процессов параллельного задания.

Другой характеристикой, позволяющей оценить работу алгоритмов является среднее время ожидания процессов в расписании:

$$\delta_j = \frac{1}{N} \sum_{i=1}^{N} t_i,$$

где t_i — время начала выполнения i-го процесса в лучшем расписании из всех, полученных на j-ой итерации ГА, N — число всех процессов в расписании.

На следующем рисунке показана зависимость разности $\delta_0 - \delta_I$ от числа всех процессов N всех заданий. Здесь δ_0 и δ_I — значения средних времен ожидания в расписаниях, полученных алгоритмом обратного заполнения и ГА соответственно.

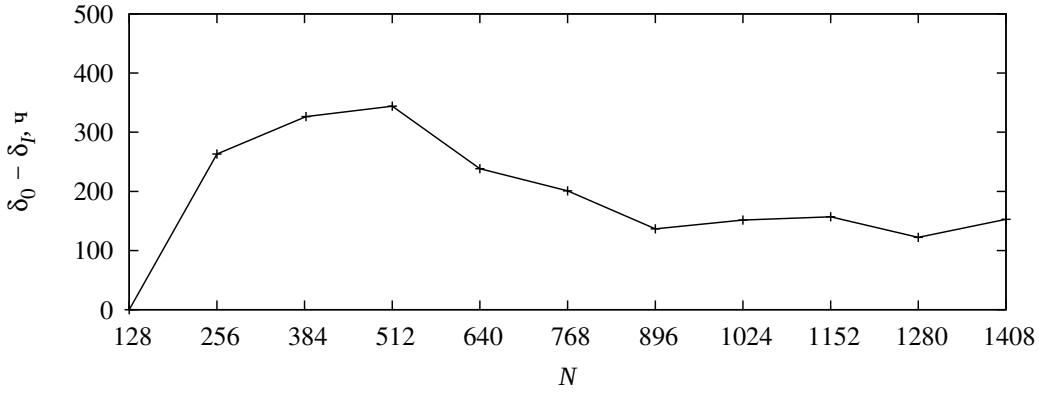

Рис. 3.14. Зависимость разности $\delta_0 - \delta_I$ от числа процессов N

Как и в случае с зависимостью $w_I - w_0$ от N, в зависимости $\delta_0 - \delta_I$ от N, показанной на рис. 3.14 при небольших N наблюдается небольшое увеличение значений $\delta_0 - \delta_I$. С ростом N выигрыш от использования ГА уменьшается и стабилизируется в некотором коридоре значений.

Таким образом, из представленных в данном разделе рисунков можно сделать вывод, что использование реализации разработанного автором алгоритма планирования параллельных заданий вместо алгоритма обратного заполнения для вычислительных ресурсов ВЦ ДВО РАН в ряде случаев целесообразно.

Заключение

В диссертационной работе предложен алгоритм планирования выполнения заданий в РВС, который может быть использован при построении распределенных вычислительных систем для решения ресурсоемких фундаментальных и прикладных задач.

Сформулируем основные результаты диссертации.

1. Проведена адаптация генетических алгоритмов для составления расписаний выполнения непараллельных и параллельных заданий в распределенной вычислительной системе. Для данных алгоритмов разработаны генетические операторы, сохраняющие непротиворечивость расписаний. Проведена адаптация алгоритма обратного заполнения с учетом ресурсных ограничений для составления расписаний в начальной популяции генетического алгоритма. Разработана форма кодирования расписания для ГА, позволяющая экономить оперативную память при составлении расписания заданий для крупных РВС. Представлен механизм учета ресурсных ограничений, а также одиночных зависимостей между заданиями в расписаниях при применении ГА. Проведен анализ влияния значений основных параметров этих генетических алгоритмов на скорость составления расписаний.

2. Представлен алгоритм планирования выполнения заданий в РВС, основанный на составлении расписаний с помощью ГА. Для плотной стыковки двух последовательно составленных расписания предложено использовать в ГА гены холостых слотов, не подвергающиеся влиянию генетических операторов.

3. Разработана и программно реализована система планирования выполнения заданий в РВС типа Grid, основанная на разработанных автором алгоритмах. Данный программный комплекс спроектирован с учетом возможности составления расписаний одновременно на мно-

жестве вычислительных узлов в сети. Эксперименты над данной системой планирования показали, что рост скорости нахождения одного субоптимального расписания не меняется с увеличением числа вычислительных узлов. Испытания для различных конфигураций РВС и числа процессов показали, что при числе процессов больше числа ВУ наблюдается выигрыш от использования ГА по сравнению с алгоритмом обратного заполнения. Проведены испытания данной системы планирования для конфигурации РВС, соответствующей вычислительным ресурсам ВЦ ДВО РАН. Численные эксперименты показали целесообразность ее применения.

Ввиду полиномиальной сложности разработанных алгоритмов составления расписаний выполнения заданий их применение для оптимизации расписаний при достаточной загруженности РВС становится оправданным. Дальнейшие исследования могут быть направлены на увеличение скорости алгоритмов, адаптацию параметров ГА для конкретной конфигурации РВС, а также на исследование адаптации генетических алгоритмов к изменяемым начальным условиям задачи составления расписания.

Литература

1. Барский А.Б. Параллельные процессы в вычислительных системах. Планирование и организация / А.Б. Барский. — М.: Радио и связь, 1990. — 256 с.

2. Бурцев В.С. Параллелизм вычислительных процессов и развитие архитектуры суперЭВМ / В.С. Бурцев. — М.: ИВВС РАН, 1997. — 152 с.

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

4. Гершуни Д.С. Планирование вычислений в системах жесткого реального времени (обзор и перспективы). Выч. техника. Системы управления. 1991. В.6 С. 4–51.

5. Головкин Б.А. Расчет характеристик и планирование параллельных вычислительных процессов / Б.А. Головкин. — М.: Радио и связь, 1983. — 272 с.

6. Журавлев Ю.И. Об экономном умножении булевых уравнений / Ю.И. Журавлев, И.М. Платоненко // ЖВМ и МФ. — 1984. — Т.24, № 1. — С. 164-166.

7. Опарин Г.А. Булевы модели синтеза параллельных планов решения вычислительных задач / Г.А. Опарин, А.П. Новопашин // Вестник НГУ. Серия: Информационные технологии. — 2008. — Том 6, Вып. 1. — С. 53-59.

8. Калиниченко Д.В. Методы и средства прогнозирования времени выполнения последовательных программ / Д.В. Калиниченко, А.П. Капитонова, Н.В. Ющенко // Методы математического моделирования. ВМиК МГУ. — 1997. — №2.

9. Коваленко В. Эволюция и проблемы Grid / В. Коваленко, Д. Корягин // Открытые системы. — 2003. — №1. — С. 23-33.

10. Коваленко В. Комплексное программное обеспечение грида вычислительного типа / В. Коваленко. — М.: ИПМ РАН, 2007. — 39 с.

11. Коваленко В.Н. Использование алгоритма Backfill в грид / В.Н. Коваленко, Д.А. Семячкин // Распределенные вычисления и Грид-технологии в науке и образовании. — Дубна, 2004. — С. 139-144.

12. Коган Б.И. Экспериментальные исследования программ / Б.И. Коган. — М.: Наука, 1988. — 184 с.

13. Коффман Э.Г. Теория расписаний и вычислительные машины / Э.Г. Коффман. — М.: Наука, 1984. — 336 с.

14. Конвей Р.В. Теория расписаний / Р.В. Конвей, В.Л. Максвелл, Л.В. Миллер. — М.: Наука, 1975. — 360 с.

15. Корнеев В.В. Параллельные вычислительные системы / В.В. Корнеев. — М.: Нолидж, 1999. — 320 с.

16. Корнеев В.В. Архитектуры с распределенной разделяемой памятью / В.В. Корнеев // Открытые системы. — 2001. — №3. — С. 15-23.

17. Кочетов Ю.А. Методы локального поиска для дискретных задач размещения / Ю.А. Кочетов. — Новосибирск: Омега Принт, 2009.

18. Курейчик В.М. Параллельные генетические алгоритмы: обзор и состояние проблемы / В.М. Курейчик, Д.С.Кныш // Известия РАН. Теория и системы управления. — 2010. — №4 — С. 72-82.

19. Курейчик В.М. Генетические алгоритмы. Монография, ТРТУ, Таганрог, 1998. — 242 с.

20. Лацис А. Как построить и использовать суперкомпьютер / А. Лацис. — М.: Бестселлер, 2003. — 240 с.

21. Липаев В.В. Распределение ресурсов в вычислительных системах / В.В. Липаев. — М.: Статистика, 1979. — 248 с.

22. Майника Э. Алгоритмы оптимизации на сетях и графах / Э. Майника. — М.: Мир, 1981. — 328 с.

23. Михалевич В.С. Методы последовательной оптимизации в дискретных сетевых задачах оптимального распределения ресурсов / В.С. Михалевич, А.И. Кукса. — М.: Наука, 1983. — 208 с.

24. Панченко Т.В. Генетические алгоритмы / Т.В. Панченко. — Астрахань: Издательский дом «Астраханский университет», 2007. — 87 с.

25. Пересветов В.В., Сапронов А.Ю., Тарасов А.Г. Вычислительный кластер бездисковых рабочих станций. Препринт № 83. / В.В. Пересветов, А.Ю. Сапронов, А.Г. Тарасов. — Хабаровск: ВЦ ДВО РАН, 2005. — 50 с.

26. Пересветов В.В., Сапронов А.Ю., Тарасов А.Г., Шаповалов Т.С. Удаленный доступ к вычислительному кластеру ВЦ ДВО РАН / В.В. Пересветов, А.Ю. Сапронов, А.Г. Тарасов, Т.С. Шаповалов // Вычислительные технологии. Новосибирск: ИВТ СО РАН. — 2006. — Т.11 (Специальный выпуск). — С. 45-51.

27. Пересветов В.В., Сапронов А.Ю., Тарасов А.Г., Шаповалов Т.С. Организация работы вычислительного кластера в режиме удалённого доступа. Препринт № 110 / В.В. Пересветов, А.Ю. Сапронов, А.Г. Тарасов, Т.С. Шаповалов // Хабаровск: ВЦ ДВО РАН, 2007.

28. Пономаренко В.С., Листровой С.В., Минухин С.В., Знахур С.В. Методы и модели планирования ресурсов в Grid-системах / В.С. Пономаренко, С.В. Листровой, С.В. Минухин, С.В. Знахур. — Харьков: Издательский Дом “ИНЖЭК”, 2008. — 408 с.

29. Попков Ю.С. Макросистемы и Grid-технологии: моделирование динамических стохастических сетей / Ю.С. Попков // Проблемы управления. — 2003. — №8 — С. 10-20.

30. Смагин С.И. Шаповалов Т.С. Генетический алгоритм составления расписания выполнения параллельных заданий в распределенной вычис-

лительной системе // Вычислительные технологии. — 2010. — Т.15, №5. — С. 41-59.

31. Танаев В.С., Гордон В.С., Шафранский Я.М. Теория расписаний. Одностадийные системы / В.С. Танаев, В.С. Гордон, Я.М. Шафранский. — М.: Наука, 1984. — 384 с.

32. Таненбаум Э. Архитектура компьютера / Э. Таненбаум. — СПб.: Питер, 2002. — 704 с.

33. Тарасов А.Г., Шаповалов Т.С., Мальковский С.И. Интеграция вычислительных ресурсов ТОГУ и ВЦ ДВО РАН с применением Grid-технологий / А.Г. Тарасов, Т.С. Шаповалов, С.И. Мальковский // Материалы международной научно-практической конференции “Суперкомпьютеры: вычислительные и информационные технологии”. — Хабаровск: Изд-во ТОГУ, 2010. — С. 133-138.

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

35. Филамофитский М.П. Система поддержки метакомпьютерных расчетов X-Com: архитектура и технология работы / М.П. Филамофитский // Вычислительные методы и программирование. — 2004. — Т.5. — № 2 — С. 123-137.

36. Фостер Я., Кессельман К., Тьюке С. Grid-службы для интеграции распределенных систем / Я. Фостер, К. Кессельман, С. Тьюке // Открытые системы. — 2003. — №1. — С. 20-26.

37. Шаповалов Т.С. Генетический алгоритм составления расписания запуска параллельных заданий в Grid / Т.С. Шаповалов // Информатика и системы управления. — 2010. — Т.4. №26. — С. 115–126.

38. Шаповалов Т.С. Система планирования выполнения заданий в Grid, основанная на составлении расписаний генетическим алгоритмом / Т.С. Шаповалов // Материалы международной научно-практической конференции “Суперкомпьютеры: вычислительные и информационные технологии”. — Хабаровск: Изд-во ТОГУ, 2010. — С. 151-153.

39. Шаповалов Т.С. Об интеграции диспетчера заданий в Grid-инструментарий Globus Toolkit. / Т.С. Шаповалов // Материалы XXXV Дальневосточной математической школы-семинара имени академика Е.В. Золотова. — Владивосток: ИАПУ ДВО РАН, 2010. — С. 904-907.

40. Шаповалов Т.С. Параллельный алгоритм планирования заданий для распределённых гетерогенных вычислительных систем. Geneur. Свидетельство об официальной регистрации программы для ЭВМ № 2009612081. — М.: Федеральная служба по интеллектуальной собственности, патентам и товарным знакам, 2009.

41. Шаповалов Т.С. Эффективность системы планирования, основанной на генетическом алгоритме, для распределенных гетерогенных вычислительных систем / Т.С. Шаповалов // Материалы межрегиональной научно-практической конференции “Информационные и коммуникационные технологии в образовании и научной деятельности” — Хабаровск: Изд-во Тихоокеан. гос. ун-та, 2009. — С. 352-358.

42. Шаповалов Т.С., Пересветов В.В. Генетический алгоритм составления расписаний для распределенных гетерогенных вычислительных систем / Т.С. Шаповалов, В.В. Пересветов // Вычислительные методы и программирование. — 2009. — Т.10, №1. — С. 159-167.

43. Шаповалов Т.С. Параллельный алгоритм планирования заданий для распределенных гетерогенных вычислительных систем. Препринт № 134 / Т.С. Шаповалов. — Хабаровск: ВЦ ДВО РАН, 2006.

44. Шаповалов Т.С. Об учете ограничений на составление расписания заданий в Grid с использованием генетических алгоритмов / Т.С. Шаповалов // Труды IX Всероссийской конференции молодых ученых по математическому моделированию и информационным технологиям — Кемерово: ИВТ СО РАН, 2008. — С. 97-98.

45. Шаповалов Т.С., Пересветов В.В., Сапронов А.Ю., Смагин С.И., Тарасов А.Г. Web и Grid технологии обеспечения доступа к ресурсам вычислительного кластера ВЦ ДВО РАН / Т.С. Шаповалов, В.В. Пересветов,

А.Ю. Сапронов, С.И. Смагин, А.Г. Тарасов // Материалы Межрегиональной конференции «Информационные и коммуникационные технологии в образовании и научной деятельности». — Хабаровск: Изд-во Тихоокеан. гос. ун-та, 2008. — С. 69-76.

46. Шаповалов. Т.С. О диспетчеризации заданий в распределенных вычислительных средах / Т.С. Шаповалов // Материалы Межрегиональной конференции «Информационные и коммуникационные технологии в образовании и научной деятельности». — Хабаровск: Изд-во Тихоокеан. гос. ун-та, 2008. — С. 356-362.

47. Шаповалов Т.С. Проект метапланировщика в Grid для Globus Toolkit на базе генетических алгоритмов / Т.С. Шаповалов // Материалы Всероссийской конференции “Современные информационные технологии для научных исследований”. — Магадан: СВНЦ ДВО РАН, 2008. — С. 86-87.

48. Шаповалов Т.С. Применение генетических алгоритмов для поиска оптимального расписания заданий в Grid / Т.С. Шаповалов // Труды международной конференции “Параллельные вычислительные технологии”. — Челябинск: Изд. ЮУрГУ, 2008. — С. 500-505.

49. Шаповалов Т.С., Тарасов А.Г., Щерба С.И. Организация Grid-сети ДВО РАН / Т.С. Шаповалов, А.Г. Тарасов, С.И. Щерба // Всероссийская научная конференция “Научный сервис в сети интернет: многоядерный компьютерный мир. 15 лет РФФИ.”. — М.: МГУ, 2007. — С. 94.

50. Шаповалов Т.С., Сапронов А.Ю. О методах планирования заданий в Grid / Т.С. Шаповалов, А.Ю. Сапронов // Труды третьей международной конференции “Распределенные вычисления и Grid-технологии в науке и образовании”. — Дубна: ОИЯИ, 2008 — С. 307-309.

51. Щерба С.И., Пересветов В.В. Сравнительный анализ эффективности программного обеспечения для вычислительных кластеров / С.И. Щерба, В.В. Пересветов // Межрегиональная научно-практическая конфе-

ренция “Информационные и коммуникационные технологии в образовании и научной деятельности”. — Хабаровск: Изд-во Тихоокеан. гос. ун-та, 2008. — С. 363-369.

52. Aarts E.H. Job Shop Scheduling by Simulated Annealing / E.H. Aarts, P.J. van Laarhoven, J.K. Lenstra // Operations Research. — 1992. — Vol. 40(1). — P. 113-125.

53. Adams J. The shifting bottleneck procedure for job shop scheduling / J. Adams, E. Balas, D. Zawack // Management Science. — 1988. — Vol. 34(3). — P. 391–401.

54. Atakan D. Biobjective Scheduling Algorithms for Execution Time–Reliability Trade-off in Heterogeneous Computing Systems / D. Atakan, Ö. Füsun // The Computer Journal. — 2005. — Vol 48(3). — P. 300-314.

55. Atakan D., Füsun Ö. Genetic Algorithm Based Scheduling of Meta-Tasks with Stochastic Execution Times in Heterogeneous Computing Systems / D. Atakan, Ö. Füsun // Cluster Computing. — 2004. — Vol. 7. — P. 177–190.

56. Baeck T. Selective pressure in evolutionary algorithms: a characterization of selection mechanisms / T. Baeck // Evolutionary Computation. IEEE World Congress on Computational Intelligence. Proceedings of the First IEEE Conference. — IEEE Press, 1994. — P. 57-62.

57. Bailey D. The NAS Parallel Benchmarks. Technical Report № RNR-94-007 / D. Bailey, E. Barszcz, J. Barton, et al. // Washington: 1994.

58. Baluja S. A massively distributed parallel genetic algorithm (mdpGA). Technical report CMU-CS-92-196 / S. Baluja // Pittsburgh: Carnegie Mellon University, 1992.

59. Barricelli A. Symbiogenetic Evolution Processes Realized by Artificial Methods / A. Barricelli // Methodos. — 1957. — Vol. 9. — P. 35-36.

60. Barricelli A. Numerical Testing of Evolution Theories: Part I / A. Barricelli // Acta Biotheoretica. — Vol. 16. — 1962. — P. 94.

61. Barricelli A. Numerical Testing of Evolution Theories: Part II / A. Barricelli // Acta Biotheoretica. — Vol. 16. — 1962. — P. 122.

62. Baker M. Cluster Computing White Paper / M. Baker // UK, Portsmouth: University of Portsmouth, 2000. — 119 p.

63. Berry M., Gordon S. Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management. 2nd edition / M. Berry, S. Gordon. — John Wiley and Sons: 2004. — 643 p.

64. Blazewicz J., Lenstra J.K. Scheduling subject to resource constraints: classification and complexity / J. Blazewicz, J.K. Lenstra // Discrete Applied Mathematics. — 1983. — Vol. 5. — P. 11-24.

65. Brian H. Cloud computing / H. Brian // Communications of the ACM. — 2008. — Vol. 51(7). — P. 9-11.

66. Boctor F.F. Some efficient multi-heuristic procedures for resource-constrained project scheduling / F.F. Boctor // European journal of operational research. — 1990. — Vol. 49(1). — P. 3-13.

67. Brune M. Managing clusters of geographically distributed high-performance computers / M. Brune, J. Gehring, A. Keller, A. Reinefeld // Concurrency — Practice and Experience. — 1999. — Vol. 11(15). — P. 887–911.

68. Bierwirth C. On permutation representations for scheduling problems / C. Bierwirth, D. Mattfeld, H. Kopfer // In 4th PPSN: 1996. — P. 310–318.

69. Bierwirth C. A generalized permutation approach to job shop scheduling with genetic algorithms / C. Bierwirth // OR Spektrum. — 1995. — Vol. 17 — P. 87–92.

70. Bruno J. Scheduling independent tasks to reduce mean finishing time / J. Bruno, E. Coffman, R. Sethi // Communications of the ACM. — 1974. — Vol. 17(7) — P. 382-387.

71. Buyya Ed.R. High Performance Cluster Computing: V.1.Architectures and Systems, V.2.Programming and applications / Ed.R. Buyya. — New Jersey: Prentice Hall PTR, 1999.

72. Buyya E.R. Economic-based Distributed Resource Management and Scheduling for Grid Computing. Doctor of Philosophy thesis / E.R. Buyya. — School of Computer Science and Software Engineering Monash University, Melbourne, Australia: 2002.

73. Buncic P. The AliEn system, status and perspectives / P. Buncic, A.J. Peters, P. Saiz. — Computing in High Energy and Nuclear Physics, La Jolla, California: 2003.

74. Collins R.J. Selection in massively parallel genetic algorithms / R.J. Collins, D.R. Jefferson // In proceedings of the 4th International Conference on Genetic Algorithms and their Applications (ICGA), San Diego CA: 1991. — P. 249-256.

75. Cohoon J.P. Genetic Placement / J.P. Cohoon, W.D. Paris // IEEE Trans. on CAD. — 1987. — Vol. 6(6). — P. 956-964.

76. Darrel W. A Genetic Algorithm Tutorial. Technical Report CS-93-103 / W. Darrel. — Department of Computer Science, Colorado State University, Fort Collins, US: 1993.

77. Davis L. Job shop scheduling with genetic algorithms / L. Davis // In proceedings of an International Conference on Genetic Algorithms and their Applications, Pittsburgh, Lawrence Erlbaum Associates: 1985. — P. 136-140.

78. Davis L. Genetic Algorithms and Simulated Annealing / L. Davis. — San Mateo: Morgan Kaufman Publisher, 1987. — 216 p.

79. Devis E.W. A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling / E.W. Devis, J.H. Patterson // Management Science. — 1975. — Vol. 21(8). — P. 944-955.

80. Devis E.W. An algorithm for optimal project scheduling under multiple resource constraints / E.W. Devis, G.E. Heidorn // Management Science. — 1971. — Vol 17(12). — P. 803-817.

81. De Jong A.K. An analysis of the behavior of a class of genetic adaptive systems. Ph.D. thesis / A.K. De Jong // University of Michigan: 1975.

82. Dean J. MapReduce: Simplified Data Processing on Large Clusters / J. Dean, S. Ghemawat // Communications of the ACM. — 2008. — Vol. 51(1). — P. 107-113.

83. Dorndorf U., Pesch E. Evolution based learning in a job shop scheduling environment / U. Dorndorf, E. Pesch // Computers Ops Res. — 1995. — Vol. 22. — P. 25-40.

84. Ellert M. Advanced Resource Connector middleware for lightweight computational Grids / M. Ellert et al. // Future Generation Computer Systems. — 2007. — Vol. 23. — P. 219-240.

85. Foster I. The anatomy of the Grid: enabling scalable virtual organizations / I. Foster, C. Kesselman, S. Tuecke // International Journal of High Performance Computing Applications. — 2001. — Vol. 15(3). — P. 200-222.

86. Foster I. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration / I. Foster, C. Kesselman, J. Nick, S. Tuecke // Computer Networks: The International Journal of Computer and Telecommunications Networking. — 2002. — Vol. 40(1). — P. 5-17.

87. Foster I., Kesselman C. The Grid 2 Blueprint for a New Computing Infrastructure. Second Edition. / I. Foster, C. Kesselman. — Elsevier, 2003.

88. Fox B., McMahon M. Genetic operators for sequencing problems. Foundations of genetic algorithms / B. Fox, M. McMahon. — Morgan Kaufmann, 1991.

89. Fox M.S. ISIS — a Knowledge-based system for factory scheduling / M.S. Fox, S.F. Smith // Expert Systems. — 1984. — Vol. 1(1). — P. 25-49.

90. Feitelson D.G. Improved utilization and responsiveness with gang scheduling in Job Scheduling Strategies for Parallel Processing / D.G. Feitelson, M.A. Jette // Lecture Notes in Computer Science. — 1997. — Vol. 1291. — P. 238–261.

91. Feitelson D.G. Metrics and benchmarking for parallel job scheduling / D.G. Feitelson, L. Rudolph // Lecture Notes in Computer Science. — 1998. — Vol. 1459 — P. 1-24.

92. Gabbert P., et al. A system for learning routes and schedules with genetic algorithms / P. Gabbert, et al. // In Proceedings of the Fourth Intl. Conf. on Genetic Algorithms, ICGA-91, Morgan Kaufmann: 1991. — P. 430-436.

93. Galantucci L. Assembly and Disassembly Planning by using Fuzzy Logic and Genetic Algorithms / L. Galantucci, G. Percoco, R. Spina // International Journal of Advanced Robotic Systems. — 2004. — Vol. 1(2). — P. 67-74.

94. Garey M. Bounds for multiprocessor scheduling with resource constraints / M. Garey, R. Graham // SIAM Journal on Computing. — 1975. — Vol. 4(2). — P. 187-200.

95. Garey M., Johnson D. Computers and intractability: a guide to the theory of NP-Completeness / M. Garey, D. Johnson. — New York: W. H. Freeman and Co., 1990. — 338 p.

96. Goldberg D. Genetic Algorithms in Search, Optimization, and Machine Learning / D. Goldberg. — Massachusetts: Addison-Wesley, 1989. — 372 p.

97. Grefenstette J.J. Parallel adaptive algorithms for function optimization. Technical report no. CS-81-19 / J.J. Grefenstette. — Nashville: Vanderbilt University, 1981.

98. Grimshaw S.A. The Legion vision of a worldwide virtual computer / S.A. Grimshaw, A.W. William // Communications of the ACM. — 1997. — Vol. 40(1). — P. 39-45.

99. Harvey W.D., Ginsberg M.L. Limited discrepancy search / W.D. Harvey, M.L.Ginsberg. — CIRL, University of Oregon, Eugene, OR, USA: 1995.

100. Havanki W.A. Treegion Scheduling for Wide Issue Processors / W.A. Havanki, S. Banerjia, T.M. Conte // In proceedings of 4th Intl. Symp. on High Performance Computer Architecture: 1998. — P. 266-276.

101. Heien E.M. Computing Low Latency Batches with Unreliable Workers in Volunteer Computing Environments / E.M. Heien, P.D. Anderson, K. Hagihara // Journal of Grid Computing. — 2009. — Vol. 7(4). — P. 501-518.

102. Hilliard M.R., et al. Machine Learning Applications to Job Shop Scheduling / M.R. Hilliard, et al. // In proceedings of the AAAI-SIGMAN Workshop on Production Planning and Scheduling. New York: ACM, 1988. — P. 728-737.

103. Hildum D. Flexibility in a knowledge-based system for solving dynamic resource-constrained scheduling problems. Umass CMPSCI Technical Report N.94-77 / D. Hildum. — University of Massachusetts, Amherst: 1994.

104. Holland J.H. Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control, and Artificial Intelligence / J.H. Holland. — Cambridge, The MIT Press: 1992.

105. Horn W.A. Single-machine job sequencing with treelike precedence ordering and linear delay penalties / W.A. Horn // SI AM Journal on Applied Mathematics. — 1972. — Vol. 23(2). — P. 189-202.

106. Horn W.A. Minimizing average flow time with parallel machines / W.A. Horn // Operations Research. — 1973. — Vol. 21(3). — P. 846-847.

107. Husbands P. Genetic algorithms for scheduling. Technical Report N.89 / P. Husbands. — AISB Quarterly: 1996.

108. Johnson T.J. An algorithm for the resource-constrainted project scheduling problem. Doctoral Thesis / T.J. Johnson. — Massachusetts Institute of Technology, Cambridge: 1967.

109. Junwei C. An agent-based resource management system for grid computing / C. Junwei, D. Spooner, J.D. Turner, S. Jarvis, D.J. Kerbyson, S. Saini, G. Nudd // 2nd IEEE/ACM International Symposium "Cluster Computing and the Grid": 2002. — P. 350-350.

110. Jackson D. Core algorithms of the Maui scheduler / D. Jackson, Q. Snell, M. Clement // Lecture Notes in Computer Science. — 2001. — Vol. 2221 — P. 87-102.

111. Kirkpatrick S., et al. Optimization by Simulated Annealing / S. Kirkpatrick, et al. // Science. — 1983. — Vol. 220 — P. 671-680.

112. Kojima K. Asynchronous parallel distributed genetic algorithm with elite migration / K. Kojima, M. Ishigame, G. Chakraborty, H. Hatsuo // International Journal of Computational Intelligence. — 2007. — Vol. 4(2) — P. 105-111.

113. Krallmann J. On the Design and Evaluation of Job Scheduling Systems / J. Krallmann, U. Schwiegelshohn, R. Yahyapour // Lecture Notes in Computer Science. — 1999. — Vol. 1659. — P. 17-42.

114. Kurowski K. User Preference Driven Multiobjective Resource Management in Grid Environments / K. Kurowski, J. Nabrzyski, J. Pulacki // In proceedings of CCGrid: 2001. — P. 114.

115. Larry J.E. The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination / Larry J.E. // In proceedings of the First Workshop on Foundations of Genetic Algorithms. Morgan Kaufmann: 1991. — P. 265-283.

116. Land A.H. An autmatic method of solving discrete programming problems / A.H. Land, A.G. Doig // Econometrica. — 1960. — Vol. 28 — P. 497-520.

117. Lawler E.L. Branch and Bound methods: a survey / Lawler E.L., Wood D.E. // Operations Research. — 1966. — Vol. 14(4) — P. 699-719.

118. Lee W. Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach / W. Lee, J.S.

Howard, P.R. Vwani, A.M. Anthony // Journal of Parallel And Distributed computing. — 1997. — Vol. 47. — P. 8-22.

119. Lifka D.A. The ANL/IBM SP Scheduling System / D.A. Lifka // Lecture Notes in Computer Science. — 1995. — Vol. 949. — P. 295-303.

120. Luis V.M. A break in the clouds: towards a cloud definition / R.M. Luis, C. Juan, L. Maik // ACM SIGCOMM Computer Communication Review. — 2008. — Vol. 39(1). — P. 50-55.

121. Mansour N. A hybrid genetic algorithm for task allocation in multicomputers / N. Mansour, G. Fox // In proceedings of the Fourth Intl. Conf. on Genetic Algorithms, ICGA-91, Morgan Kaufmann: 1991. — P. 466-473.

122. Moore M. An Accurate and Efficient Parallel Genetic Algorithm to Schedule Tasks on a Cluster / M. Moore // International Parallel and Distributed Processing Symposium (IPDPS'03): 2003. — P. 145.

123. Müller-Merbach H. Ein Verfahren zur Planung des optimalen Betriebsmitteleinsatzes bei der Terminierung von Großprojeckten / H. Müller-Merbach // Zeitschrift für wirtschaftliche Fertigung. — 1967. — Vol. 62. — P. 83-88.

124. Mualem A.W. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling / A.W. Mualem, D.G. Feitelson // In proc. 12th Intl. Parallel Processing Symposium: 1998. — P. 542-546.

125. McGregor S. Embracing Plagiarism: Theoretical, Biological and Empirical Justification for Copy Operators in Genetic Optimisation / S. McGregor, I. Harvey // Genetic Programming and Evolvable Machines. —2005. — Vol. 6(4). — P. 407-420.

126. Nakano R. Genetic Algorithms for Job-Shop Scheduling Problems / R. Nakano, T. Yamada // In proceedings of Modern Heuristic for Decision Support, London: 1997. — P. 67-81.

127. Nakano R. Conventional genetic algorithms for job-shop problems / R. Nakano, T. Yamada // In proceedings of the Fourth Intl. Conf. on Genetic Algorithms (ICGA-91), Morgan Kaufmann: 1991. — P. 474-479.

128. Neumann K. Stochastic project networks — temporal analysis, scheduling and cost minimization / K. Neumann. — Berlin: Springer-Verlag, 1990.

129. Norbis M.I. A multiobjective, multilevel heuristic for dynamic resource constrained scheduling problems / M.I. Norbis, J.M. Smith // European Journal of Operational Research. — 1988. — Vol. 33(1). — P. 30-41.

130. Nguyen X. Softening the Structural Difficulty in Genetic Programming with TAG-Based Representation and Insertion/Deletion Operators / X. Nguyen, R. McKay, D. Essam, H. Abbass // Genetic and Evolutionary Computation (GECCO-2004). — 2004. — Vol. 3103. — P. 605-616.

131. Patterson J.H. A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem / J.H. Patterson // Management Science. — 1984. — Vol. 30(7) — P. 854-867.

132. Panwalker S. A survey of scheduling rules / S. Panwalker, W. Iskander // Operations Research. — 1997. — Vol. 25(1). — P. 45-61.

133. Palmer G. An Integrated Approach to Manufacturing Planning / G. Palmer. — Huddersfield: University of Huddersfield, 1994.

134. Pesch E. Learning in Automated manufacturing: a local search approach / E. Pesch. — Heidelberg: Physica-Verlag, 1994.

135. Pugliese F. Modeling and Supporting Grid Scheduling / F. Pugliese, D. Talia, R. Yahyapour // Journal of Grid Computing. — 2008. — Vol. 6(2). — P. 195-213.

136. Qingjiang W. De-centralized job scheduling on computational Grids using distributed backfilling / Qingjiang W., Xiaolin G., Shouqi Z., Yang L. // Concurrency and Computation: Practice and Experience. — 2006. — Vol. 18(14). — P. 1829-1838.

137. Reeves C.R. Genetic algorithms and neighbourhood search / C.R. Reeves // In Evolutionary Computing. — 1994. — Vol. 865. — P. 115-130.

138. Sadeh N. Look-Ahead Techniques for Micro-Opportunistic Job Shop Scheduling. PhD thesis / N. Sadeh. — School of Computer Science, Carnegie Mellon University, Pittsburgh: 1991.

139. Sampson S.E. Local search techniques for the generalized resource constrained project scheduling problem / S.E. Sampson, E.N. Weiss // Naval Research Logistics. — 1993. — Vol. 40(5). — P. 665.

140. Schwiegelshohn U. Analysis of First-Come-First-Serve Parallel Job Scheduling / U. Schwiegelshohn, R. Yahyapour // In proceedings of the 9th SIAM Symposium on Discrete Algorithms: 1998. — P. 629-638.

141. Syswerda G. The Application of Genetic Algorithms to Resource Scheduling / G. Syswerda // In proceedings from the 4th International Conference on Genetic Algorithms. San Mateo, Morgan Kaufmann: 1990. — P. 502-508.

142. Smith S.F. The use of multiple problem decompositions in time constrained planning tasks / S.F. Smith, P. Ow // In proceedings of the ninth international joint conference on artificial intelligence. — 1985. — Vol. 2. — P. 1013-1015.

143. Schwiegelshohn U. Preemptive weighted completion time scheduling of parallel jobs / U. Schwiegelshohn // In Proceedings of the 4th Annual European Symposium on Algorithms (ESA96): 1996. — P. 39-51.

144. Schwiegelshohn U. Smart bounds for weighted response time scheduling / U. Schwiegelshohn, W. Ludwig, J.L. Wolf, J.J. Turek, P. Yu // SIAM Journal on Computing. — 1999. — Vol. 28(1). — P. 237-253.

145. Shapovalov T.S. Genetic Algorithm Based Parallel Jobs Scheduling / T.S. Shapovalov, A.G. Tarasov // In Proceedings of First Russia and Pacific Conference on Computer Technology and Applications. - Vladivostok: IACP FEB RAS, 2010. — P. 211-216.

146. Smith W.E. Various optimizers for single-stage production / W.E. Smith // Naval Research Logistics Quarterly. — 1956. — Vol.3. — P. 59-66.

147. Smith W. Scheduling with advanced reservations / W. Smith, I. Foster, V. Yaylor // IPDPS 2000. Proceedings. 14th International: 2000. — P. 127-132.

148. Smith W. Predicting application run times using historical information / W. Smith, I. Foster, V. Taylor // Lecture Notes on Computer Science. — 1998. — Vol. 1459. — P. 122-142.

149. Smith W., Wong P. Resource selection using execution and queue wait time predictions. NASA Technical report NAS-02-003 / W. Smith, P. Wong. — NASA: 2002.

150. Shubhra S.R. New Operators of Genetic Algorithms for Traveling Salesman Problem / S.R. Shubhra, B. Sanghamitra, K.P. Sankar // 17th International Conference on Pattern Recognition (ICPR'04). — 2004. — Vol. 2 — P. 497-500.

151. Srinivasan S. Characterization of backfilling strategies for parallel job scheduling / S. Srinivasan, R. Kettimuthu, V. Subramani, S. Sadayappan // In proceedings of the International Conference on Parallel Processing Workshops, IEEE Computer Society Press, Los Alamitos: 2002. — P. 514-519.

152. Simoes A. Enhancing Transposition Performance / A. Simoes, E. Costa // In proceedings of the 1999 Congress on Evolutionary Computation (CEC 99). Washington: 1999. — P. 1434-1441.

153. Simoes A. Transposition versus Crossover: An Empirical Study / A. Simoes, E. Costa // In proceedings of the Genetic and Evolutionary Computation Conference. — 1999. — Vol. 1. — P. 612-619.

154. Steuer R.E. Multiple Criteria Optimization, Theory, Computation and Application / R.E. Steuer. — Krieger Pub Co: 1986. — 546 p.

155. Smarr L. Metacomputing / L. Smarr, C. Catlett // Communications of the ACM. — 1992. — Vol. 35(6). — P. 44-52.

156. Syswerda G. Uniform crossover in genetic algorithms / G. Syswerda // In 3rd ICGA, Los Altos: 1989. — P. 2-9.

157. Tanese R. Distributed genetic algorithms / R. Tanese // In Proceedings of the 3rd International Conference on Genetic Algorithms and their application (ICGA). San Mateo, CA, Morgan Kaufmann: 1989. — P. 434-439.

158. Thain D. Grid Computing: Making The Global Infrastructure a Reality / D. Thain, T. Tannenbaum, M. Livny — John Wiley: 2003. — 1060 p.

159. Turek J.J. Scheduling parallel tasks to minimize average response time / J.J. Turek, U. Schwiegelshohn, J.L. Wolf, P. Yu // In Proceedings of the 5th SIAM Symposium on Discrete Algorithms: 1994. — P. 112-121.

160. Ulder N.L.J. Genetic local search algorithm for the traveling salesman problem / N.L.J. Ulder, E. Pesch, P.J.M. van Laarhoven, J. Bandelt, H. Aarts // In 1st PPSN: 1994. — P. 109-116.

161. Ullman J.D. Polynomial complete scheduling problems / J.D. Ullman // Operating Systems Review. — 1973. — Vol. 7(4). — P. 96-101.

162. Viet T. A Master-Slave Algorithm for Hybrid MPI-OpenMP Programming on a Cluster of SMPs / T. Viet, T. Yoshinaga, M. Sowa // Joho Shori Gakkai Kenkyu Hokoku. — 2002. — Vol. 80. — P. 107-112.

163. Vose M.D. Random heuristic search: Applications to GAs and functions of unitation / M.D. Vose, J.E. Rowe // Computer Methods in Applied Mechanics and Engineering. — 2000. — Vol. 186(2). — P. 195-220.

164. Wren A. Genetics, structures and covers — an application to scheduling. Technical Report 90.23 / A. Wren, D. Wren. — School of Computer Science, University of Leeds: 1990.

165. Yahyapour R. Design and Evaluation of Job Scheduling Strategies for Grid Computing / R. Yahyapour // Computer Engineering Institute: 2002.

166. Yamada T. Job-Shop Scheduling by Simulated Annealing Combined with Deterministic Local Search / T. Yamada, R. Nakano // Kluwer academic publishers: 1996 — P. 237-248.

167. Yamada T. A genetic algorithm with multi-step crossover for job-shop scheduling problems / T. Yamada, R. Nakano // In GALESIA-95: 1995. — P. 146-151.

168. Yoo A.B. An efficient and scalable coscheduling technique for large symmetric multiprocessor clusters / A.B. Yoo, M. Jette // Lecture Notes in Computer Science. — 2001. — Vol. 2221 — P. 21-40.

169. Zhang Y., et al. An integrated approach to parallel scheduling using gang-scheduling, backfilling and migration / Y. Zhang, H. Franke, J.E. Moreira, et al. // Lecture Notes in Computer Science. — 2001. — Vol. 2221. — P. 133-158.

170. Zhang Y. Improving parallel job scheduling by combining gang scheduling and backfilling techniques / Y. Zhang, H. Franke, J. Moreira, A. Sivasubramaniam // In proceedings of the 14th International Symposium on Parallel and Distributed Processing, IEEE Computer Society Press, Los Alamitos: 2000 — P. 133-142.

Глоссарий

В данной работе применяются следующие термины:

виртуальная организация — динамическое сообщество людей, и/или учреждений, которое совместно использует вычислительные ресурсы в соответствии с согласованными между ними правилами;

виртуальный узел — один вычислительный элемент (процессор или одно ядро процессора, если он является многоядерным), рассматриваемый в связке с соответствующими этому элементу ресурсами (различными видами машинной памяти, системным и прикладным программным обеспечением и т.д.);

вычислительный кластер — параллельная вычислительная система, включающая набор компьютеров, объединенных одной или несколькими коммуникационными сетями и управляемых единым диспетчером заданий;

вычислительный процесс — поток выполнения инструкций задания на процессоре с отдельным адресным пространством, который может включать трансляцию инструкций во внутреннее представление вычислительной машины, пошаговую интерпретацию инструкций;

вычислительные ресурсы — совокупность программных и аппаратных средств для выполнения вычислительного процесса (вычислительные узлы, среда передачи данных, прикладное программного обеспечение, системы хранения данных и т.д.);

вычислительный узел — вычислительная машина, являющаяся составной частью вычислительного кластера;

гетерогенность — свойство вычислительных ресурсов, определяющее программную и аппаратную разнородность вычислительных узлов, являющихся частью этих вычислительных ресурсов;

демон — в UNIX-подобных операционных системах выполняемая в фоновом режиме программа;

задание — указание запустить ПО в РВС с набором входных данных, а также описание необходимых ей для выполнения ресурсов и способа вывода результатов;

распределенная вычислительная система (РВС) — система, интегрирующая распределенные вычислительные ресурсы посредством сети передачи данных и системного ПО;

очередь заданий — механизм управления заданиями, при котором формируется набор заданий, ожидающих выполнения;

хромосома — упорядоченная последовательность генов, кодирующая решение в ГА;

ген — атомарный для генетических операторов элемент хромосомы, описывающий некоторое свойство или часть решения;

холостой процесс — вычислительный процесс, не выполняющий полезных вычислительных действий;

параллельное приложение — приложение, реализующее вычислительный алгоритм, в котором различные его вычислительные блоки могут независимо расчитываться на нескольких вычислительных узлах кластера в рамках отдельных процессов;

популяция — множество хромосом одного поколения в ГА;

подпопуляция — часть популяции, каким-либо образом взаимодействующая с другими частями данной популяции в ГА;

поколение — очередная популяция после прохождения одного полного цикла рекомбинации в ГА;

масштабируемость — способность системы увеличивать вычислительную мощность без существенного снижения отношения общей производительности к числу узлов при включении в состав системы дополнительных узлов;

клиент-серверная архитектура — архитектура ПО с выделенным приложением (сервером), в задачи которого входит обслуживание (предоставление сервисов) множества приложений (клиентов) посредством определённого программного интерфейса;

системное программирование — разработка средств общего ПО, в том числе операционных систем, вспомогательных программ, пакетов программ общесистемного назначения, например, автоматизированных систем управления, систем управления базами данных и т.д.;

слот — отрезок времени, выделенный расписанием процессу для его выполнения на определенном ВУ;

среда передачи данных — физическая среда, по которой происходит передача электрических сигналов, использующихся для переноса той или иной информации, представленной в цифровой форме;

схема — шаблон подобия между хромосомами в ГА;

управляющий узел — вычислительная машина, обладающая функциями управления заданиями пользователей и ресурсами вычислительной системы;

Grid — вид РВС, включающий в качестве узлов вычислительные кластеры и другие высокопроизводительные системы и представляющую согласованную, открытую и стандартизованную среду, которая обеспечивает гибкое, безопасное и скоординированное разделение ресурсов в рамках виртуальной организации;

open source software — программное обеспечение, в котором пользователю предоставлена свобода на запуск, изучение, распростране-

ние и изменение при условии соблюдения соответствующих данному виду ПО свободных лицензий.

Приложение А. Основные понятия теории генетических алгоритмов

ГА представляет собой стохастический метод поиска решения задач оптимизации. В нем сочетаются механизмы генетического наследования и естественного отбора. Алгоритм требует представления решений в форме хромосом, элементы которых называются генами. Популяцией является множество хромосом.

Классический ГА [104] (см. рис. 3.15) описывает два генетических оператора *рекомбинации*: оператор скрещивания (синонимы: кроссовер, кроссинговер) и оператор мутации. Помимо классических генетических операторов также находят применение и другие: инверсии [150], плагиата (plagiarism operator) [125], транспозиции [152, 153], вставки/удаления [130]. В результате рекомбинации возникают новые комбинации генов. Таким образом происходит исследование пространства поиска решений.

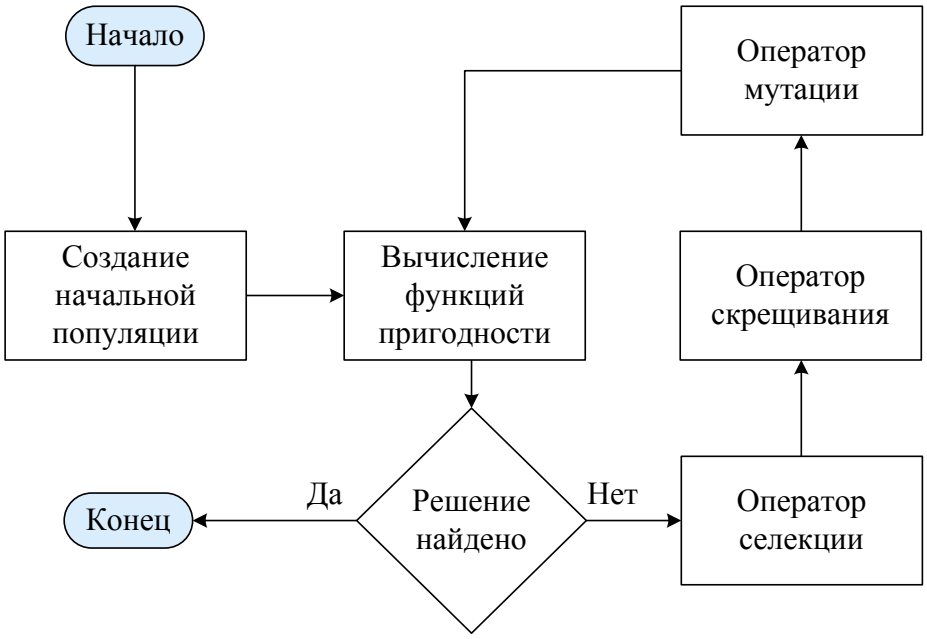

Рис. 3.15. Схема классического генетического алгоритма

Хромосомы, к которым будут применяться генетические операторы скрещивания и мутации, выбираются оператором селекции. Существуют несколько вариантов данного оператора (селекция методом рулетки, турнирный метод, ранговый метод и др.). Все генетические операторы классического ГА являются вероятностными.

Для сравнения решений используются значения их функций пригодности. Наиболее употребляемые синонимы для термина “функция пригодности”: фитнесс-функция (fitness-function), функция приспособленности, целевая функция, функция оценки, стоимостная функция. Для различных предметных областей необходимо подбирать достаточно эффективную функцию пригодности, позволяющую однозначно установить, что одно решение “лучше” другого.

Теорема Холланда о схемах [104], хотя и подвергается критике, но является основополагающей в теории. Недостатком теоремы является то, что она не применима к моделям ГА, отличающимся от классической. Помимо этого теорема не учитывает возможности мутации и скрещивания не только разрушать, но и создавать новые схемы. Схема является шаблоном подобия между хромосомами. ГА, вместо полного перебора пространства поиска, производит выборку значительного числа гиперплоскостей с высоким значением функции пригодности. Каждая гиперплоскость соответствует множеству похожих хромосом. Схема H — строка длины l, где l — длина хромосомы. Схема состоит из элементов алфавита $\{x_1, x_2, ..., x_n, *\}$, где $\{*\}$ является неопределенным символом, а x_i — уникальный ген.

Так как теорема Холланда оперирует бинарным кодом хромосом, то будем рассматривать алфавит $\{0, 1, *\}$. Схема H определяет множество всех хромосом длины l, имеющих в соответствующих позициях значения 0 или 1. Схема, не содержащая ни одного элемента $\{*\}$, является некоторой определенной хромосомой.

Характеристиками схемы являются порядок и определяющая длина (встречается термин “определенная длина”). Порядком схемы $\theta(H)$ является число фиксированных генов (в случае бинарного кодирования хромосом — 0 или 1) схемы H. Определяющая длина $\sigma(H)$ — это число элементов между первым и последним фиксированными элементами схемы H.

Строительными блоками называют схемы с высокой пригодностью, низким порядком и короткой определяющей длиной. Пригодность схемы определяется как среднее арифметическое значение функции пригодности хромосом, соответствующих данной схеме. После применения к популяции оператора селекции чаще остаются хромосомы с более высоким значением функции пригодности.

Оператор скрещивания реже разрушает схемы с более короткой определяющей длиной, а оператор мутации реже разрушает схемы низкого порядка. Поэтому подобные схемы имеют больше шансов перейти в следующее поколение.

Теорема Холланда о схемах. Пусть $m(H,t)$ — число схем H в поколении t, l — длина хромосомы, $l(H)$ — длина схемы, $l(H) <= l-1$. Также, пусть p_c — вероятность скрещивания, p_m — вероятность мутации. Тогда ожидаемое количество схем H в следующем поколении будет:

$$m(H,t+1) >= m(H,t)\frac{f(H,t)}{\overline{f}(t)}[1-p_c\frac{\sigma(H)}{l-1}](1-p_m)^{\theta(H)},$$

где $\overline{f}(t)$ — среднее значение функции пригодности хромосом популяции поколения t, а $f(H,t)$ — значение функции пригодности схемы H поколения t.

Данная теорема показывает, что строительные блоки растут по экспоненте, в то время как схемы с пригодностью ниже средней распадаются с той же скоростью. Рекомбинация и экспоненциальный рост строительных блоков ведет к формированию более пригодных строительных блоков [96].

Теорема Холланда выявляет несколько важных аспектов поведения алгоритма. Мутации с большей вероятностью разрушают схемы высокого порядка, в то время, как скрещивания с большей вероятностью разрушают схемы с большей определяющей длиной. Популяция сходится к оптимальному решению пропорционально отношению пригодности лучшей хромосомы к средней пригодности хромосом популяции. Данное отношение — мера давления отбора (selection pressure).

Увеличение либо p_c, либо p_m или уменьшение давления отбора ведет к усиленному исследованию пространства поиска, но не позволяет использовать все лучшие схемы. Уменьшение либо p_c, либо p_m или увеличение

давления отбора ведет к улучшению использования найденных схем, но тормозит исследование пространства решений в поисках новых, более пригодных схем. При проектировании ГА необходимо учитывать сохранение равновесия между тем и другим, что обычно известно как проблема “баланса исследования и использования”. Универсальная методология управления сходимостью ГА до сих пор не выработана.

Классический генетический алгоритм [163]. Пусть S — пространство поиска решений. Пусть $P \subset S$ — одна популяция. Функция пригодности f — отображение пространства поиска S на пространство положительных действительных чисел: $f : S \to R_+$. Решения в популяции представляются в виде строк, состоящих из элементов алфавита $A = \{a_i : \ i = 1, 2. \ldots, d\}$. Здесь алфавит A — конечный набор символов количеством d. Хромосомой является l-местный кортеж, элементы которого (гены) — символы алфавита A, l — длина хромосомы.

ГА осуществляет рекомбинации, начиная с популяции P_0. Популяция P_{t+1} генерируется из P_t в результате действия генетических операторов. Такая эволюция приводит к популяции, содержащей решение с достаточно большим значением функции пригодности f. В идеале, рекомбинации приведут к популяции, содержащей только хромосомы, кодирующие расписания $s^* \in S$, $f(s^*) = max\,(f(s))$.

Genitor [76]. В данной модели после создания начальной популяции случайным образом выбираются 2 хромосомы, к которым применяется оператор скрещивания и мутации. В результате создается только один потомок, который занимает место менее приспособленной хромосомы в популяции.

Метод прерывистого равновесия [75] основан на палеонтологической теории прерывистого равновесия, которая описывает эволюцию за счет вулканических и других достаточно резких изменений окружающей среды. После применения оператора селекции хромосомы в популяции перемешиваются случайным образом и применяются основные операторы ГА. Потомки и наиболее пригодные родительские особи смешиваются в новой популяции. Размер популяции меняется в зависимости от числа лучших с точки зрения функции пригодности хромосом.

Гибридные модели ГА [77]. Идея гибридных моделей заключается в сочетании ГА с другими методами поиска, подходящими к решаемой задаче. В данном методе чаще всего после применения генетических операторов оптимизируют решения с помощью выбранного метода и затем заносят получившиеся хромосомы в новую популяцию. В результате каждая оптимизированная таким образом хромосома получает возможность достичь локального оптимума, вблизи которого она находится. Часто гибридные алгоритмы оказываются достаточно эффективными, но дополнительные вычисления могут неприемлемо увеличивать время поиска решения. Примером гибридной модели может служить объединение ГА и метода имитации отжига [78].

Принимались [121] небезуспешные попытки объединить ГА с методом градиентного спуска для составления расписаний вычислительного кластера. Подобный гибридный алгоритм с применением предметно-зависимой информации позволяет находить расписания более эффективно, чем другие методы.

Модель CHC [115]. Название данной модели расшифровывается как “Cross-population selection, Heterogeneous recombination and Cataclysmic mutation”. Выбирается пара хромосом с большим хеммингловым расстоянием между ними. В результате скрещивания потомку переходит половина генов каждой родительской хромосомы. Дублирование хромосом не допускается. Используют “стрессовый” вариант оператора мутации с изменением более трети генов через заданное число поколений.

Параллельные генетические алгоритмы. Для ускорения нахождения решения применяют параллельные варианты ГА. При этом ускорение может достигаться как за счет выполнения бóльшего объема вычислительной работы в единицу времени, так и за счет улучшения самой модели ГА. Метод распараллеливания алгоритма зависит как от модели алгоритма, так и от архитектуры вычислительной системы. Распараллеливание ГА для ускорения поиска решения — это выполнение алгоритма в отдельных потоках на одной многопроцессорной вычислительной машине и/или “распределение” алгоритма между отдельными вычислительными узлами, объединенными сетью передачи данных. В программной реализации раз-

работанных в рамках диссертационной работы алгоритмах применяются оба вида распараллеливания.

Для распараллеливания алгоритма по потокам существует множество программных библиотек. В данном случае имеет смысл распределять между потоками непересекающиеся группы хромосом в популяции для одновременного применения к ним генетических операторов (если это позволяет выбранная модель ГА). В случае сложной в вычислительном отношении функции пригодности также можно осуществлять ее вычисление параллельно. Также можно использовать для вычисления функции пригодности специализированные процессоры, например, процессоры графических карт или сигнальные процессоры.

Модели параллельных генетических алгоритмов делятся на модели с одной глобальной популяцией и множеством подпопуляций. В первом случае выбор “родительских” хромосом осуществляется из всей популяции так, как если бы это был непараллельный ГА. Затем происходит либо применение генетических операторов, либо вычисление функции пригодности одновременно на разных вычислительных узлах или процессорах. Данный тип моделей алгоритмически не отличается от последовательных моделей и призван лишь ускорить процесс поиска решения.

Примером ГА с одной глобальной популяцией может служить модель синхронного ГА “хозяин-рабочий” (synchronous master-slave genetic algorithm) [97]. В данной модели используется один вычислительный процесс, хранящий всю популяцию в своей памяти и выполняющий применение генетических операторов самостоятельно, но для вычисления значений функции пригодности использующий “рабочие” процессы, которые могут выполняться на других вычислительных узлах. В случае, когда рабочий процесс завершает обработку данных раньше других, он может получить от управляющего процесса следующую порцию информации для вычислений. Данная модель малоэффективна в случае низкой вычислительной сложности функции пригодности.

Модели ГА со множеством подпопуляций можно разделить на островные и клеточные. Основная идея островных моделей — изолировать подпопуляции друг от друга, осуществляя их независимую эволюцию на

протяжении определенного периода времени (периода изоляции). Каждая отдельная подпопуляция может обрабатываться как на различных процессорах одной вычислительной машины, так и на удаленных узлах. В случае, если период изоляции совпадает со временем выполнения ГА, такой алгоритм носит название разделенного (partitioned genetic algorithm).

Так как каждая подпопуляция на начальном этапе содержит различные, случайно сгенерированные хромосомы, то разные подпопуляции чаще всего будут эволюционировать различными путями. Другими словами, изолированные подпопуляции имеют возможность самостоятельно исследовать пространство поиска решений [157]. Исследование этого свойства будет проведено в диссертации более полно. Правила миграции определяют топологию миграции. Различают топологии кольца, полной сети и другие [24].

Идея клеточных моделей ГА состоит в назначении одному процессору одной или (реже) нескольких хромосом, которые образуют микропопуляции. Все процессоры логически выстраиваются в многомерную решетку так, чтобы каждый имел одинаковое число “соседей”. Микропопуляции могут взаимодействовать с другими микропопуляциями только соседних узлов. Так, для двухмерной решетки каждая микропопуляция взаимодействует только с четырьмя соседними.

Оператор селекции в клеточных ГА не требует информации о значениях функций пригодности всех особей в популяции. Взамен этого, родительские хромосомы выбираются только из соседних микропопуляций. Правилами выбора родительских хромосом могут быть случайный выбор, выбор хромосом с максимальным значением функции пригодности, а также более сложные [58, 74].

Приложение Б. Форматы файлов программного обеспечения Geneur

Листинг 1. Пример содержания файла описания прикладного программного обеспечения

```
<software>
  <name> abinit </name>
  <site>
    <hostname> mercury.febras.net </hostname>
    <kperf> 1 </kperf>
    <path>/opt/abinit/bin/abinit</path>
    <acl mode=allow>
      <user> tsh </user>
      <vo> abinit_users </vo>
    </acl>
  </site>
  <site>
    <hostname> niagara.febras.net </hostname>
    <kperf> 1.2 </kperf>
    <path>/usr/local/bin/abinit</path>
  </site>
</software>
```

Листинг 2. Пример описания задания на языке JSDL-WG

```
<JobDefinition>
  <JobDescription>
    <JobIdentification>
      <JobName> NPB_BT_A_2 </JobName>
      <Description> NAS Parallel Benchmarks:
              bt, class A, 2 process</Description>
    </JobIdentification>

    <Application>
      <ApplicationName> NPB </ApplicationName>
      <ApplicationVersion> 3.3 </ApplicationVersion>
      <POSIXApplication>
        <Executable> bt.A.2 </Executable>
        <Output> bt.A.2.out </Output>
        <ProcessCountLimit> 32 </ProcessCountLimit>
        </POSIXApplication>
```

```
      </Application>

    <Resources>
      <OperatingSystem>
        <OperatingSystemType>
          <OperatingSystemName> Linux </OperatingSystemName>
          <OperatingSystemVersion> 2.6 </OperatingSystemVersion>
        </OperatingSystemType>
      </OperatingSystem>

      <CPUArchitecture>
        <CPUArchitectureName> x86 </CPUArchitectureName>
        <IndividualCPUSpeed>
          <exact> 2400 </exact>
        </IndividualCPUSpeed>

        <IndividualCPUTime>
          <UpperBoundedRange> 172800 </UpperBoundedRange>
        </IndividualCPUTime>
      </CPUArchitecture>

      <IndividualCPUCount>
        <exact> 2.0 </exact>
      </IndividualCPUCount>

      <TotalResourceCount>
        <exact> 16.0 </exact>
      </TotalResourceCount>

      <IndividualPhysicalMemory>
        <LowerBoundedRange> 1500 </LowerBoundedRange>
     </IndividualPhysicalMemory>
    </Resources>

  </JobDescription>
</JobDefinition>
```

В данном примере задание с именем NPB_BT_A_2 требует для своего выполнения операционной системы Linux версии[25] 2.6, минимум 1500 МБ оперативной памяти. Также она будет разбита на 32 процесса, которые распределятся по 16 узлам, в каждом из которых ровно 2 процессора (ядра процессора рассматриваются как отдельные процессоры). Время ее расчета на процессоре архитектуры x86 составляет порядка 172800 секунд (данные для одного процесса задания).

[25]В случае операционной системы Linux под версией понимаются первые две цифры версии ядра

Листинг 3. Пример конфигурационного файла geneur.conf:

```
# Размер популяции или одной подпопуляции в случае,
# если вычислительных модулей несколько
population_size=30

# Процент "элитных" особей в популяции
elitism_percent=10

# Вероятность мутации в процентах
mutation_percent=0.016

# Вероятность скрещивания в процентах
interbreedion_percent=15

# Число точек скрещивания
interbreedion_points=4

# Тестовое (лучшее и заранее известное) значение пригодности
test_max_fitness=0

# Максимальное отклонение в процентах значения
# функции пригодности от тестового
test_max_fitness_percent=5

# Критерий останова 1: число поколений ГА до останова
# при неизменной лучшей пригодности по популяции
stop_iterations=500

# Критерий останова 2: максимальное число поколений
# до останова
max_cicles=10000

# Нужно ли сохранять статистические данные
save_statistic=false

# Директория, куда сохранять статистические данные
stat_dir=test/stat

# Директория с файлами-описателями узлов РВС
hosts_dir=test/hosts

# Максимальное число планируемых за раз заданий.
# Если 0, то не ограничено
max_jobs=0

# Число поколений между синхронизацией лучшего решения
# с сервером планирования
sync_iterations=100
```

```
# Уровень логирования, варианты:
# silent (без логирования),
# verbose (минимальное логирование),
# debug (логирование для отладки)
log_level=verbose

# Имя файла логирования
log_file=/var/log/geneurd.log

# Сетевой адрес сервера планирования
emand_host=127.0.0.1

# Сетевой порт сервера планирования
emand_port=12340

# Сетевой порт вычислительного модуля
geneurd_port=12341
```

Листинг 4. Пример конфигурационного файла eman.conf:

```
# Имя файла логирования
log_file=/var/log/emand.log

# Сетевой порт сервера планирования
emand_port=12340

# Сетевой порт вычислительного модуля
geneurd_port=12341

# Число потоков, параллельно обрабатывающих сетевые соединения
# от вычислительных модулей
threadpool_size=8

# Период планирования
scheduling_period=300
```

Приложение В. Программное обеспечение вычислительных ресурсов ВЦ ДВО РАН

Системное и прикладное программное обеспечение (ПО) для вычислительных кластеров ВЦ ДВО РАН основано в большей степени на открытом программном коде (open source software). Открытый код позволяет без ограничений разрабатывать новые и совершенствовать старые компоненты ПО. Операционная система — Linux CentOS.

Для создания параллельных приложений на вычислительных кластерах используется технология передачи сообщений в стандарте MPI: установлены пакеты *LAM*[26], *Open MPI*[27] и *MPICH*[28], *Intel MPI* из пакета *Intel Cluster Toolkit*.

Пользователям вычислительных ресурсов доступны языки программирования и стандартные средства разработки приложений, традиционно включаемые в состав дистрибутивов операционной системы Linux. Кроме этого, на вычислительных кластерах устанавлено и коммерческое программное обеспечение *Intel Cluster Toolkit*, в которое входят компиляторы языков программирования *Intel Fortran*, *C* и *C++*, а также инструменты построения и отладки высокопроизводительного ПО, такие как *Intel Trace Analyzer and Collector* и *Intel Cluster MKL*.

Для разделения ресурсов на кластерах применяется система диспетчеризации PBS TORQUE. Задачи по распределению ресурсов PBS TORQUE осуществляет при помощи набора очередей и задач. Основным свойством очереди является выделяемое заданию процессорное (также используют астрономическое) время. Если лимит времени исчерпан, то задача снимается с соответствующим системным posix-сигналом. При снятии задачи программе пользователя кроме стандартных сигналов о завершении может быть отправлен один из свободных, определяемых приложе-

[26] http://www.lam-mpi.org/
[27] http://www.open-mpi.org
[28] http://www-unix.mcs.anl.gov/mpi/mpich1

нием сигналов. Это может быть полезно для организации приложения с контрольными точками.

Основным разделяемым ресурсом на кластере являются свободные процессоры. Если имеется достаточное количество свободных процессоров необходимых для задачи, то задача ставится на выполнение. В противном случае задача ставиться в очередь. При наличии свободных процессоров выполняться будет задача с наибольшим приоритетом. Если приоритеты равны, то задачи ставятся на выполнение по алгоритму планирования FCFS.

На кластерах можно компилировать и выполнять и непараллельные программы. При выполнении, непараллельной программе выделяется один узел кластера, который выбирает система, но можно указать и конкретный узел по его имени или IP-адресу в файле описания задания.

Важное значение в организации работы вычислительных кластеров имеет обеспечение удаленного доступа к их ресурсам. Данный доступ осуществляется тремя способами: по протоколу передачи данных ftp, по защищенному протоколу ssh и через web-интерфейс [26, 27].

Обеспечение удаленного доступа к ресурсам вычислительного кластера ВЦ ДВО РАН включает интерфейсы администратора и пользователей кластера. Разработанные автором диссертации интерфейсы основаны на web-технологиях и представляют собой защищенные наборы web-страниц (см. рис. 3.16). Система диспетчеризации сохраняет данные в реляционных таблицах СУБД MySQL, что позволяет осуществлять гибкие запросы на выборку хранимой информации.

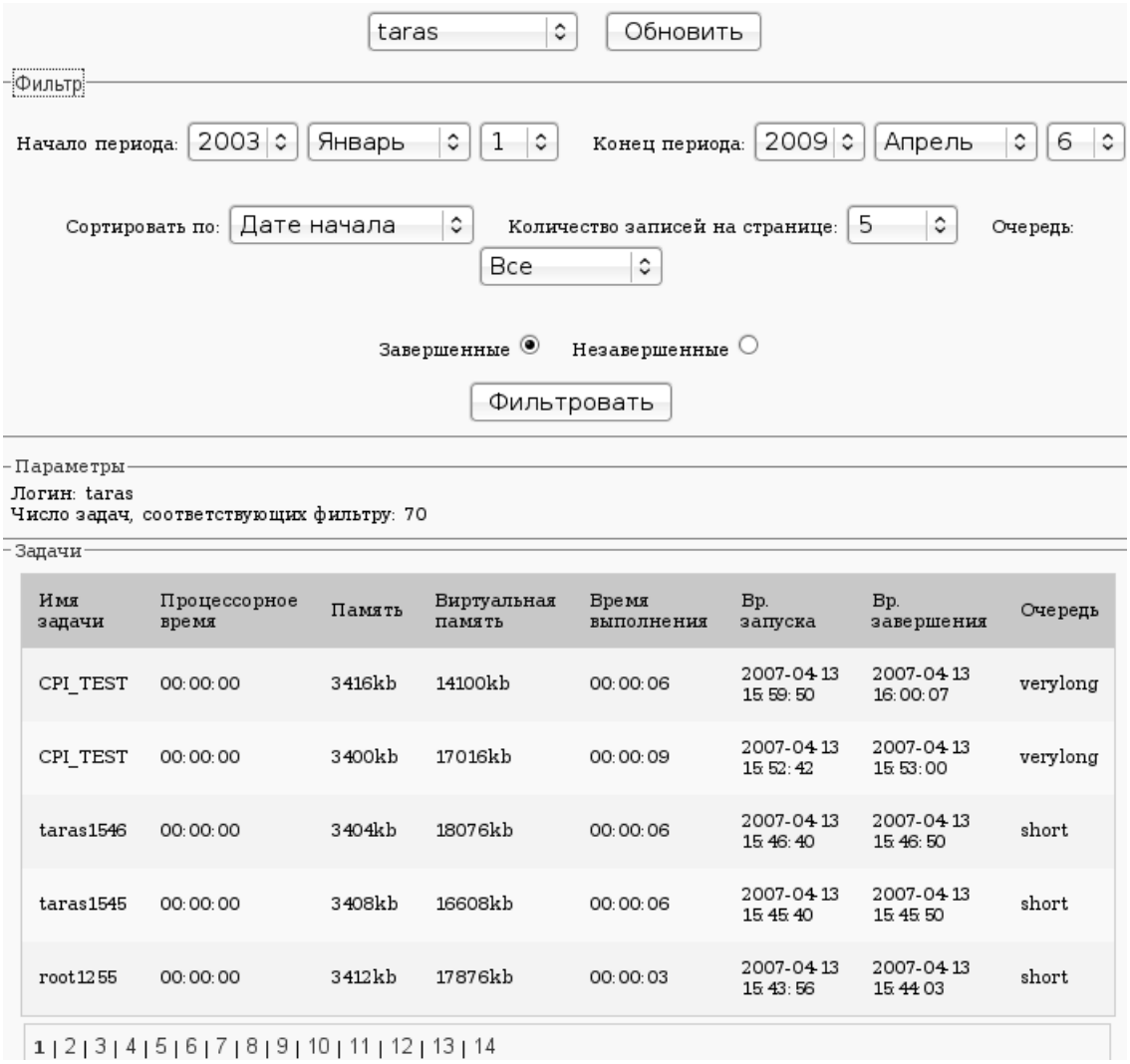

Рис. 3.16. Статистика заданий одного из пользователей на вычислительном кластере в панели управления администратора

Администратор получает возможность в удобном виде просматривать следующие статистические данные: время старта и завершения расчета задания, полное время выполнения, объем используемой виртуальной физическая памяти, используемые ресурсы процессора, принадлежность к определенной очереди задания, имя задания и другие. Администратор может использовать фильтр задач по различным параметрам. Статистика представляется в форме таблиц.

Также через web-интерфейс администратор осуществляет такие действия, как управление регистрацией пользователя по заявке, просмотр сводной статистики по всем пользователям, управление информационным сайтом поддержки (публикация новостей, рассылка писем всем пользовате-

лям). Очередь кандидатов на получение новой учетной записи представляется администратору в виде таблицы с указанием параметров регистрации. Страница статистики администратора имеет дополнительную возможность выбора пользователя, по которому нужно получить статистику, что удобно для проведения анализа активности пользователей.

Помимо статистических сведений о заданиях, в таблицах базы данных хранятся данные пользователей для связи с ними при необходимости. Личную информацию пользователь должен самостоятельно внести при регистрации своей учетной записи. Для этого применяется web-страница регистрации нового пользователя, показанная на рис. 3.17.

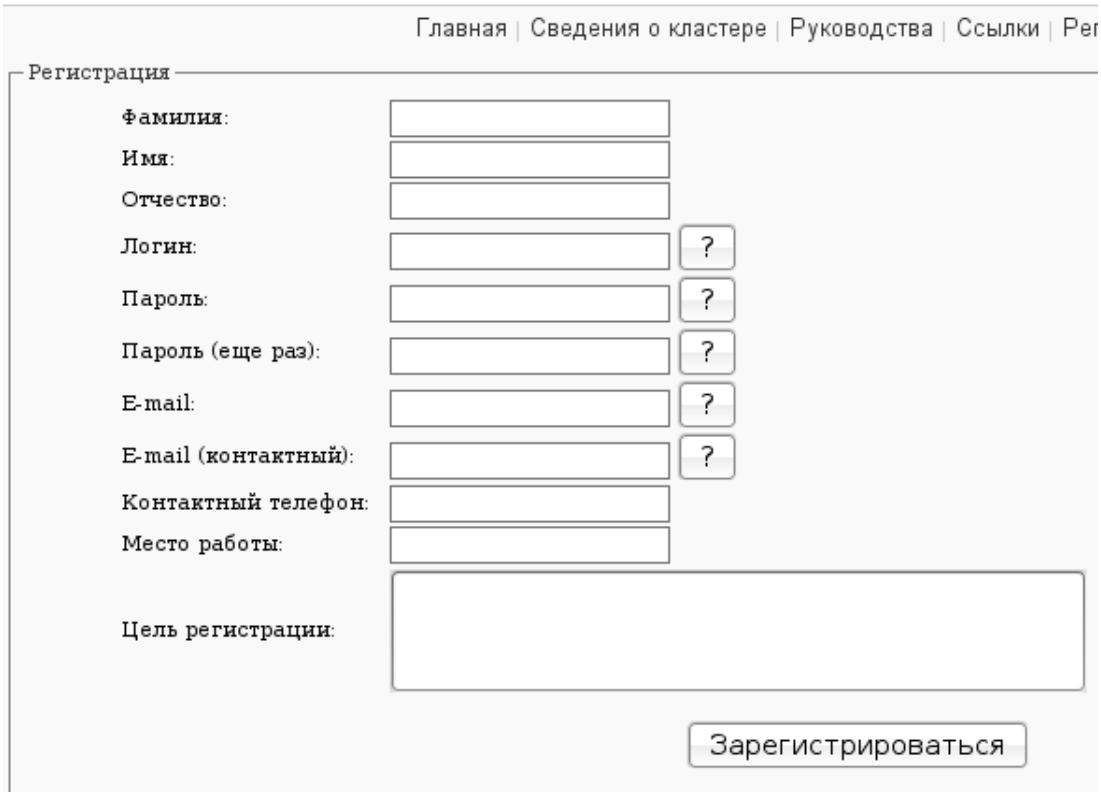

Рис. 3.17. Web-страница регистрации пользователей на вычислительных кластерах

На регистрационной форме пользователь должен заполнить ряд обязательных полей. Поля `Фамилия, Имя, Отчество` заполняются как кириллицей, так и латинскими буквами и цифрами. `Логин`: имя пользователя применяемое для входа на кластер (только латинские буквы и цифры). `Пароль`: пароль для входа на кластер (только латинские буквы и цифры). `E-mail (сообщения кластера)`: по этому адресу будут отправляться со-

общения связанные с заданиями пользователя. `E-mail (контактный)`: по этому адресу будут отправляться организационные сообщения, связанные с регистрацией пользователя, ошибками в работе и т.п. `Цель регистрации`: в этом поле пользователь должен оставить дополнительную информацию, указывающую причины регистрации.

После заполнения и отправки формы, заявка на регистрацию отправляется администратору кластера, а данные о пользователе сохранятся во временной таблице СУБД. Заявка рассматривается администратором, решение о регистрации согласовывается, в случае необходимости, с руководством. При отсутствии ошибок заполнения формы и принятия положительного решения о регистрации, пользователь будет зарегистрирован в операционной системе кластера, а личные данные из временной таблицы базы данных перенесутся в постоянную. На контактный e-mail пользователя будет отправлено сообщение, содержащее краткие инструкции для подключения.