ИПМ им.М.В.Келдыша РАН • Электронная библиотека

Препринты ИПМ • Препринт № 63 за 2007 г.

ISSN 2071-2898 (Print)
ISSN 2071-2901 (Online)

В. Н. Коваленко, Е. И. Коваленко,
Д.А. Корягин, Д. А. Семячкин

# Управление параллельными заданиями в гриде с неотчуждаемыми ресурсами



***Рекомендуемая форма библиографической ссылки:*** Управление параллельными заданиями в гриде с неотчуждаемыми ресурсами / В. Н. Коваленко [и др.] // Препринты ИПМ им. М.В.Келдыша. 2007. № 63. 28 с.
https://library.keldysh.ru/preprint.asp?id=2007-63

Р О С С И Й С К О Й  А К А Д Е М И И  Н А У К

О Р Д Е Н А  Л Е Н И Н А

ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ

ИМЕНИ М.В.КЕЛДЫША

В.Н.КОВАЛЕНКО, Е.И.КОВАЛЕНКО, Д.А.КОРЯГИН,
Д.А.СЕМЯЧКИН

# УПРАВЛЕНИЕ ПАРАЛЛЕЛЬНЫМИ ЗАДАНИЯМИ В ГРИДЕ С НЕОТЧУЖДАЕМЫМИ РЕСУРСАМИ

Москва
2007 г.

**УДК 519.68**

***В.Н. Коваленко, Е.И. Коваленко, Д.А. Корягин, Д.А. Семячкин.*** **Управление параллельными заданиями в гриде с неотчуждаемыми ресурсами.**

Задача управления параллельными заданиями много раз решалась для ближайшего аналога грида — кластерных систем, однако в условиях грида она усложняется, и для её решения требуются новые подходы. Основная трудность состоит в необходимости накопления и затем гарантированного синхронного выделения ресурсов в нескольких кластерах (коаллокация ресурсов). В настоящей работе предлагается решение этой задачи с помощью метода опережающего планирования, разработанного для практически важной формы организации грида с неотчуждаемыми ресурсами, когда они не выделяются в грид полностью, а используются в режиме разделения с их владельцами.

***Ключевые слова:*** грид, управление заданиями, планирование параллельных заданий, алгоритм обратного заполнения, резервирование ресурсов, коаллокация, неотчуждаемые ресурсы.

***V.N. Kovalenko, E.I. Kovalenko, D.A. Koryagin, D.A. Semyachkin.*** **Parallel Job Management in Grid with Non-Dedicated Resources.**

The problem of parallel job management was successfully solved many times for the nearest analogs of the grid — cluster and MPP systems, however under grid conditions it becomes more complicated and new approaches are required for its solution. The main difficulty is the necessity of accumulation and then guaranteed synchronous allocation of resources in several clusters (co-allocation of resources). In the paper a solution for this problem is proposed by means of the lookahead scheduling method designed for the practically important form of the grid with non-dedicated resources that are not granted to the grid completely but are shared with their owners.

***Key words:*** grid, job management, parallel job scheduling, Backfill, resource reservation, co-allocation, non-dedicated resources

---

***Работа выполнена при поддержке Российского фонда фундаментальных исследований (проект 05-01-00626-а) и гранта Научной школы (НШ-383.2006.9).***

## ОГЛАВЛЕНИЕ

# 1. ВВЕДЕНИЕ

Параллельное (или многопроцессорное) задание относится к базовым понятиям теории параллельных вычислений и представляет собой программу, реализованную в виде нескольких процессов, каждый из которых запускается на отдельном процессоре (процессорном узле) и в ходе своего выполнения может взаимодействовать с остальными процессами. Для обмена данными между процессами обычно используются стандартные средства передачи сообщений, такие как Message Passing Interface (MPI) [1] и Parallel Virtual Machine (PVM) [2]. Отметим, что вопрос скорости и надёжности каналов связи, используемых для межпроцессорного взаимодействия, является важным.

Параллельные задания обрабатываются на массивно-параллельных компьютерах (MPP) и кластерах, которые, как правило, используются коллективно и для которых характерна множественность заданий и ресурсов. Процесс управления, называемый **диспетчеризацией**, заключается в автоматической обработке множества заданий и включает: определение для каждого задания времени запуска и доступных исполнительных ресурсов — **планирование**, доставку входных файлов на исполнительный узел, мониторинг выполнения задания и доставку результата его выполнения. Одним из основных требований к управлению заданиями является гарантирование запуска, то есть на момент запуска в системе должны существовать подходящие для задания свободные ресурсы. Гарантирование запуска обеспечивается с помощью планирования.

По сравнению с обычными однопроцессорными заданиями, управление параллельными заданиями значительно более сложно: необходимо подбирать исполнительные ресурсы так, чтобы все его процессы стартовали и завершались одновременно — это так называемая **задача коаллокации ресурсов** [3, 4]. К настоящему времени для современных параллельных архитектур и кластерных систем разработаны методы, решающие эту задачу. Их обзору посвящен второй раздел. Рассматриваемая нами постановка задачи коаллокации в гриде и наиболее значимые, усложняющие её свойства грида приводятся в третьем разделе.

В данной работе мы предлагаем решение задачи управления параллельными заданиями в гриде с помощью метода опережающего планирования [5], применённого нами ранее для управления однопроцессорными заданиями в программном комплексе GrAS (Grid Advanced Scheduler) [6]. Метод основан на построении плана распределения ресурсов на некоторый период времени вперед и гарантировании его выполнения за счёт использования механизма предварительного резервирования, обеспечивающего детерминированность планирования. Это позволяет применить его для обслуживания параллельных заданий в гриде.

Формальная постановка задачи планирования в гриде, предполагающая совместное использование ресурсов с их владельцами, приведена в заключительной части третьего раздела. При планировании в гриде существенно, каким образом учитываются задания, находящиеся в локальных очередях кластерных ресурсов. Предлагаемый способ учёта основан на экономическом подходе, а формализация планирования опирается на представление плана в виде множества временных слотов, где каждый слот соответствует началу и концу некоторого кластерного задания. Планирование заключается в подборе слотов, удовлетворяющих ресурсному запросу задания с учётом их стоимости. В локальных системах для подбора слотов используется алгоритм Backfill [7], прямое применение которого в условиях грида, как показано в четвертом разделе, не даёт линейных оценок сложности от количества слотов. Основной результат работы приведён в пятом разделе и заключается в построении алгоритма с такой сложностью. Рассмотрена также модификация алгоритма для обеспечения самого раннего завершения задания, учитывающая различную производительность процессоров.

## 2. ПЛАНИРОВАНИЕ ПАРАЛЛЕЛЬНЫХ ЗАДАНИЙ

К настоящему времени для многопроцессорных компьютеров и кластеров разработан ряд методов, с помощью которых решается задача планирования параллельных заданий, большая часть которых описана в работах [7, 8, 9, 10, 11]. Эти методы принято разделять на две группы. Первая группа основана на идее разделения пространства ресурсов (space-sharing) между заданиями, согласно которой каждое задание получает необходимый объем ресурсов на требуемое время в эксклюзивном режиме. Методы второй группы используют разделение времени процессоров (time-sharing) между несколькими заданиями. Это означает, что сразу несколько заданий в произвольный момент времени могут разделять одни и те же вычислительные ресурсы. Описанию методов, предназначенных для обслуживания параллельных заданий, посвящены следующие два подраздела.

### 2.1. Методы разделения ресурсов

Простейшим методом планирования, основанным на разделении пространства ресурсов, является метод FCFS (First Come First Served) [9], согласно которому задание, поступившее в очередь раньше других, имеет самый высокий приоритет и, следовательно, должно быть запущено первым. Если для его запуска ресурсов оказывается недостаточно, ожидается момент времени, когда накопится требуемый ему объём свободных ресурсов, и задание будет запущено. Таким образом, порядок заданий в очереди не может быть нарушен: задания запускаются одно за другим согласно их приоритетам. Метод гарантирует запуск параллельного задания, однако неэффективно

использует ресурсы: в период их накопления для запуска самого приоритетного задания часть из них простаивает. Согласно имеющейся статистике [12] загрузка ресурсов в этом случае составляет примерно 50-80%. По этой причине метод FCFS в чистом виде в реальных системах практически не используется, а применяются его модификации, способные лучше загружать ресурсы.

Модификация «первый подходящий» (First-Fit) метода FCFS допускает нарушение порядка очереди, позволяя низкоприоритетным заданиям занимать ресурсы, оставшиеся незагруженными. Это повышает эффективность их использования, однако приводит к тому, что большая часть процессоров занимается мелкими заданиями — возникает фрагментация ресурсов. Так, даже если задание имеет самый высокий приоритет, необходимый ему объём ресурсов может никогда не образоваться и, следовательно, задание может никогда не стартовать — возникнет эффект «зависания» (starvation). Типовой способ решения этой проблемы заключается в ограничении числа низкоприоритетных заданий, способных «обогнать» более приоритетные задания. Это позволяет получить компромисс между предотвращением зависания заданий и эффективностью использования ресурсов, но требует в каждом конкретном случае подбирать число обгонов, которым необходимо ограничиваться для эффективной работы метода.

Более аккуратный способ решения предлагает метод обратного заполнения (Backfill), изначально разработанный для массивно-параллельных систем типа IBM SP2 [7], а в настоящее время достаточно широко применяющийся на практике, в том числе и в кластерных системах. В отличие от FCFS он требует от пользователей оценку времени выполнения их заданий, что позволяет выделять ресурсы заданиям не непосредственно в момент освобождения, а заблаговременно. Для этого строится план распределения ресурсов — расписание запусков заданий. При построении расписания ресурсы выделяются заданиям в порядке их приоритетов, причём задание может получить некоторые ресурсы только при условии, что они уже не отведены более приоритетным заданиям (в консервативном варианте) или самому приоритетному заданию (в агрессивном варианте). С помощью механизма предварительного резервирования метод гарантирует получение ресурсов высокоприоритетными заданиями, и, вместе с тем, допускает нарушение порядка очереди, что способствует повышению коэффициента общей загрузки ресурсов.

Под резервированием, как правило, понимается объект, состоящий из:

- множества зарезервированных ресурсов,
- времени действия резервирования,
- прав доступа, определяющих, для каких заданий доступны зарезервированные ресурсы.

Важно отметить, что в методе Backfill преследуются две конфликтующие друг с другом цели: повышение эффективности использования ресурсов путём

заполнения «дырок» в расписании мелкими заданиями и предотвращение зависания больших заданий — прогнозированием их запуска. Варьируя количеством резервирований, создаваемых при однократном планировании очереди заданий или комбинируя Backfill с другими известными методами упорядочивания заданий в очереди и оптимизационными методами, можно получить его различные варианты, увеличивающие вес одной из этих целей.

В зависимости от количества создаваемых резервирований различают два варианта метода обратного заполнения — **агрессивный** и **консервативный**. В агрессивном варианте (рис. 1) ресурсы резервируются только под самое приоритетное задание из очереди (задание 1). Это означает, что в случае, если это задание не может быть немедленно запущено, метод определяет, когда освободится требуемый ему объём ресурсов, и резервирует их под это задание. Получается, что остальные задания из очереди не могут задержать запуск самого приоритетного задания: они должны либо закончиться до начала резервирования, либо использовать те процессоры, которые не используются в резервировании под это задание (задания 3,4). Недостаток этого варианта заключается в том, что может быть задержан запуск заданий, для которых резервирование не делается (задание 2).

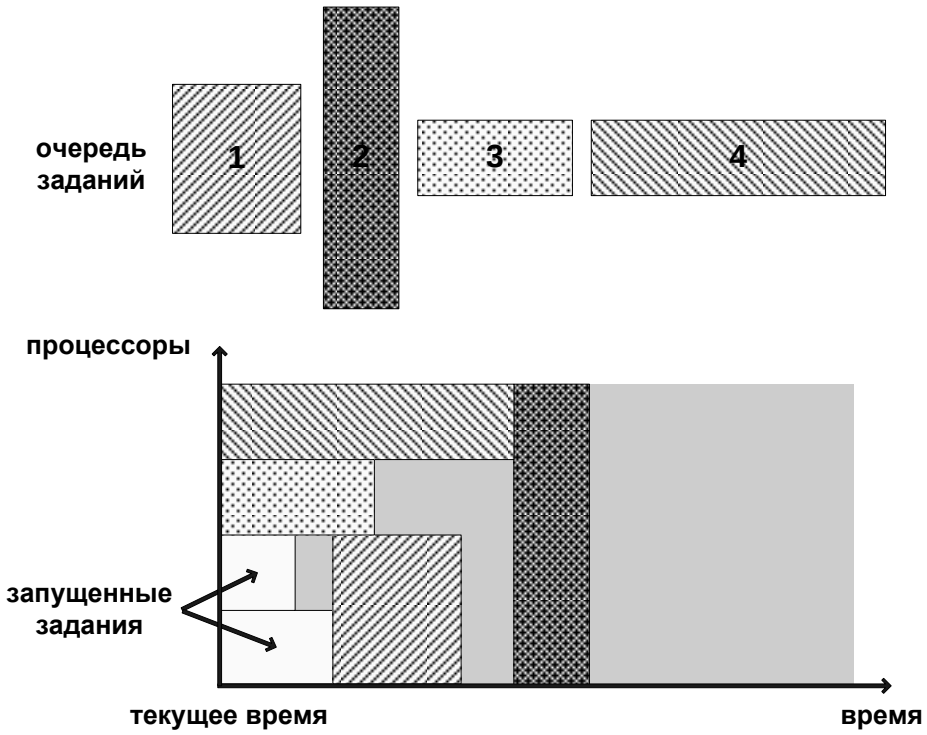


Рис. 1. Агрессивный вариант Backfill.

Консервативный вариант (рис. 2) метода лишен этого недостатка: ресурсы резервируются для всех заданий из очереди, но он менее эффективно использует ресурсы. Согласно результатам моделирования на стандартных трассах [7] существенных задержек запуска заданий в агрессивном варианте

метода не происходит. Поэтому на практике консервативный вариант метода обратного заполнения используется реже агрессивного. Еще более хорошие результаты можно получить, используя возможность ручной настройки числа резервирований, которая есть, например, в планировщике Maui [13, 14]. Проведенные эксперименты [15] показали, что оптимальным значением этого параметра является 2-4 резервирования. Некоторые системы определяют значение этого параметра по ходу работы, применяя адаптивный вариант метода, согласно которому резервирование делается под задание, которое было сильно задержано.

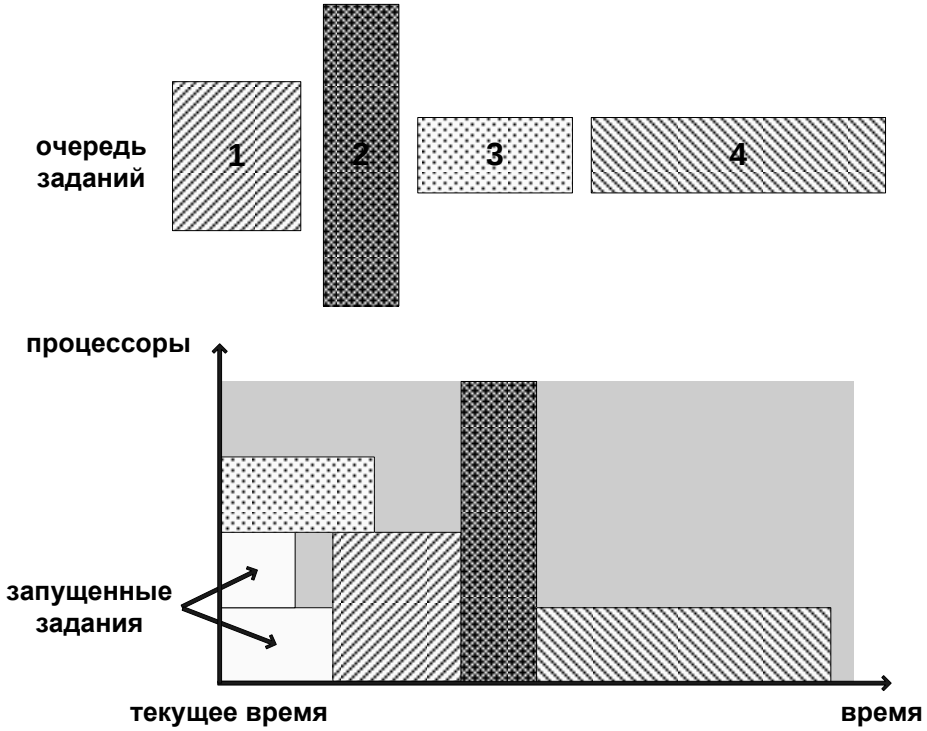


Рис. 2. Консервативный вариант Backfill.

Большинство современных планировщиков, например, Extensible Argonne Scheduling sYstem (EASY) [10], располагают задания в порядке их поступления в очередь (метод FCFS). В общем случае порядок заданий определяется их приоритетами, которые формируются на основе ряда критериев. На практике часто используется так называемая «гибкая» приоритизация, согласно которой диспетчерский приоритет задания складывается их трех типов приоритетов: административного (для выделения отдельных пользователей или проектов), пользовательского (для выделения отдельных заданий одного пользователя) и системного (для выделения задержанных заданий и обеспечения гарантии их запуска). Могут использоваться и другие методы упорядочивания заданий в очереди. Например, для повышения эффективности загрузки ресурсов используется метод Shortest Job First (SJF) [16], а для обеспечения скорейшего выполнения больших заданий — метод Largest Job First (LJF) [16].

Как правило, планировщики рассматривают задания из очереди по порядку, одно за другим. Однако, порядок, в котором задания планируются, может вести к фрагментации ресурсов. Чтобы избежать этого, можно рассматривать очередь целиком (или какую-то её часть) и пытаться найти множество заданий, которые совместно максимально эффективно загрузили бы ресурсы. Этот подход использует методы динамического программирования и рассмотрен в работе [17].

## 2.2. Методы разделения времени

Основная проблема методов планирования, основанных на разделении времени процессоров, при обслуживании параллельных заданий состоит в следующем. Процессы одного задания могут быть прерваны и перезапущены в различные моменты времени, что приводит к сильному снижению общей эффективности использования ресурсов, так как некоторое количество процессорного времени тратится на ожидание одним процессом задания перезапуска другого. Метод комплектного планирования (Gang scheduling) [11] направлен на решение этой проблемы. Его идея состоит в том, что все процессы одного задания прерываются и перезапускаются синхронно на всех процессорах (если это возможно), что позволяет избавиться от затрат на ожидание. Для этого, как показано на рис.3, процессы одного (задание A) или сразу нескольких заданий (задания B и C) группируются в комплекты так, что каждый комплект содержит одинаковое количество процессов, как правило, совпадающее с общим числом процессоров в системе. Далее процессы из каждого комплекта одновременно запускаются на этих процессорах, при этом прерывание и перезапуск происходят сразу для всех процессов, входящих в комплект.

Ранние варианты метода комплектного планирования подразумевали, что все поступающие задания сразу же запускаются. При интенсивном потоке заданий это приводит к тому, что на каждом процессоре одновременно оказывается большое количество заданий. Для части из них не хватает оперативной памяти процессорного узла и при прерывании они откачиваются во внешнюю память, что сильно ухудшает эффективность метода. Поэтому была предложена его модификация, заключающаяся в том, что задания, для которых недостаточно оперативной памяти, попадают в очередь. Для распределения заданий из очереди часто применяются методы, основанные на разделении пространства ресурсов, описанные в предыдущей части.

| комплекты \ процессоры | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | A0 | A1 | A2 | A3 | A4 | A5 |
| 1 | B0 | B1 | B2 | C0 | C1 | C2 |
| 2 | D0 | D1 | D2 | D3 | D4 | E0 |
| 3 | E1 | E2 | E3 | E4 | E5 | E6 |
| 4 | A0 | A1 | A2 | A3 | A4 | A5 |
| 5 | B0 | B1 | B2 | C0 | C1 | C2 |
| 6 | D0 | D1 | D2 | D3 | D4 | E0 |
| 7 | E1 | E2 | E3 | E4 | E5 | E6 |

Рис. 3. Комплектное планирование.

Несмотря на то, что метод комплектного планирования является наиболее известным, он не получил широкого использования в реальных системах с большим количеством процессорных узлов из-за трудности обеспечения синхронного прерывания процессов комплекта по всем вовлечённым процессорам. В частности, проблема связана с тем, что узлы массивно-параллельных и кластерных систем, как правило, работают под управлением операционных систем с различным количеством системных демонов, обеспечивающих функционирование системы. Эти демоны могут запускаться в непредсказуемое время, прерывая запущенные на узле процессы. Обеспечить синхронизацию таких прерываний по всем процессорам сложно, и это приводит к дополнительным накладным расходам.

## 3. УПРАВЛЕНИЕ ПАРАЛЛЕЛЬНЫМИ ЗАДАНИЯМИ В ГРИДЕ

Как показано в предыдущей главе, задача управления параллельными заданиями успешно решается для массивно-параллельных компьютеров (MPP) и кластерных систем, однако в условиях грида она усложняется, и для её решения требуются новые подходы.

### 3.1. Свойства грида

Среди наиболее значимых отличительных свойств грида, усложняющих задачу коаллокации, можно выделить следующие.

1. Автономность ресурсов.

Грид формируется из вычислительных ресурсов, принадлежащих различным организациям, их подразделениям или отдельным пользователям, которые осуществляют собственную политику управления этими ресурсами. Это предполагает возможность использования ресурсов их владельцами для собственных нужд, поэтому необходимо обеспечивать, чтобы выделение

ресурсов для заданий грида не противоречило существующей политике управления этими ресурсами.

2. Гетерогенность ресурсов.

Ресурсы, вовлечённые в грид, имеют различные характеристики аппаратного (например, производительность процессора, объем оперативной и дисковой памяти и т. п.) и программного обеспечения (операционная система, локальный менеджер ресурсов, программные библиотеки и т. п.). Это делает необходимым подбирать заданию только те ресурсы, характеристики которых отвечают его требованиям.

3. Пространственная распределённость.

Образующие грид вычислительные ресурсы пространственно распределены в открытой сетевой среде, в которой велики коммуникационные издержки и предъявляются повышенные требования к безопасности. Это означает, что в отличие от локальных систем в гриде передача файлов задания по сети требует большего времени, которое необходимо учитывать при планировании, чтобы успевать доставлять все части задания на исполнительные ресурсы до предполагаемого начала его выполнения.

4. Двухуровневая организация

Мы рассматриваем наиболее употребительный в настоящее время способ построения грида для организации высокопроизводительного компьютинга (High Performance Computing) из многопроцессорных установок, таких как массивно-параллельные компьютеры (MPP) и кластеры. Не ограничивая общности, далее будем употреблять для всех этих типов установок термин «кластер», а для грида, образованного таким способом, — **«грид с кластеризованными ресурсами»**. При такой организации множество компьютеров, принадлежащих одному административному домену, объединяются в кластер, и включаются в грид как один узел.

Процесс обработки заданий происходит на двух уровнях — **глобальном** (уровне планировщика грида) и **локальном** (уровне локального менеджера ресурсов). После того, как задание передается на исполнительные ресурсы, контроль над ним начинает осуществлять локальный менеджер в соответствии с собственной политикой. Управлять заданиями со стороны грида возможно только через имеющиеся интерфейсы локальных менеджеров. Большинство из них были созданы задолго до появления грида и проектировались как замкнутые системы, поэтому возможности взаимодействия программного обеспечения грида с такими менеджерами ограничены традиционными для них пользовательскими интерфейсами для запуска и управления заданиями, а также информирования о состоянии отдельных заданий и очередей. Интерфейсы, необходимые для гарантирования выделения ресурсов локальными менеджерами, такие как прогнозирование и резервирование, стали появляться только в последних версиях некоторых систем.

## 3.2. Задача коаллокации в гриде

Рассматриваемая нами постановка задачи коаллокации в гриде выглядит следующим образом. В качестве ресурсов рассматриваются вычислительные ресурсы $R=\{R_1,R_2,...,R_N\}$, каждый из которых представляет собой некоторое количество процессорных узлов — процессоров и связанную с ними оперативную и дисковую память. Ресурсы пространственно распределены (находятся в произвольных точках сети), автономны (имеют собственную политику управления) и гетерогенны (имеют различные характеристики). Предполагается, что процессорные узлы выделяются заданиям в эксклюзивном режиме: в любой момент времени на нем выполняется не более одного задания.

В систему планирования грида (планировщик) от работающих независимо друг от друга пользователей поступает неограниченный во времени поток заданий $J(T)=\{J_1(t_1),J_2(t_2),...,J_M(t_M)\}$, где $t_i$ — время поступления задания. Результатом планирования одного параллельного задания $J_i(t_i)$, требующего $P$ процессорных узлов, должен являться набор точных аллокаций ресурсов:

$$A_i=\{A_{i1}(r_{j_1},t_b,t_e),...,A_{iK}(r_{j_K},t_b,t_e)\},K=\overline{1,P}$$

($r_{j_1}$ — ресурсы, используемые для аллокации, $t_b$ — время начала аллокации, $t_e$ — время конца аллокации) таких, что: $r_{j_i}\subseteq R_{j_i},i=\overline{1,K}$, $|r_{j_1}|+|r_{j_2}|+...+|r_{j_K}|=P$, где $|r_{j_i}|$ — количество аллоцированных процессорных узлов на ресурсе. Отметим, что такая постановка задачи коаллокации позволяет подобрать ресурсы для запуска параллельного задания как в одном, так и в нескольких кластерах.

Точными называют такие аллокации, в которых определяется множество исполнительных ресурсов $r_{j_1}\cup...\cup r_{j_K}$ для выполнения задания и временной интервал $[t_b,t_e]$, на который они отводятся этому заданию. На этом интервале ресурсы считаются занятыми и недоступными для других заданий. Однако перечисленные выше свойства грида делают возможность построения точных аллокаций проблематичной. Вместо этого во многих методах и практических реализациях планировщиков результатом являются аллокации, в которых исполнительные ресурсы и время начала аллокации определены примерно. Для параллельных заданий это влечёт серьезные дефекты их обработки. Как правило, запуск параллельного задания осуществляется следующим образом. Один из процессов параллельного задания, запустившийся первым (раньше остальных) на некотором узле объявляется

главным процессом (master process). На него возлагается функция, состоящая в обеспечении синхронизации всех процессов параллельного задания. После старта этот процесс ожидает запуска всех остальных процессов параллельного задания, но не более чем некоторое заранее определенное время (таймаут ожидания). Получается, что если неизвестно точное время начала аллокаций для всех процессов, то таймаут ожидания главного процесса может истечь, а остальные процессы задания могут так и не стартовать. Это приведет к тому, что задание не будет запущено, то есть «зависнет».

Таким образом, для предотвращения «зависания» параллельного задания планирование в гриде должно быть детерминированным, то есть должно строить точные аллокации, и, кроме того, запуск заданий на исполнительных ресурсах должен осуществляться в соответствии с построенными аллокациями. Для обеспечения второго условия необходимо, чтобы:

- характеристики исполнительных ресурсов отвечали требованиям задания;
- задание было доставлено на исполнительные ресурсы к моменту начала аллокации;
- исполнительные ресурсы были к этому моменту свободны, и политика управления локальных менеджеров этих ресурсов не препятствовала их выделению для задания.

### 3.3. Использование методов приоритетного планирования в гриде

Во втором разделе настоящей работы рассмотрены методы планирования, основанные на разделении пространства ресурсов и на разделении времени. Методы разделения времени направлены на улучшение среднего времени обработки заданий. Среди методов разделения пространства ресурсов выделены методы приоритетного планирования и методы, основанные на интегральных критериях. Последние способны оптимизировать загрузку ресурсов или общее время счёта пакета заданий. Мы, однако, полагаем, что планирование в гриде в первую очередь должно быть направлено на справедливое разделение ресурсов между заданиями, обеспечивая при этом возможность управления порядком выделения ресурсов для индивидуальных заданий. Для таких целей наиболее подходящим представляется приоритетное планирование.

Согласно классификации, приведённой в работе [18], современные методы приоритетного планирования делятся на два класса. К первому классу принадлежат методы, основанные на использовании очередей заданий. К ним относятся рассмотренные во втором разделе методы FCFS, First-Fit и другие. Традиционно они применяются в кластерных системах управления, таких как PBS/Torque [19], LoadLeveler [20] и других. Однако есть пример применения и в гриде — брокер ресурсов WMS [21], входящий в состав программного обеспечения gLite [22], которое используется в наиболее крупном на

сегодняшний день инфраструктурном грид-проекте EGEE (Enabling Grids for E-sciencE) [23].

Основной принцип работы методов этого класса заключается в выделении ресурсов для стоящих в очереди заданий в порядке их приоритетов, исходя из текущего состояния ресурсов. Накопление ресурсов, необходимых для запуска параллельного задания, может быть осуществлено с помощью блокировки свободных ресурсов, что приводит к их простою в течение неопределенного времени. Механизм предварительного резервирования, предназначенный для обеспечения детерминированного планирования, трудно применим из-за отсутствия информации о времени образования требуемого заданию объема ресурсов. Таким образом, использование методов, основанных на использовании очередей, в гриде неэффективно, а основанным на них системам присущи довольно жесткие ограничения по применению: они не способны исключить такие нежелательные эффекты, как непредсказуемость времени обработки заданий, а также задержка обработки в ситуациях, когда имеются простаивающие ресурсы. Существенным недостатком этих систем является невозможность планирования параллельных заданий с синхронным выделением ресурсов в нескольких кластерах.

Второй класс включает методы, использующие для распределения ресурсов полноценный план на будущее или расписание. К ним относится рассмотренный во втором разделе метод обратного заполнения (Backfill). С их помощью автоматически обеспечивается накопление ресурсов и естественным образом реализуется механизм предварительного резервирования ресурсов. Эти свойства позволяют использовать методы этого класса для обслуживания параллельных заданий в гриде. В настоящее время существует небольшое количество систем, основанных на этих методах. К их числу относятся, например, планировщик Maui, применяющийся в локальных системах, и система Computing Center Software (CCS) [18].

Отметим, что система CCS предназначена для диспетчеризации заданий грида с одним уровнем управления, когда ресурсы отчуждаются от владельцев и целиком отдаются в грид, а все задания распределяются только с глобального уровня. Такой способ организации ресурсов мы будем называть **гридом с отчуждаемыми ресурсами**. Представляется, что такой подход не может претендовать на универсальность, однако может быть полезен в специальных условиях применения, например, когда владельцы сами не используют свои ресурсы, а выступают в роли профессиональных провайдеров.

### 3.4. Экономический подход к планированию

Мы рассматриваем форму организации ресурсов грида, предполагающую их использование совместно с владельцами — когда ресурсы не отчуждаются от владельцев, а задания поступают не только с глобального, но и с локального

уровня (рис. 4). Такой способ организации ресурсов будем называть **гридом с неотчуждаемыми ресурсами**. Его достоинство состоит в том, что грид может образовываться динамически, на некоторое время, требуемое для решения конкретной задачи, а для его создания не требуется формирования специальной ресурсной базы: достаточно лишь вовлечь уже имеющиеся ресурсы, не полностью загружаемые своими владельцами. Однако с точки зрения планирования, неотчуждаемость ресурсов усложняет ситуацию, так как задания, поступающие с локального уровня, не могут контролироваться на глобальном уровне, но должны учитываться при распределении заданий грида.

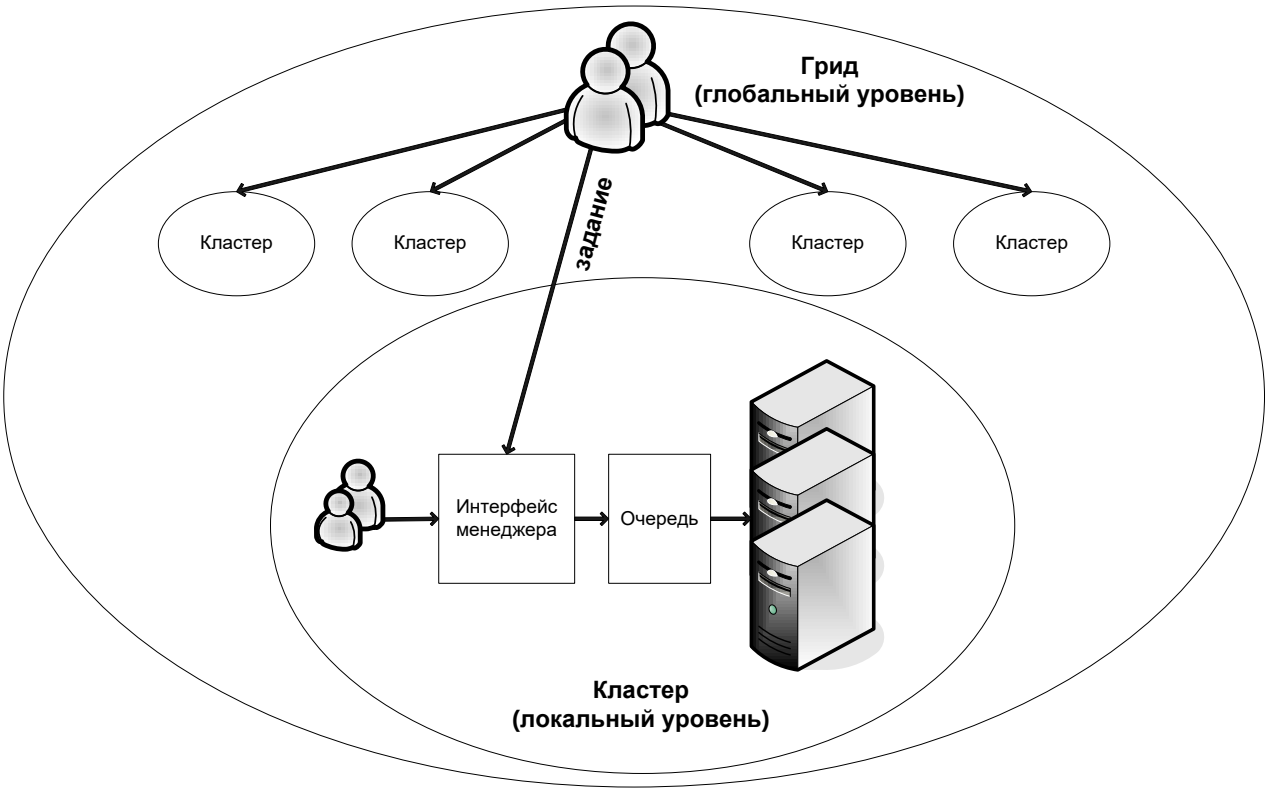


Рис. 4. Грид с неотчуждаемыми ресурсами.

Для этой практически важной формы организации ресурсов грида был предложен **метод опережающего планирования**, который может быть отнесен ко второму классу методов приоритетного планирования, рассмотренных в предыдущем разделе. Согласно этому методу процесс распределения ресурсов организуется на базе моделей рынка (экономический подход) [24]: владельцы ресурсов выступают в качестве продавцов, а пользователи — в качестве покупателей этих ресурсов. Реализуя этот подход, предлагаются следующие соглашения по разделению ресурсов и способы их поддержки при планировании.

Первое соглашение касается разделения ресурсов между пользователями. Основная идея заключается в том, что пользователь должен иметь возможность управлять скоростью получения ресурсов при планировании. Для этого им устанавливается плата за выполнение каждого задания в рамках

выделенного ему бюджета. При увеличении платы шансы быстрее выполнить задание повышаются, однако при этом ограничение бюджета не позволяет пользователю монополизировать весь ресурсный пул.

В условиях поступления двух потоков заданий — локального (поступающих с локального уровня) и глобального (поступающих с глобального уровня), необходимо каким-то образом разделять ресурсы между этими потоками. Для этого предлагается второе соглашение о разделении ресурсов: глобальное задание может занять ресурсы при условии, что плата, назначенная пользователем за выполнение задания, не меньше их стоимости. Такое соглашение позволяет осуществлять динамическую балансировку потоков заданий, обеспечивая конкуренцию глобальных и локальных заданий, и является ключевым для грида с неотчуждаемыми ресурсами. Его принципиальной особенностью является то, что стоимость ресурсов меняется во времени и определяется не только их объемами, но вычисляется с учётом приоритетов локальных заданий, которые могут занять те же ресурсы.

Наличие двух потоков заданий является важным отличием грида с неотчуждаемыми ресурсами от локальных систем и грида с отчуждаемыми ресурсами. План в этом случае содержит не только выполняющиеся в кластерах задания, но и задания, находящиеся в локальных очередях. Поэтому его можно представить в виде множества временных слотов вида: (процессорный узел, время начала, время конца, стоимость ресурса), где каждый слот соответствует началу и концу некоторого задания или свободному участку плана. Изобразить план можно в виде временной развёртки состояний ресурсов: для каждого ресурса на оси времени наносятся точки, соответствующие началу и концу слотов, которые представляются прямоугольниками со сторонами, равными времени занятия ресурса и его объему (рис. 5).

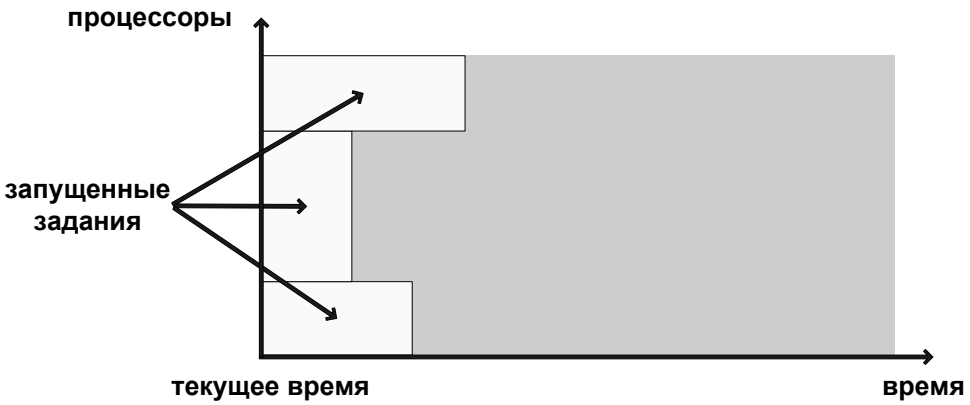


Рис. 5. Изображение плана распределения ресурсов на будущее, характерное для локальных систем и грида с отчуждаемыми ресурсами.

Согласно представленной экономической модели глобальные задания могут размещаться не только на свободных ресурсах, но и на ресурсах, на

которые претендуют локальные задания, при условии, что плата за эти задания не ниже стоимости ресурсов. Таким образом, планирование параллельных заданий заключается в подборе множества слотов, которые:

- можно синхронно выделить в количестве, необходимом заданию;
- удовлетворяют ресурсному запросу задания;
- подходят по стоимости.

Будем называть их **подходящими** для запуска задания слотами.

Назовем слоты **неподходящими** для задания, если они не удовлетворяют хотя бы одному из вышеперечисленных условий.

## 4. ИЗВЕСТНЫЕ АЛГОРИТМЫ ПОДБОРА СЛОТОВ

Процесс подбора слотов для запуска задания состоит в определении наилучшего (согласно некоторому критерию) времени, начиная с которого существуют подходящие слоты (или части слотов) в количестве, равном требуемому заданию количеству процессорных узлов. Рассмотрим существующие алгоритмы, способные решать эту задачу в условиях грида.

### 4.1. Эвристический алгоритм подбора слотов

В работе [25] приведён эвристический алгоритм, позволяющий подбирать слоты для заданий, исходя из критерия, определенного пользователем в виде целевой линейной функции. С помощью этой функции можно указать, например, что слоты необходимо подбирать так, чтобы задание запустилось как можно раньше ($F_1$) или чтобы стоимость выполнения задания была минимальной ($F_2$):

$$F_1 = -\text{StartTime}$$

$$F_2 = -\text{JobCost}$$

Кроме того, данный подход позволяет пользователю задавать сразу несколько интересующих его критериев, объединяя их в виде аддитивной функции с определяемыми весами ($F_3$).

$$F_3 = -(\text{StartTime} + 2 \cdot \text{JobCost})$$

Задача подбора слотов в этом случае сводится к максимизации линейной функции на ограниченном множестве слотов. Недостаток этого метода заключается в том, что в общем случае используемая целевая функция не

является монотонной, поэтому для определения её максимума на множестве всех слотов алгоритм должен перебрать все возможные их комбинации требуемого заданию объема, количество которых может быть велико. Для сокращения времени работы этого алгоритма в работе предложено применять эвристические методы, однако в этом случае получаемое решение уже не является наилучшим.

### 4.2. Алгоритм подбора слотов метода Backfill

Алгоритм подбора слотов, используемый в методе обратного заполнения (Backfill), разработан для локальных систем, но применим и в гриде с отчуждаемыми ресурсами. Существенно, что в таких условиях подбор слотов на очередном шаге планирования начинается с такого состояния расписания, которое содержит только запущенные задания и не содержит запланированных заданий (резервирований). Назовем это состояние начальным. В терминах слотов это означает, что каждому процессорному узлу соответствует один единственный свободный слот, начало которого совпадает со временем ожидаемого завершения задания, выполняющегося на процессорном узле (рис. 2), а конец соответствует бесконечному времени.

Это ограничение позволяет на подготовительном этапе алгоритма за один проход плана, упорядоченного по возрастанию времени начала слотов, определить, сколько процессоров будет свободно в каждой точке временной оси. Заметим, что изменение количества свободных процессоров происходит только в моменты завершения заданий. После самого последнего завершения будут свободны все процессоры, а каждое предыдущее уменьшает количество свободных процессоров на единицу.

По построенному таким образом начальному расписанию происходит последовательное размещение заданий, стоящих в очереди. В качестве критерия выбора слотов в алгоритме обычно выступает время старта задания: слоты подбираются так, чтобы запустить задания как можно раньше. Алгоритм, размещающий каждое задание, позволяет подобрать слоты для него за время, линейно зависящее от количества слотов, которое равно общему количеству заданий (запущенных и стоящих в очереди). Это обеспечивается одним проходом по списку слотов, количество элементов в котором прямо пропорционально количеству заданий в системе.

В случае грида с неотчуждаемыми ресурсами, ситуация усложняется из-за того, что расписание в начале очередного шага планирования не находится в начальном состоянии: в нем необходимо учитывать не только уже выполняющиеся в кластерах задания, но и задания, находящиеся в кластерных очередях. Учесть такие задания можно путем получения информации об их планируемом размещении (ее можно получить, моделируя работу кластерного планировщика, как показано в работе [26]).

Получается, что теперь процессорному узлу соответствуют несколько слотов, относящихся к разным промежуткам времени. Причиной этого является то, что в расписании отражены не только ожидаемые времена освобождения запущенных заданий, но и времена начал и концов резервирований, сделанных кластерными планировщиками под стоящие в локальных очередях задания (рис. 6).

Для их учёта при подсчете свободных процессоров на подготовительном этапе алгоритма подбора метода Backfill в каждой точке на временной оси, необходимо дополнительно проверять, не запланирован ли на это время старт одного или нескольких заданий, находящихся в локальных очередях. Если локальное задание запланировано, необходимо также проверить, может ли оно быть замещено глобальным заданием согласно принятому соглашению о разделении ресурсов с владельцами. Это определяется путём вычисления стоимости ресурсов и её сравнения с платой за задание, назначенной пользователем.

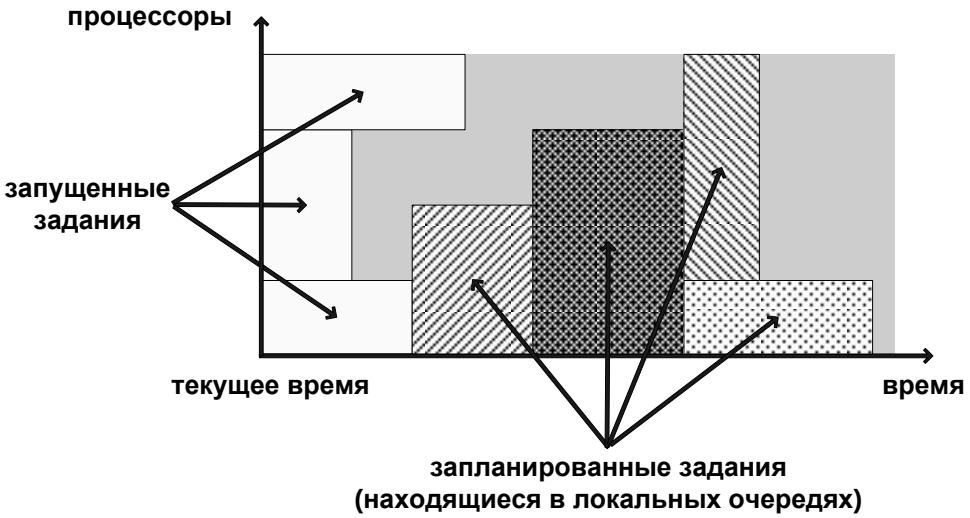


Рис. 6. Изображение плана распределения ресурсов на будущее, характерное для грида с неотчуждаемыми ресурсами.

В предположении, что на каждом процессорном узле запланировано как минимум одно локальное задание, сложность такой модификации алгоритма будет квадратичной от общего количества слотов. В условиях грида (при большом числе объектов — узлов и заданий) существенна разработка алгоритма минимальной сложности, и этот алгоритм приведён в первой части следующего раздела.

Кроме того, как было упомянуто выше, алгоритм подбирает слоты так, чтобы запустить задание как можно раньше. Представляется, однако, более полезным для пользователя подбирать слоты так, чтобы задания завершались как можно раньше. Для этого предложена модификация алгоритма, рассмотренная в заключительной части следующего раздела.

# 5. АЛГОРИТМ ПОДБОРА СЛОТОВ, УЧИТЫВАЮЩИЙ РАЗДЕЛЕНИЕ РЕСУРСОВ

## 5.1. Алгоритм подбора слотов

Для подбора слотов в условиях грида с неотчуждаемыми ресурсами предлагается алгоритм линейной сложности от количества слотов и его модификация, позволяющая заданиям завершаться как можно раньше. Алгоритм основан на совместном использовании двух представлений плана распределения ресурсов на некоторый период времени вперед в виде списков, в одном из которых слоты упорядочены по возрастанию времени начала, а во втором — по возрастанию времени конца. Алгоритм работает в два этапа.

На первом этапе осуществляется последовательный проход по первому списку (строка 15), в процессе которого подсчитывается количество слотов, которые могут быть использованы для запуска задания (строка 7). Одновременно с этим проходится второй список (строка 11), движение по которому позволяет исключить слоты, время действия которых истекло (строка 10). Как только набирается достаточное количество слотов для запуска задания, первый этап работы алгоритма завершается (строка 13), и на выходе получается время, начиная с которого для задания можно синхронно выделить требуемое количество слотов.

Первый этап алгоритма

```
/*
p1 – текущий слот из первого списка;
p2 – текущий слот из второго списка;
p1, p2 перед началом работы алгоритма содержат первые слоты из
первого и второго списков соответственно;
T – пользовательская оценка времени выполнения задания;
P – количество процессоров, необходимое заданию;
allocation_start – время возможного старта задания;
allocation_end – время возможного завершения задания;
slot
{
  start – время начала слота;
  end – время конца слота;
}
M – количество слотов, которые можно синхронно выделить во время
начала текущей аллокации;
*/

1  loop
2  {
3    allocation_start := p1.start;
4    allocation_end := allocation_start + T;
5    if ((p1.end ≥ allocation_end) &&
```

```
      (meet_job_requirements(p1)) &&
      (meet_cost(p1))) then
6     {
7       M := M + 1;
8       while (p2.end < allocation_end) do
9       {
10        if ((p2.start ≤ allocation_start) &&
           (meet_job_requirements(p2)) &&
           (meet_cost(p2))) then M := M – 1;
11        p2 := next(p2);
12      } //end while
13      if (M ≥ P) then exit loop;
14    } //end if
15    p1 := next(p1);
16  } //end loop
```

На втором этапе совершается еще один проход по первому списку с целью определения слотов для запуска задания в найденное время (строка 8). После этого задание размещается на этих слотах (строка 5), а сами слоты считаются использованными и удаляются из обоих списков (строка 6).

Второй этап алгоритма

```
1   loop
2   {
3     if ((allocation_end < p1.end) &&
       (allocation_start ≥ p1.start) &&
       (p1_meet_job_requirements) &&
       (p1_meet_cost)) then
4     {
5       use p1 for allocation;
6       remove p1 from lists;
7     } // end if
8     p1 := next(p1);
9   } //end loop
```

Таким образом, для подбора слотов для одного задания необходимо совершить два прохода по плану распределения ресурсов на первом этапе работы алгоритма и один проход на втором, следовательно, итоговая сложность разработанного алгоритма является линейной от общего количества слотов:

$$f(n) = C_1 \cdot (2 \cdot n + n) = C_2 \cdot n, C_1, C_2 = const$$

Заметим, что алгоритм не учитывает тот факт, что отдельные слоты из плана могут быть продолжением других. Предположим, что список содержит пару подходящих заданию слотов таких, что конец одного из них совпадает с

началом второго и оба слота относятся к одному процессору. Тогда эти два слота должны рассматриваться алгоритмом как один подходящий для задания слот. В противном случае, если длина задания окажется больше длины каждого из этих слотов, но меньше их суммы, алгоритм будет работать некорректно. Этот недостаток устраняется с помощью введения третьего списка, в котором слоты упорядочены по процессорам и для каждого процессора расположены непрерывно по времени. В начале работы алгоритма объявляются указатели в количестве, равном количеству процессоров, каждый из которых указывает на первый слот каждого процессора. С учётом этой корректировки в худшем случае необходимо осуществить один проход третьего списка. Таким образом, сложность алгоритма по-прежнему остается линейной.

## 5.2. Обоснование корректности алгоритма

Дадим обоснование первому этапу алгоритма (обоснование второго — очевидно).

**Утверждение.** На каждом шаге алгоритма число $M$ соответствует количеству слотов, которые можно синхронно выделить для построения аллокации для задания со временем начала, равным времени начала текущего слота.

**Доказательство.** Слоты, которые можно синхронно выделить для построения аллокации для задания со временем начала, равным времени начала текущего слота, должны быть подходящими. Покажем, что на каждом шаге алгоритма число $M$ соответствует количеству подходящих слотов. Рассуждаем количеством подходящих слотов $M$ индукцией по числу шагов (равному общему количеству слотов). После выполнения первого шага $M = 0$ (если он является неподходящим) или $M = 1$ (в противном случае). Очевидно, что $M$ соответствует количеству подходящих слотов.

Предположим, что после выполнения $K$ шагов алгоритма число $M$ соответствует количеству подходящих слотов. Необходимо доказать, что после завершения $(K+1)$-ого шага число $M$ будет также соответствовать количеству подходящих слотов.

Пусть после выполнения $K$ шагов: $M = m$, в $p_1$ содержится $(K+1)$-ый слот из первого списка, а в $p_2$ — $L$-ый слот из второго списка. В силу упорядоченности второго списка по возрастанию времени конца слоты, следующие за $L$-ым слотом во втором списке, являются подходящими. На $(K+1)$-ом шаге в качестве времени начала аллокации берется время начала $(K+1)$-ого слота, которое больше либо равно времени начала предыдущего, $K$-ого, слота. Тогда в качестве времени конца аллокации берется время начала

аллокации плюс длина задания. Если время конца $(K+1)$-ого слота меньше времени конца аллокации, то он не является подходящим, следовательно, на этом шаге число $M$ не изменяется и остается равным $m$. Тогда из предположения индукции следует, что $M$ соответствует количеству подходящих слотов. Если время конца $(K+1)$-ого слота больше либо равно времени конца аллокации, то он является подходящим, и $M$ увеличивается на единицу $(M = m+1)$. Далее для каждого слота из второго списка, начиная с $L$-ого, не являющегося подходящим и признанного подходящим на предыдущих шагах, вычитаем из $M$ единицу. Пусть таких слотов оказалось $q$ штук, тогда $M = m - q + 1$. Учитывая предположение индукции, делаем вывод, что $M$ после завершения $(K+1)$-ого шага также соответствует количеству подходящих слотов. Утверждение доказано.

### 5.3. Учёт различной производительности процессоров

Рассмотренный выше алгоритм определяет самое раннее время старта задания. Если в системе имеются процессорные узлы с разной производительностью, то более полезным, по-видимому, является критерий скорейшего завершения задания. Отметим, что, если задание выполняется на нескольких процессорах, то время его выполнения будет определяться процессором с минимальной производительностью. Введем градацию производительностей процессоров:

$$\overline{G} = (P_1, P_2, \ldots, P_k), P_1 \geq P_2 \geq \ldots \geq P_k, k = const$$

Пользовательская оценка времени выполнения задания $T$, сделанная для эталонного процессора $(P = 1)$, пересчитывается для каждой производительности $P_i$ из градации $\overline{G}$:

$$\overline{T^*} = \left( \frac{T}{P_1}, \frac{T}{P_2}, \ldots, \frac{T}{P_k} \right)$$

Алгоритм подбора слотов пытается разместить задания не на всех процессорах сразу, а на группах процессоров ( $P \geq P_i, i = \overline{1,k}$ ), начиная с самой производительной ( $P \geq P_1$ ), до тех пор, пока подходящие для задания слоты не будут найдены. При этом в качестве оценки времени выполнения задания

каждый раз берётся соответствующая пересчитанная оценка из $\overline{T^*}$. Таким образом, в первую очередь делается попытка подобрать слоты на наиболее производительных процессорах, чтобы выполнить задание как можно быстрее. В худшем случае алгоритм будет выполнен $k$ раз, поэтому в результате модифицированный алгоритм также имеет линейную сложность:

$$f(n) = C_1 \cdot (3 \cdot k) \cdot n = C_2 \cdot n, C_1, C_2 = const$$

Число групп $k$ определяет степень «огрубления» производительности процессоров, но из практических соображений не может быть слишком большим.

### 5.4. Реализация планировщика

С помощью разработанного метода и алгоритма выполнена программная реализация прототипа системы планирования параллельных заданий в гриде с неотчуждаемыми ресурсами. Прототип планировщика основан на современном подходе построения грид-систем — архитектуре веб-служб [27]. В качестве промежуточного программного обеспечения выбран признанный во всем мире как стандарт де-факто инструментарий Globus Toolkit 4 [28].

Архитектурно система состоит из набора служб: службы планирования, службы резервирования и службы приёма заданий, а также базы данных планирования и ресурсных Агентов, составляющих прогноз использования ресурсов в кластерах. Кроме того, имеется пользовательский интерфейс, аналогичный по принципу использования интерфейсу запуска заданий в Globus Toolkit.

Работа службы планирования является циклической: на каждом цикле строится план распределения заданий по ресурсам на основе информации, находящейся в базе данных планирования. Во время работы цикла все задания, поступающие от службы приема заданий, и обновления кластерных расписаний, поставляемые ресурсными Агентами, буферизуются средствами СУБД и не учитываются на текущем цикле. По окончанию цикла они актуализируются в базе данных для использования на следующих циклах. После построения плана определяется подмножество заданий, время начала аллокаций которых достаточно близко к моменту начала выполнения. Для этих заданий посредством службы предварительного резервирования выполняется резервирование ресурсов, гарантирующее их выделение в соответствии с построенными планировщиком аллокациями, и инициируется доставка заданий в кластеры средствами Globus Toolkit.

Информационная база планирования разработана с использованием СУБД с открытым исходным кодом PostgreSQL [29]. Вся информация, необходимая для планирования, представлена в виде таблиц, а алгоритм планирования

реализован хранимыми процедурами, написанными на языке PL/pgSQL [30]. Это позволило повысить скорость работы с базой данных на несколько порядков по сравнению с клиент-серверными средствами.

Ресурсный Агент основан на специальном режиме SIMULATION кластерного планировщика Maui. Этот режим позволяет моделировать процесс размещения заданий в кластере в будущем времени, обеспечивая генерацию локальных расписаний. Кроме этого, Maui поддерживает аппарат предварительного резервирования, позволяющий зарезервировать локальные ресурсы для гарантированного запуска заданий грида. На основе этого аппарата выполнена реализация службы предварительного резервирования.

## 6. ЗАКЛЮЧЕНИЕ

Несмотря на то, что идея организации высокопроизводительного компьютинга с помощью технологий грида в настоящее время воплощена в ряде инфраструктур, работающих в производственном режиме, задача коаллокации в гриде с неотчуждаемыми ресурсами, возникающая при обслуживании параллельных заданий, остается нерешённой. В данной работе предлагается один из возможных вариантов её решения.

В работе проанализированы существующие методы планирования и представлен подход к решению задачи управления параллельными заданиями с помощью метода опережающего планирования, основанного на использовании расписаний, для актуальной формы грида с неотчуждаемыми ресурсами. Рассмотрен вопрос применимости разработанных для кластеров алгоритмов подбора слотов в условиях грида. Показано, что для использования известного метода Backfill требуется его модификация, которая не даёт линейных оценок сложности в условиях грида с неотчуждаемыми ресурсами. Кроме того, алгоритм Backfill подбирает ресурсы так, чтобы запустить задание как можно раньше, что в условиях наличия процессоров разной производительности является не самым лучшим способом.

В рамках представленного в работе подхода получены также следующие результаты. Во-первых, формализована задача планирования в гриде с неотчуждаемыми ресурсами с помощью экономической модели. Во-вторых, построен алгоритм подбора ресурсов линейной сложности для запуска параллельного задания в гриде с неотчуждаемыми ресурсами. В-третьих, предложена модификация этого алгоритма также линейной сложности, учитывающая различную производительность процессорных узлов. На основе этих результатов программно реализован прототип системы планирования параллельных заданий в гриде с неотчуждаемыми ресурсами.

## 7. ЛИТЕРАТУРА

1. Message Passing Interface http://www-unix.mcs.anl.gov/mpi/

2. Parallel Virtual Machine http://www.csm.ornl.gov/pvm/
3. J. M. P. Sinaga, H. H. Mohammed, and D. H. J. Epema. A dynamic co-allocation service in multicluster systems. In 10th Job Scheduling Strategies for Parallel Processing, 2004.
   http://www.cs.huji.ac.il/~feit/parsched/jsspp04/p-04-10.ps.gz
4. E. Elmroth and J. Tordsson. A standards-based Grid resource brokering service supporting advance reservations, coallocation and cross-Grid interoperability. Submitted for journal publication, 2006.
   http://www.cs.umu.se/~elmroth/papers/elmroth_tordsson_2006_draft.pdf
5. Коваленко В.Н., Коваленко Е.И., Корягин Д.А., Любимский Э.З.. Основные положения метода опережающего планирования для грид вычислительного типа. Вестник СамГУ — Естественнонаучная серия. 2006. №4(44). с. 238–264.
   http://www.ssu.samara.ru/~vestnik/est/2006web4/ivs/200642001.pdf
6. Коваленко В.Н., Коваленко Е.И., Шорин О.Н. Разработка диспетчера заданий грид, основанного на опережающем планировании. Препринт № 133. 2005. Москва: ИПМ РАН, с. 1–28.
   http://www.gridclub.ru/library/publication.2006-01-10.2132656354
7. A. W. Mu'alem and D. G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. Parallel & Distributed Syst. 12(6), 2001. pp. 529–543.
   http://www.cs.huji.ac.il/~feit/papers/SP2backfil01TPDS.ps.gz
8. D. G. Feitelson, A Survey of Scheduling in Multiprogrammed Parallel Systems. Research Report RC 19790 (87657), IBM T. J. Watson Research Center, 1994.
   http://www.cs.huji.ac.il/~feit/papers/SchedSurvey97TR.ps.gz
9. U. Schwiegelshohn and R. Yahyapour. Analysis of First-Come-First-Serve Parallel Job Scheduling. In Proceedings of the 9th SIAM Symposium on Discrete Algorithms, 1998. pp. 629–638.
   http://www-ds.e-technik.uni-dortmund.de/~yahya/papers/cei_soda98.pdf
10. D. Lifka. The ANL/IBM SP scheduling system. In Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph (eds.), pp. 295–303, Springer-Verlag, 1995. Lect. Notes Comput. Sci. vol. 949.
    http://www.cs.huji.ac.il/~feit/parsched/jsspp95/p-95-16.pdf
11. D.G. Feitelson and M.A. Jette. Improved utilization and responsiveness with gang scheduling. In Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph (eds.), pp. 238–261, Springer-Verlag, 1997. Lect. Notes Comput. Sci. vol. 1291.
    http://www.cs.huji.ac.il/~feit/parsched/jsspp97/p-97-11.ps.gz
12. S.Hotovy. Workload evolution on the Cornell Theory Center IBM SP2. In Job Scheduling Strategies for Parallel Processing, D. G. Feitelson and L. Rudolph (eds.), pp. 27–40, Springer-Verlag, 1996. Lect. Notes Comput. Sci. vol. 1162.
    http://www.cs.huji.ac.il/~feit/parsched/jsspp96/p-96-2.ps.gz

13. Maui scheduler. http://www.supercluster.org/maui
14. D. Jackson, Q. Snell, M. Clement, Core algorithms of the Maui scheduler. In Feitelson, D. G., Rudolph, L., eds.: Job Scheduling Strategies for Parallel Processing. Volume 2221 of Lecture Notes in Computer Science., Berlin Heidelberg New York, Springer-Verlag, 2001. pp. 87–102.
http://www.cs.huji.ac.il/~feit/parsched/jsspp01/p-01-6.ps.gz
15. S-H. Chiang, A. Arpaci-Dusseau, and M. K. Vernon. The impact of more accurate requested runtimes on production job scheduling performance. In Job Scheduling Strategies for Parallel Processing, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn (eds.), pp. 103–127, Springer Verlag, 2002. Lect. Notes Comput. Sci. vol. 2537.
http://pages.cs.wisc.edu/~vernon/papers/hpjs.02jsspp.pdf
16. A. Streit. On Job Scheduling for HPC-Clusters and the dynP Scheduler. TR-00101, PC2 - Paderborn Center for Parallel Computing, Paderborn University, 2001.
http://www.fz-juelich.de/jsc/vsgc/pub/streit-2001-OJS.pdf
17. E. Shmueli and D. G. Feitelson. Backfilling with lookahead to optimize the performance of parallel job scheduling. In Job Scheduling Strategies for Parallel Processing, D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn (eds.), pp. 228–251, Springer-Verlag, 2003. Lect. Notes Comput. Sci. vol. 2862.
http://www.cs.huji.ac.il/~feit/parsched/jsspp03/p-03-12.ps.gz
18. Matthias Hovestadt, Odej Kao, Axel Keller, Achim Streit: Scheduling in HPC Resource Management Systems: Queuing vs. Planning. Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP), LNCS #2862, 2003. pp. 1–20
http://www.uni-paderborn.de/pc2/papers/files/409.pdf
19. Portable Batch System. http://www.openpbs.org
20. LoadLeveler
http://www.mhpcc.edu/training/workshop/loadleveler/MAIN.html
21. Workload Management System User and Reference Guide,
https://edms.cern.ch/file/572489/1/WMS-guide.pdf
22. gLite — Lightweight Middleware for Grid Computing. http://www.glite.org
23. Проект Enabling Grids for E-SciencE (EGEE), http://www.eu-egee.org
24. R. Buyya, J. Giddy, and D. Abramson. A case for economy Grid architecture for service-oriented Grid computing. In Proceedings of the Heterogeneous Computing Workshop, 2001.
http://www.gridbus.org/papers/ecogrid.pdf
25. C. Ernemann, V. Hamscher, and R. Yahyapour. Economic scheduling in Grid computing. In D. Feitelson and L. Rudolph, editors, Job Scheduling Strategies for Parallel Processing (Proceedings of the Eighth International JSSPP Workshop; LNCS #2537), pages 129–152. Springer-Verlag, 2002.
http://www.cs.huji.ac.il/~feit/parsched/jsspp02/p-02-4.ps.gz

26. Коваленко Е.И., Шорин О.Н. Расширение возможностей кластерных систем управления для информационного обслуживания Грид-диспетчера. Труды международной конференции «Распределённые вычисления и Грид-технологии в науке и образовании». Дубна: ОИЯИ, 2004, с. 144–147.
http://www.gridclub.ru/library/publication.2007-10-17.9333595412
27. Foster I., Kesselman C., J. Nick, Tuecke S. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. http://www.globus.org/research/papers/ogsa.pdf. Русский перевод: http://www.gridclub.ru/library/publication.2004-11-29.8307957187
28. Globus Toolkit. http://www.globus.org
29. PostgreSQL. http://www.postgresql.org
30. PL/pgSQL. http://www.postgresql.org/docs/7.4/static/plpgsql.html