Запропоновано інформаційну технологію розподілу завдань для GRID-систем, яка заснована на використанні імітаційного середовища моделювання GRASS. GRASS відтворює процес функціонування в часі елементарних подій, які протікають в GRID-системі із збереженням логіки їх взаємодії. Дане рішення дозволяє проводити обчислювальні експерименти, що реалізують різні методи розподілу, з подальшим вибором найбільш ефективного вирішення на основі збору, аналізу та інтерпретації результатів моделювання

Ключові слова: розподілені системи, GRID-система, планувальник завдань (брокер), інформаційна технологія, середа GRASS, обчислювальні ресурси, політики розподілу

Предложена информационная технология распределения заданий для GRID-систем, основанная на использовании имитационной среды моделирования GRASS. GRASS воспроизводит процесс функционирования во времени элементарных событий, которые протекают в GRID-системе с сохранением логики их взаимодействия. Данное решение позволяет проводить вычислительные эксперименты, реализующие разные методы распределения, с последующим выбором наиболее эффективного решения на основе сбора, анализа и интерпретации результатов моделирования

Ключевые слова: распределенные системы, GRID-система, планировщик заданий (брокер), информационная технология, среда GRASS, вычислительные ресурсы, политики распределения

УДК 004.051: 004.75
DOI: 10.15587/1729-4061.2016.71892

РАЗРАБОТКА ИНФОРМАЦИОННОЙ ТЕХНОЛОГИИ РАСПРЕДЕЛЕНИЯ ЗАДАНИЙ ДЛЯ GRID-СИСТЕМ С ИСПОЛЬЗОВАНИЕМ ИМИТАЦИОННОЙ СРЕДЫ МОДЕЛИРОВАНИЯ GRASS

Т. В. Филимончук
Ассистент*
E-mail: tetiana.filimonchuk@nure.ua
М. А. Волк
Кандидат технических наук, доцент*
E-mail: maksym.volk@nure.ua
И. В. Рубан
Доктор технических наук,
профессор, заведующий кафедры*
E-mail: ihor.ruban@nure.ua
В. Н. Ткачев
Кандидат технических наук, ассистент*
E-mail: vitalii.tkachov@nure.ua
*Кафедра электронных вычислительных машин
Харьковский национальный
университет радиоэлектроники
пр. Науки, 14, г. Харьков, Украина, 61166

1. Введение

GRID – это географически распределенная инфраструктура, которая построена на основе множества разнородных сетевых ресурсов (кластера, сервера, отдельные ПК) и используемая для решения научных задач на больших вычислительных мощностях. GRID является скоординированной, стандартизованной и открытой средой, что позволяет осуществлять оптимальное распределение вычислительных ресурсов для запуска поступающих на вход системы заданий.

GRID-системы подразделяются на три типа: добровольные, научные и коммерческие. Каждый из представленных типов обладает рядом достоинств и недостатков, характеризуется большим количеством решений, реализаций, форм организации вычислений. Одной из актуальных задач является разработка прикладного программного обеспечения, главной функцией которого является предоставление пользователю GRID-систем удобного интерфейса между приложением и необходимыми вычислительными ресурсами.

2. Анализ литературных данных и постановка проблемы

Известно множество методов и алгоритмов распределения заданий по вычислительным ресурсам. Огромная ниша в данном вопросе отводится программам, которые осуществляют распределение заданий по ресурсам – планировщикам заданий (брокерам). Основная задача брокера – построение плана распределения, удовлетворяющего требованиям поставщиков заданий. При построении плана распределения осуществляется оптимизация значений параметров целевой функции, за счет которых происходит сокращение времени выполнения поступивших в систему

заданий, что приводит к эффективному использованию вычислительных ресурсов.

Зачастую при распределении заданий выбирается первый подходящий ресурс без какой-либо оптимизации [1–4]. Другие алгоритмы не учитывают особенности среды с неотчуждаемыми ресурсами, т. е. не происходит мониторинг динамики загрузки вычислительных ресурсов, не учитывается конкуренция поставщиков заданий [5, 6]. В [6] приведены критерии для выбора слотов на основе функций полезности, которые задает поставщик заданий. Однако даже здесь оптимизация выполняется на уровне выбора наилучшего из доступных вычислительных ресурсов. Множество алгоритмов не ориентировано на предварительное резервирование, а основывается на очередности заданий в очереди [7, 8]. Однако использование предварительного резервирования [1–3, 9, 10] дает выигрыш по времени выполнения пула заданий, что, в свою очередь, позволяет сократить время простоя вычислительных ресурсов в системе.

Множество подходов к распределению заданий по вычислительным ресурсам [10, 11] ориентируются на использование моделей целочисленного линейного или смешанного программирования. В данном случае поставщики заданий могут задавать предпочтительные временные рамки использования вычислительных ресурсов. В [12] предлагается использовать сочетание модели целочисленного линейного программирования с генетическим алгоритмом, что позволяет составлять план распределения и учитывать стоимость использования вычислительных ресурсов для отдельных групп поставщиков. В [13] предлагается модель распределения заданий, которая позволяет осуществлять перепланировку «на лету» за счет информации о фактической динамике загруженности системы.

Методы планирования, описанные в [4, 9–13], используют ряд критериев (стоимость использования ресурсов, уровень загрузки вычислительных ресурсов, использование близлежащих ресурсов для связанных задач в задании). В исследованиях [14–17] предлагаются использовать методы планирования, которые делают упор на предпочтениях поставщиков заданий, администраторов виртуальных организаций или поставщиков вычислительных ресурсов.

В формат ресурсного запроса [17] вводится критерий оптимизации, который учитывает интересы пользователей. Данный критерий используется в ходе поиска альтернатив для выполнения заданий, т. е. владельцы вычислительных ресурсов получают возможность управлять вычислительной загруженностью системы. Также здесь реализована политика разбиения задания на отдельные подзадания, которые могут быть запущены на различных вычислительных ресурсах, что позволяет повысить эффективность загруженности системы.

В [18] выбор вычислительных ресурсов для поступающих в систему заданий осуществляется с помощью логико-вероятностного алгоритма. Предложенный алгоритм ориентирован на многоуровневое планирование заданий по заданным критериям качества (стоимость, время выполнения, показатели надежности). Планирование осуществляется в четыре шага на основе механизма регулирования спроса и предложения ресурсов.

В [19] предложена классификация поступающих заданий, которая может быть использована как надстройка для планировщика заданий (брокера). Использование предложенной надстройки повышает эффективность использования вычислительных ресурсов за счет оптимизации распределения вычислительных ресурсов.

Также присутствуют методы [14, 15], в которых имеется возможность введения заданного критерия отбора (например, с помощью языка JSDL), однако они характеризуются линейной временной сложностью, т. к. зависят от числа вычислительных ресурсов, которые доступны на данный момент планирования.

Проведенный анализ показывает, что недостатком существующих GRID-систем является использование единого брокера, который ориентирован на определенный класс задач и при распределении поступивших заданий использует одну политику распределения. Однако GRID – в первую очередь гетерогенная система [20] и, кроме того, задания, поступающие на выполнение, могут носить неоднородный характер. Большинство современных планировщиков не принимают во внимание цену использования вычислительного ресурса, что также приводит к неэффективному использованию ресурсов.

3. Цель и задачи исследования

Целью данной работы является разработка информационной технологии для распределения заданий на вычислительные ресурсы в гетерогенной GRID-системе.

Для достижения поставленной цели необходимо решить следующие задачи:

– разработать математическую модель представления заданий и вычислительных ресурсов в GRID-системах;

– разработать метод выбора наилучшего варианта распределения;

– провести ряд экспериментов по распределению заданий на основе предложенного метода в среде имитационного моделирования GRASS.

4. Использование имитационного моделирования для распределения заданий в GRID-системах

Обоснуем целесообразность использования предложенной среды моделирования GRASS (GRID Advanced Simulation System) для исследований применения разных стратегий планирования заданий.

Существует множество сред моделирования GRID-систем. Наиболее популярными из них являются SimGrid, GridSim, MicroGrid, ChicSim, OptorSim, Alea [21, 22]. Проведенный сравнительный анализ показывает, что большинству из них присуща узкая специализация, ограниченность архитектур, отсутствие доступных и открытых версий. Также для работы с такими системами необходимы знания специализированных языков программирования.

Для создания GRID-системы необходимы вложения средств, которые не всегда доступны, поэтому более дешевым вариантом является использование

имитационного моделирования. Имитационное моделирование основывается на построении математической модели, с помощью которой можно исследовать свойства реальной системы.

Система GRASS [23] является компьютерной моделью GRID-системы, которая позволяет разработать структуру распределенных вычислительных систем с учетом решаемых ею задач и исследовать ее свойства. Данная модель позволяет:

– воспроизвести процесс функционирования во времени элементарных событий, протекающих в системе с сохранением логики их взаимодействия;

– произвести ряд вычислительных экспериментов, которые позволят осуществить сбор, анализ результатов моделирования, а также сопоставление полученных данных с реальным поведением исследуемого объекта.

В GRID-системах задания запускаются не по мере их поступления, а в определенное время суток. Следовательно, можно воспользоваться пулом заданий и ресурсов из реальной GRID-системы, и с помощью среды GRASS промоделировать и получить наилучший план распределения, который можно предложить реальной системе. Полученный наилучший план минимизирует не только конечное время выполнения пула заданий, но и позволяет повысить эффективность использования вычислительных ресурсов GRID-системы. Однако в данном случае возникает проблема, связанная с тем, что среда GRASS имитирует работу реальной GRID-системы и результат ее работы может быть получен только через определенное время, когда план распределения может быть уже не актуален. Данная проблема может быть решена двумя способами:

– запуск среды GRASS осуществляется параллельно на нескольких процессорах (по количеству присутствующих в среде политик распределения);

– введение временного масштабирования для процесса моделирования.

Первое решение позволяет сократить время моделирования, так как первый полученный план – это наилучшее решение по времени для заданного пула заданий. Однако данное решение является долгосрочным вариантом, что не всегда возможно применить на практике. Второе решение, основанное на введении временного коэффициента, позволяет уменьшить время распределения (переход от часов к секундам). После получения планов по всем методам распределения необходимо перевести секунды в часы и получить реальное время моделирования, на основе которого провести анализ и выбрать наилучший план распределения.

5. Технологии распределения заданий с использованием среды GRASS

GRID-система является сложным объектом, содержащим множество взаимосвязанных разнородных ресурсов, функционирование которых подчиняется ряду заданных правил. Одной из главных задач такой системы является согласованное распределение ресурсов для решения предоставляемых задач. Модель распределения заданий в GRID-системе может быть построена на основе двух множеств: множестве вычислительных ресурсов R и множестве заданий Z, а также политике распределения q, т. е. $G = \{R, Z, q\}$.

Задания, поступая в GRID-систему, образуют поток $\{Z_i, i=1, 2,...,M\}$, где i – порядковый номер задания, а M – количество заданий. Каждое задание содержит ряд параметров, необходимых для их запуска в данной среде (1):

$$Z_i = \left\{ar_i^z, os_i^z, pc_i^z, ps_i^z, ms_i^z, dc_i^z, pr_i^z\right\}, \ \forall i = 1...M, \qquad (1)$$

где ar_i (architecture) – архитектура процессора; os_i (operating system) – операционная система; pc_i (processor count) – количество процессоров; ps_i (processor speed) – быстродействие процессоров; ms_i (memory size) – объем оперативной памяти; dc_i (disk capacity) – доступный объем винчестера; pr_i (priority) – приоритет задания.

Любое задание представляет собой пакет задач $\left\{z_a, a = 1,2,...,A\right\}$ $\left(z_a \in Z_i\right)$, объединенных определенной тематикой. Каждая задача – это отдельная исполняемая программа. Задание может быть запущено на выполнение только в том случае, когда для всех задач будут подобраны запрашиваемые ресурсы.

Ресурсы также образуют множество

$$\left\{R_j, j = 1,2,...,N\right\},$$

где j – номер вычислительного ресурса, а N – число ресурсов в GRID-системе. Любой вычислительный ресурс, поступающий в GRID-систему, описывается рядом характеристик, которые представлены кортежем (2):

$$R_j = \left\{ar_j^r, os_j^r, pc_j^r, ps_j^r, ms_j^r, dc_j^r\right\}, \ \forall j = 1...N, \qquad (2)$$

В различных системах распределения входные кортежи (1) и (2) могут иметь несущественные различия – это связано с тем, что системы развиваются и в ходе работы с ними возникают необходимость введения дополнений, связанных с требованиями поставщиков заданий.

Важное место в GRID-системах отводится планировщикам заданий (брокерам), в обязанности которых ставится создание расписания использования вычислительных ресурсов. В настоящее время насчитывается более двадцати известных планировщиков для GRID-систем: Portable Batch System (PBS), Sun Grid Engine (SGE), TORQUE, Condor, LoadLever, MAUI Scheduler и др. Перечисленные выше планировщики не предоставляют пользователю единого эффективного способа распределения заданий. С их помощью возможно лишь воспользоваться одним простым алгоритмом, который запускается при условии, что все задания и ресурсы системы приведены к заданному заранее общему описанию.

В настоящее время не существует универсального планировщика заданий, т. к. задания в GRID-системах разнородные. Следовательно, при распределении необходимо учитывать дополнительные параметры, которые позволят повысить эффективность использования вычислительных ресурсов системы. Например, анализ задач в задании с точки зрения их связности позволит сократить время выполнения пула заданий в системе за счет устранения потерь по времени, которые вызваны обменом данных между отдельными задачами в задании.

Расширим модель GRID-системы за счет введения в кортежи 1 и 2 параметров, которые уменьшат время простоя вычислительных ресурсов в GRID-системе (3), (4):

$$Z_i = \left\{ar_i^z, os_i^z, pc_i^z, ps_i^z, ms_i^z, dc_i^z, pr_i^z, ca_i^z, rt_i^z\right\},$$

$$\forall i = 1...M, \qquad (3)$$

$$R_j = \left\{ar_j^r, os_j^r, pc_j^r, ps_j^r, ms_j^r, dc_j^r, bw_j^r, d_j^r\right\}, \quad \forall j = 1...N, \qquad (4)$$

где ca_i (coefficient of association) – коэффициент связности задач в задании; rt_i (run time) – время выполнения задания (время использования ресурса); bw_j (bandwidth) – суммарная пропускная способность канала (от брокера до ресурса) с учетом состояния сети на текущий момент времени; d (delay) – суммарная задержка времени передачи пакета с учетом состояния сети на текущий момент времени.

Также введем в модель множество методов распределения вместо единой политики распределения (5):

$$Q_k = \left\{mn_k, lp_k\right\}, \ \forall k = 1...k, \qquad (5)$$

где mn – алгоритм распределения заявок по вычислительным ресурсам; lp – перечень входных параметров, учитываемых при распределении.

Как уже упоминалось выше, недостатком GRID-системы является использование единого брокера, который ориентирован на определенный класс задач. Предложенная среда GRASS позволяет оперировать на данный момент несколькими алгоритмами распределения: First-Come First-Served (FCFS), Last In First Out (LIFO), Highest Priority First (HPF), метод линейного программирования (Simplex), метод распределения по освободившемуся ресурсу (Smart), метод обратного заполнения (Backfill), а также метод распределения заданий на вычислительные ресурсы с учетом сетевого трафика (Backfill mod). Backfill mod – это модернизация метода обратного заполнения, который, в отличии от существующего, позволяет производить распределения заданий, поступающих в GRID-систему в зависимости от связанности задач в каждом из заданий.

Для выбора эффективного плана распределения был разработан метод выбора наилучшего плана распределения, который был внедрен в среду моделирования GRASS. Для этого необходимо:

– сформировать пулы заданий и ресурсов для различных классов заданий (либо сформировать в среде GRASS используя заданные генераторы, либо использовать пулы заданий из реальных GRID-систем);

– выбрать политику (метод) распределения;

– запустить моделирование в среде GRASS;

– проанализировать log-файлы для всех политик (методов) распределения;

– выбрать наилучший план распределения на основании принятых правил и ограничений.

Предложенные политики распределения заданий в среде GRASS для различных входных пулов заданий и ресурсов могут давать выигрыш по времени и сокращать простой вычислительных ресурсов, поэтому при анализе необходимо задавать жесткие ограничения для выборки.

Рассмотрим работу информационной технологии распределения заданий [24] в гетерогенной GRID-системе, которая использует имитационную среду моделирования GRASS (рис. 1). Предложенная технология включает в себя следующие этапы.

Этап 1. Задание параметров эксперимента. Для запуска эксперимента следует осуществить настройку конфигурационного файла plugins.xml, в котором описываются взаимосвязи между именами модулей в системе и названиями файлов и библиотек, а также пути к конфигурационным файлам [25]. Файл plugins.xml позволяет задавать параметры, передаваемые модулям при загрузке, которые могут быть использованы для задания режима работы или инициализации внутренних значений: для заданий, вычислительных ресурсов и методов распределения.

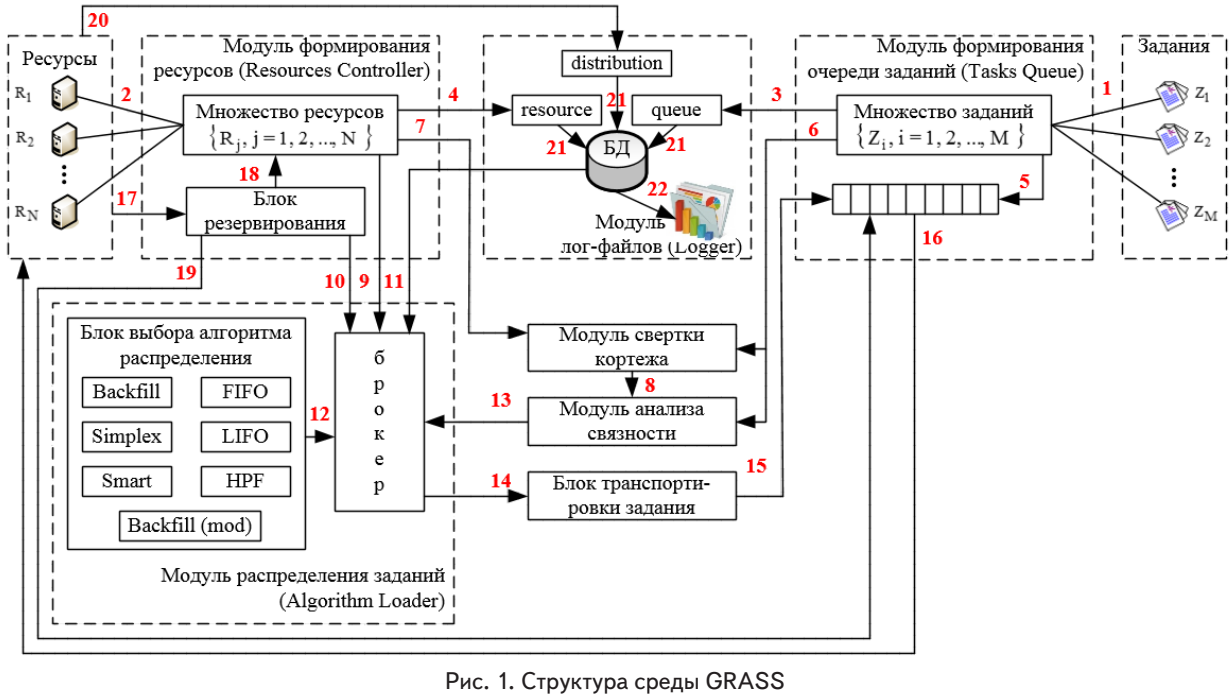

Рис. 1. Структура среды GRASS

Результатом работы данного этапа является считывание и декодирование конфигурационного файла (plugins.xml), загрузка основных плагинов системы для запуска процесса распределения.

Этап 2. Загрузка в среду моделирования GRASS информации о вычислительных ресурсах и заданиях.

Задание, поступающее в систему, – это пакет задач, который представляет собой отдельную исполняемую программу. Процессор в каждый момент времени может выполнять только одну задачу и задание может быть запущено на выполнение только в том случае, если для всех задач задания подобраны вычислительные ресурсы. Любое задание, поступающее в среду GRASS, можно разбить на две составляющие: характеристики (параметры) задания и задание (в виде .exe файла, файлы входных данных, БД и т. д.).

Задания, поступающие в среду GRASS, образуют поток (связь 1, рис. 1) $\{Z_i, i = 1, 2, ..., M\}$, где i – номер задания, а M – количество заданий. Поступив в систему, происходит разделение задания по предложенным выше составляющим. На начальном этапе для системы главное значение имеет информация о задании (3), которая оперирует данными необходимыми для поиска вычислительных ресурсов, пригодных для запуска [26]. Файл (tasks_data.xml), содержащий данную информацию, приведен к заданному заранее формату и позволяет осуществить подбор конкретного ресурса для запуска в данной системе (рис. 2).

```
<tasks>
    <task time="1000">
            <parameter cpu_arch="AMD"/>
            <parameter cpu_count="64"/>
            <parameter cpu_speed="1000"/>
            <parameter os="Windows XP"/>
            <parameter physical_memory="16"/>
            <parameter priority ="18"/>
            <parameter timeout="180000"/>
            <parameter hdd_size="16" />
            </task>
</tasks>
```

Рис. 2. Фрагмент файла tasks_data.xml

Среда GRASS позволяет не только подгружать сформированный заранее файл заданий с помощью плагина SimpleTasksManager, но и в ходе процесса распределения загрузить генератор формирования заданий, используя плагин SimpleTasksGenerator из файла tasks_model.xml (рис. 3).

В настоящий момент в среде GRASS реализованы следующие генераторы:

– для целых чисел: равномерный (Uniform), константный (Constant), экспоненциальный (Exponent) и нормальный (Normal);

– для списков: последовательный выбор по одному значению из предложенного заранее списка (ListLinear) и случайный выбор (ListRandom).

Одновременно с заданиями в систему поступает информация о доступных ресурсах (связь 2, рис. 1) $\{R_j, j = 1, 2, ..., N\}$, где j – порядковый номер ресурса, а N – число ресурсов в GRID-системе. Ресурсы, как и задания, также приведены к заданному формату (resources_data.xml) (рис. 4).

```
<delay generator="Uniform">
<parameter a="1500"/>
<parameter b="2300"/>
</delay>
<memory generator="Normal">
<double mean="1000000"/>
<double sigma="5000"/>
</memory>
<priority generator="Constant">
<double value="100"/>
</priority>
<time generator="Exponential">
<double gamma="0.5"/>
</time>
<OS generator="ListLinear">
<parameter values="stringlist">
<string value="Linux 2.6.2"/>
<string value="Linux 2.4.2"/>
<string value="Windows 98"/>
<string value="Windows 2000"/>
<string value="Windows XP"/>
<string value="Windows Vista"/>
<string value="Windows 7"/>
<string value="Solaris"/>
<string value="Mac Os X"/>
</parameter>
</OS>
```

Рис. 3. Фрагмент файла tasks_model.xml для генерирования пула заданий

```
<resources>
    <resource name="name1" time="480">
            <parameter cpu_arch="AMD"/>
            <parameter cpu_count="8"/>
            <parameter cpu_speed="1000"/>
            <parameter os="Windows XP"/>
            <parameter physical_memory="16"/>
            <parameter hdd_size="64"/>
            <parameter free_cpu="8"/>
    </resource>
```

Рис. 4. Фрагмент файла resources_data.xml

Так же, как и при формировании заданий, можно воспользоваться плагинами SimpleResourcesManager и SimpleResourcesGenerator. Первый плагин используется для загрузки ресурсов из файла resources_data.xml, который был сформирован заранее, второй плагин загрузит выбранный генератор ресурсов, который указан в файле конфигурации resources_model.xml. Генераторы формирования вычислительных ресурсов аналогичны генераторам заданий (рис. 3). Параллельно работе данного этапа происходит запуск 7-го этапа, который отвечает за ведение log-файлов в среде GRASS.

Этап 3. Выбор политики распределения. Процесс распределения заданий осуществляется после проверки параметра tasks_count плагина algorythmLoader. Как только количество заданий в очереди превысит число, указанное в параметре tasks_count, происходит вызов алгоритма распределения (5), который определен в параметре DistributionAlgorythm. Согласно выбранному методу распределения [26] произойдет под-

бор ресурсов для каждого из заданий очереди. Модуль распределения заданий, который является составной частью среды GRASS, содержит ряд методов, которые имитируют работу различных алгоритмов распределения, каждый из которых использует свой набор параметров (3) и (4) для распределения. Методы FCFS и LIFO не используют дополнительные параметры, т. к. это простейшие политики, служащие для наглядного сравнения с другими методами.

Этап 4. Формирование дополнительных параметров для наиболее эффективного распределения. Все поступившие в систему задания помещаются в очередь заданий (связь 5, рис. 1), и параллельно происходит передача информации о каждом задании (в виде кортежа 3) в модуль свертки кортежа и модуль анализа связности (связь 6, рис. 1). Модуль свертки кортежа осуществляет вычисление обобщенного критерия оценки для каждого задания, который позволит более продуктивно управлять процессом распределения заданий на вычислительные ресурсы и покажет, какую часть ресурса занимает задание в процессе выполнения [27]. Одновременно на данный модуль происходит передача информации о ресурсах, имеющихся в системе (связь 7, рис. 1). Результатом работы модуля свертки кортежа является перечень ресурсов, на которых каждое задание может быть распределено.

Далее информация поступает на модуль анализа связанности (связь 8, рис. 1), где происходит анализ задач в задании. Если задачи в задании имеют высокую связанность (необходим обмен информацией между задачами в ходе их выполнения) или для данного задания необходима пересылка большого объема входных и выходных данных, произойдет вызов метода checkQueueStrict(), который сравнит требования задания с имеющимися вычислительными возможностями.

Если в системе присутствуют вычислительные ресурсы для запуска данного задания, то оно останется в очереди и получит состояние Waiting и для него в дальнейшем будут подобраны вычислительные ресурсы. Если таких ресурсов не окажется, то заданию присвоится статус Cancelled, после чего оно покинет очередь, т. к. данный статус говорит о том, что задание не может быть запущено на данной конфигурации.

В данном случае при распределении задач из задания акцент будет сделан на подбор вычислительных ресурсов таким образом, чтобы уменьшить время на пересылку данных для задач задания.

Если задачи в задании не связаны между собой, то в системе вызовется метод checkQueueBase(), который аналогично методу checkQueueStrict() осуществит подбор вычислительных ресурсов с возможностью распределения отдельных задач из задания различным вычислительным ресурсам и задание останется в очереди в состоянии Waiting до момента запуска его на вычислительном ресурсе.

Этап 5. В модуль распределения заданий (брокер), поступает информация, позволяющая принять решения для распределения задания на вычислительный ресурс или ресурсы, выполнение на которых будет для него наиболее оптимальным, как по аппаратным показателям, так и по характеристикам производительности. Для распределения брокеру необходимо получить следующую информацию:

– информацию о ресурсах (4), которые на данный момент присутствуют в системе (связь 9, рис. 1);

– информацию о ресурсах, которые на данный момент зарезервированы (связь 10, рис. 1) и время их освобождения;

– информацию из БД о предыдущих запусках задания (связь 11, рис. 1);

– политику (метод) распределения (связь 12, рис. 1);

– информация о связанности задач в задании (связь 13, рис. 1).

На основе полученных данных осуществляется распределение. Результатом работы данного модуля является план (связь 14, рис. 1), который каждому заданию определяет вычислительный ресурс, подходящий по указанным ранее требованиям. Любое задание, поступившее в среду GRASS, будет распределено, исключением может быть только случай, когда в системе не окажется требуемых вычислительных ресурсов.

Этап 6. Процесс пересылки задания на ресурс или ресурсы, указанные в плане распределения. Для этого из блока, отвечающего за транспортировку, осуществляется запрос в очередь заданий (связь 15, рис. 1), который в соответствии с планом распределения выбирает конкретное задание и присваивает ему состояние Running. Далее задание (вторая составляющая) отправляется на выбранный ресурс для последующего запуска (связь 16, рис. 1). Т. к. произошли действия с ресурсами (в данном случае: выбран ресурс для выполнения задания), то необходимо оповестить систему об этом (связь 17, рис. 1). Блок резервирования отсылает информацию системе, о том, что ресурс не может быть использован для последующего запуска до момента окончания выполнения задания (связь 18, рис. 1). Когда снова произойдут действия с ресурсами (например, произойдет освобождение ресурса), блок резервирования получит информацию о произошедшем событии (связь 17, рис. 1) и оповестит систему о возможности использовании ресурса для распределения (связь 18, рис. 1), а также отправит сообщение в очередь заданий для удаления задания из очереди в связи с окончанием его выполнения (связь 19, рис. 1).

Этап 7. Сбор статистической информации о распределении. Для получения статистической информации о распределении и анализа ее в дальнейшем, происходит запуск модуля, который отвечает за log-файлы. В среде GRASS за сбор статистики отвечает модуль ведения лог-файлов (Logger). Данный модуль предоставляет возможность гибкого и централизованного ведения логов для всех плагинов среды моделирования GRASS.

В системе имеется несколько типов log-файлов, которые фиксируют ряд действий, что позволяет при необходимости быстро получить требуемую информацию из БД.

При поступлении заданий в систему происходит автоматическая запись их в файл queue.log (связь 3, рис. 1), а при поступлении ресурсов (связь 4, рис. 1) в файл resources.log. В момент запуска задания на ресурсе или окончания выполнения задания происходит фиксация информации о данном действии в файле raspred.log (связь 20, рис. 1), т. е. происходит дозапись данного файла. Все записи из log-файлов пересылают-

ся в БД системы (связь 21, рис. 1), данные из которой могут быть использованы в дальнейшем для осуществления выборки по заданным параметрам или для графического отображения информации за определенный промежуток времени (связь 22, рис. 1).

Процесс моделирования продолжается до тех пор, пока в очереди присутствуют задания.

6. Особенности системы имитационного моделирования GRASS

Среда GRASS – это проект с открытым исходным кодом, который реализует модульную среду моделирования GRID-систем [23], является кроссплатформенным, реализован на языке C++ с использованием кроссплатформенных библиотек Qt4, Boost.

Среда моделирования GRASS имеет как консольный, так и графический интерфейс пользователя, который позволяет наблюдать за процессом моделирования в режиме реального времени. Графический интерфейс предоставляет пользователю больше возможностей, т. к. отображает и визуализирует статистику работы среды моделирования, состояние очереди заданий и ресурсов, а также наглядно демонстрирует выполнение заданий вычислительными ресурсами. Помимо интерактивного наблюдения за процессом моделирования, среда GRASS позволяет составлять отчет для последующего более детального его анализа. Отчет реализован с использованием скриптов, что позволяет легко менять как его внешний вид, так и содержание.

Среда моделирования GRASS имеет модульную архитектуру: состоит из ядра и динамически подключаемых модулей (плагинов). Каждый модуль выполняет свою узкоспециализированную задачу, обращаясь при необходимости к другим модулям системы. Ядро предоставляет средства межмодульного взаимодействия, а также обеспечивает начальную загрузку и конфигурацию системы. Каждый модуль имеет уникальный строковый идентификатор (имя или ID), который предоставляет набор интерфейсов взаимодействия с ним для других компонентов системы. Каждый интерфейс модуля также имеет имя и может быть реализован любым количеством модулей. Таким образом, для получения какого-либо интерфейса модуля необходимо знать только его имя и имя интерфейса взаимодействия.

Модуль в среде GRASS представляет собой динамически подключаемую библиотеку dll (dynamic linked library), которая реализует фабричный метод, создающий экземпляр класса модуля. Класс модуля реализует интерфейс Framework::IPlugin, что позволяет универсально работать с ним, не учитывая особенности реализации.

Для создания гибкой модульной системы необходима возможность простого изменения набора плагинов и их параметров без модификации ядра или самих библиотек модулей. Для этого в среде GRASS используется система конфигурационных файлов на основе XML.

Разработанная программная система является базой для продолжения исследований в области моделирования GRID-систем.

7. Обсуждение результатов исследования применения предложенной информационной технологии

В ходе исследования информационной технологии распределения заданий были проведены ряд экспериментов. Их целью было показать рациональность использования не единого брокера для всех поступающих заданий на вход системы, а выбирать наилучший, ориентируясь на классы заданий.

С помощью среды GRASS были сформированы 2 пула: заданий и ресурсов. Пул заданий включает в себя 300 заданий, каждое из которых описано кортежем (3), пул ресурсов сформирован в соответствии с кортежем (4) и включает в себя 80 вычислительных ресурсов. Каждое задание характеризуется требованиями, выставляемыми поставщиками заданий.

Используя различные политики распределения, было проведено моделирование. Результаты моделирования сравниваются между собой, и на основе анализа принимается решение о наилучшем распределении для конкретного пула заданий. На рис. 5 показано время работы всех методов среды GRASS для конкретного пула заданий, а на рис. 6 – процент простоя вычислительных ресурсов.

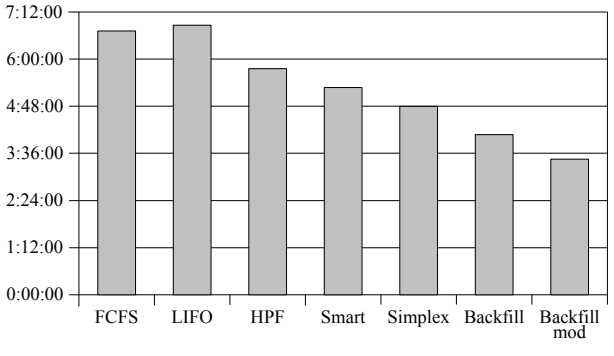

Рис. 5. Время распределения пула заданий в среде GRASS

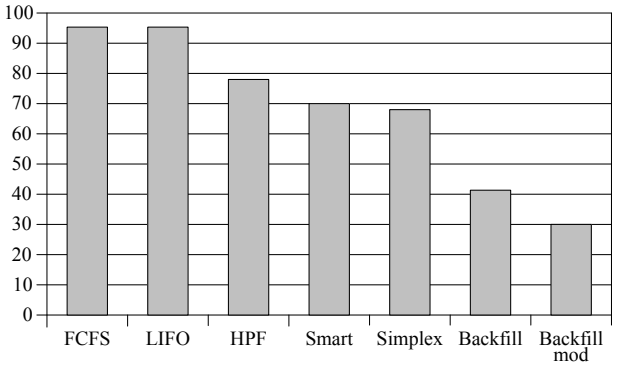

Рис. 6. Процент простоя вычислительных ресурсов в среде GRASS

Все задачи, поступающие в GRID-систему, можно условно разделить на 4 класса, которые характеризуются:

– большим объем входных данных и большим объемом выходных данных;

– малым объем входных данных и большим объемом выходных данных;

– большим объем входных данных и малым объемом выходных данных;

– малым объем входных данных и малым объемом выходных данных.

В табл. 1 и на рис. 7 представлены результаты моделирования работы всех методов в среде GRASS в соответствии с приведенной классификацией заданий.

Таблица 1

Время работы методов среды GRASS для разных классов задач

Методы распределения	Классы задач			
	1-й	2-й	3-й	4-й
FCFS	6:50:21	5:16:28	5:23:53	2:32:42
LIFO	6:54:43	5:58:50	6:01:52	2:37:53
HPF	6:33:45	6:00:35	6:08:32	2:40:22
Smart	6:28:55	6:03:51	6:10:44	3:09:49
Simplex	6:33:46	5:33:46	4:56:21	2:57:55
Backfill	5:19:54	5:01:49	3:28:43	3:42:46
Backfill (mod)	4:48:39	3:58:23	2:47:33	3:48:01

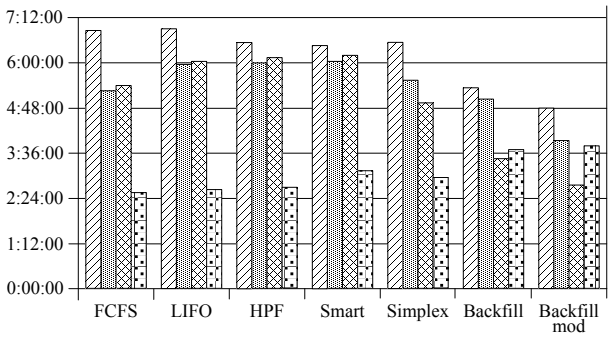

Рис. 7. Время работы методов среды GRASS в зависимости от класса заданий

В процессе проведения экспериментов была выявлена зависимость результатов распределения от класса задач. На данный момент не существует единого метода распределения, который на любом пуле заданий позволял бы получать наилучший план. Но в случае, когда известны характеристики GRID-системы и входного потока заданий, можно провести моделирование и выбрать наилучший метод распределения для конкретного пула заданий.

Достоинством среды моделирования GRASS является то, что она оперирует алгоритмом, который анализируя входной поток заданий, выбирает метод распределения, который дает наилучшее распределение по заданным требованиям поставщика задания.

8. Выводы

В результате проведенных исследований были проанализированы методы распределения заданий в гетерогенных GRID-системах. Из проведенного анализа можно сделать вывод, что существующие на рынке планировщики обладают рядом недостатков, главный из которых – это ориентация на конкретный класс задач. Это можно объяснить тем, что данные планировщики предназначались изначально для кластерных систем, которые в свою очередь позже включались в GRID-инфраструктуру, а задания продолжали выполняться локально.

В процессе исследований была предложена модификация математических моделей представления заданий и ресурсов в GRID-системе. Использование в модели представления заданий коэффициента связности (этап 4) позволяет осуществлять подбор вычислительных ресурсов с учетом минимизации времени на обмен данными между задачами. Введение в модель представления вычислительных ресурсов величины, учитывающей объем трафика и задержки передачи информации по каналу, позволяют сократить время выполнения пула заданий, что повысит эффективность использования вычислительных ресурсов в GRID-системе.

Основываясь на математической модели представления ресурсов и заданий, разработан метод выбора наилучшего варианта распределения на основании анализа данных, полученных в ходе эксперимента. Предложенная среда GRASS позволяет оперировать несколькими алгоритмами распределения: FCFS, LIFO, HPF, Simplex, Smart, Backfill, Backfill mod, что позволяет промоделировать распределение для конкретного пула заданий на различных политиках распределения и в дальнейшем проанализировать результаты моделирования. В связи с тем, что предложенная система является модульной, нет никаких препятствий для реализации и подключения новых алгоритмов распределения.

В ходе исследования был проведен ряд экспериментов по распределению заданий на вычислительные ресурсы в среде имитационного моделирования GRASS для различных политик распределения. Результаты, полученные в ходе экспериментов, свидетельствуют о сокращении времени выполнения пула заданий до 24 % и повышении эффективности использования вычислительных ресурсов до 32 % для ряда политик распределения (методов), реализованных в среде GRASS.

Литература

1. Aida, K. Scheduling Mixed-parallel Applications with Advance Reservations [Text] / K. Aida, H. Casanova // Proceedings of the 17th international symposium on High performance distributed computing - HPDC '08, 2008. – P. 65–74. doi: 10.1145/1383422.1383432
2. Ando, S. Evaluation of Scheduling Algorithms for Advance Reservations [Text] / S. Ando, K. Aida // Information Processing Society of Japan SIG Notes. HPC-113, 2007. – P. 37–42.
3. Elmroth, E. A Standards-based Grid Resource Brokering Service Supporting Advance Reservations, Coallocation and Cross-Grid Interoperability [Text] / E. Elmroth, J. Tordsson // Concurrency and Computation: Practice and Experience. – 2009. – Vol. 21, Issue 18. – P. 2298–2335. doi: 10.1002/cpe.1441
4. Cafaro, M. Preference-Based Matchmaking of Grid Resources with CP-Nets [Text] / M. Cafaro, M. Mirto, G. Aloisio // Journal of Grid Computing. – 2013. – Vol. 11, Issue 2. – P. 211–237. doi: 10.1007/s10723-012-9235-2

5. Kurowski, K. Multicriteria Aspects of Grid Resource Management [Text] / K. Kurowski, J. Nabrzyski, A. Oleksiak, J. Weglarz // International Series in Operations Research & Management Science, 2003. – P. 271–293. doi: 10.1007/978-1-4615-0509-9_18
6. Ernemann, C. Economic Scheduling in Grid Computing [Text] / C. Ernemann, V. Hamscher, R. Yahyapour; D. G. Feitelson, L. Rudolph, U. Schwiegelshohn (Eds.) // Lecture Notes in Computer Science. – 2002. – Vol. 2537. – P. 128–152. doi: 10.1007/3-540-36180-4_8
7. Rodero, I. Enabling Interoperability among Grid Meta-Schedulers [Text] / I. Rodero, D. Villegas, N. Bobroff, Y. Liu, L. Fong, S. Sadjadi // Journal of Grid Computing. – 2013. – Vol. 11, Issue 2. – P. 311–336. doi: 10.1007/s10723-013-9252-9
8. Azzedin, F. A Synchronous Co-allocation Mechanism for Grid Computing Systems [Text] / F. Azzedin, M. Maheswaran, N. Arnason // Cluster Computing. – 2004. – Vol. 7, Issue 1. – P. 39–49. doi: 10.1023/b:clus.0000003942.73875.29
9. Castillo, C. Resource Co-allocation for Large-scale Distributed Environments [Text] / C. Castillo, G. N. Rouskas, K. Harfoush // 18th ACM International Symposium on High Performance Distributed Compuing, ACM, 2009. – P. 137–150.
10. Takefusa, A. An Advance Reservation-based Co-allocation Algorithm for Distributed Computers and Network Bandwidth on QoS-guaranteed Grids [Text] / A. Takefusa, H. Nakada, T. Kudoh, Y. Tanaka; E. Frachtenberg, U. Schwiegelshohn (Eds.) // Lecture Notes in Computer Science. – 2010. – Vol. 6253. – P. 16–34. doi: 10.1007/978-3-642-16505-4_2
11. Blanco, H. MIP Model Scheduling for Multiclusters [Text] / H. Blanco, F. Guirado, J. L. L rida, V. M. Albornoz // Lecture Notes in Computer Science. – 2012. – Vol. 7640. – P. 196–206. doi: 10.1007/978-3-642-36949-0_22
12. Garg, S. K. A Linear Programming-driven Genetic Algorithm for Meta-scheduling on Utility Grids [Text] / S. K. Garg, P. Konugurthi, R. Buyya // International Journal of Parallel, Emergent and Distributed Systems. – 2011. – Vol. 26, Issue 6. – P. 493–517. doi: 10.1080/17445760.2010.530002
13. Olteanu, A. A Dynamic Rescheduling Algorithm for Resource Management in Large Scale Dependable Distributed Systems [Text] / A. Olteanu, F. Pop, C. Dobre, V. Cristea // Computers and Mathematics with Applications. – 2012. – Vol. 63, Issue 9. – P. 1409–1423. doi: 10.1016/j.camwa.2012.02.066
14. Toporkov, V. Slot Selection Algorithms in Distributed Computing [Text] / V. Toporkov, A. Toporkova, A. Tselishchev, D. Yemelyanov // Journal of Supercomputing. – 2014. – Vol. 69, Issue 1. – P. 53–60. doi: 10.1007/s11227-014-1210-1
15. Toporkov, V. Slot Selection Algorithms in Distributed Computing with Non-dedicated and Heterogeneous Resources [Text] / V. Toporkov, A. Toporkova, A. Tselishchev, D. Yemelyanov; V. Malyshkin (Ed.) // Lecture Notes in Computer Science. – 2013. – Vol. 7979. – P. 120–134. doi: 10.1007/978-3-642-39958-9_10
16. Топорков, В. В. Методы и эвристики планирования в распределенных вычислениях с неотчуждаемыми ресурсами [Текст] / В. В. Топорков, А. В. Бобченков, Д. М. Емельянов, А. С. Целищев // Вестник ЮУрГУ. Серия «Вычислительная математика и информатика». – 2014. – Т. 3, № 2. – С. 43–62.
17. Топорков, В. В. Метапланирование вычислений в распределенных средах с неотчуждаемыми ресурсами [Текст]: тр. межд. конф. / В. В. Топорков, Д. М. Емельянов, А. С. Топоркова; под. ред. Е. Л. Глориозова // Информационные технологии в науке, образовании и управлении. IT + S&E`16 (Гурзуф, 22.05. – 01.06.2016), 2016. – С. 22–31.
18. Костромин, Р. О. Модели, методы и средства управления вычислениями в интегрированной кластерной системе [Текст] / Р. О. Костромин // Фундаментальные исследования. – 2015. – № 6. – С. 35–38.
19. Феоктистов, А. Г. Методология концептуализации и классификации потоков заданий масштабируемых приложений в разнородной распределенной вычислительной среде [Текст] / А. Г. Феоктистов // Системы управления, связи и безопасности. – 2015. – № 4. – С. 1–25.
20. Venugopal, S. A Grid Service Broker for Scheduling e-Science Applications on Global Data Grids [Text] / S. Venugopal, R. Buyya, L. Winton // Concurrency and Computation: Practice and Experience. – 2006. – Vol. 18, Issue 6. – P. 685–699. doi: 10.1002/cpe.974
21. Астриков, Д. Ю. Моделирование системы планирования распределенного высокопроизводительного вычислительного комплекса [Текст] / Д. Ю. Астриков, Д. А. Кузьмин, А. И. Панасюк // Доклады Академии наук Высшей школы Российской федерации. – 2014. – № 2-3 (23-24). – С. 34–41.
22. Милюков, В. В. Моделирование фрагментов GRID-системы в симуляторе GridSim [Текст] / В. В. Милюков, Ю. В. Сосновский // Оптимізація виробничих процесів. – 2013. – № 14. – С. 218–222.
23. Волк, М. А. Архитектура имитационной модели grid-системы, основанная на подключаемых модулях [Текст] / М. А. Волк, А. С. Горенков, Р. Н. Гридель // Системи обробки інформації. – 2010. – Вип. 1. – С. 17–20.
24. Филимончук, Т. В. Информационная технология распределения заданий на вычислительные ресурсы в GRID-системах [Текст]: сб. науч. ст. / Т. В. Филимончук, В. Н. Ткачев // Информатика, математическое моделирование, экономика. Пятая Международная научно-практическая конференция, 2015. – С. 204–209.
25. Волк, М. А. Структура программного комплекса имитационного моделирования элементов GRID-систем для научных исследований [Текст] / М. А. Волк // Системи обробки інформації. – 2009. – Вип. 3. – С. 125–128.
26. Волк, М. А. Модуль распределения заданий в GRID-системах [Текст] / М. А. Волк, М. А. Филимончук, Т. В. Филимончук // Системи обробки інформації. – 2012. – Вип. 2. – С. 177–182.
27. Волк, М. А Обобщенный критерий оценки задания для технологии планирования заданий в GRID. В 3-х т. Т. 2 [Текст]: сб. науч. ст. / М. А. Волк, Т. В. Филимончук // Информатика, математическое моделирование, экономика. Третья Международная научно-практическая конференция, 2013. – С. 172–176.