Применение генетических алгоритмов для поиска оптимального расписания заданий в GRID

Т.С. Шаповалов

Предлагается алгоритм поиска оптимального расписания заданий, адаптированный для вычислительной GRID-сети и основанный на генетическом алгоритме. Приводится описание реализации алгоритма на базе GRID-инструментария Globus Toolkit.

1. Введение

Развитие высокопроизводительных параллельных вычислений подошло к этапу необходимости объединения разрозненных вычислительных ресурсов. Несмотря на непрекращающийся рост вычислительных мощностей современных компьютерных технологий, остаются задачи во многих областях науки, для которых имеющейся аппаратной производительности не хватает. GRID - технология, позволяющая динамически образовывать среду коллективных вычислений, расширяя возможности по использованию простаивающих ресурсов.

Планирование - ключевой момент в любой вычислительной среде, каким является и вычислительная GRID-сеть. С развитием GRID-технологий все отчетливее просматривается проблема эффективного сопоставления ресурсов и запускаемых задач - составления расписания заданий. В настоящий момент существуют несколько алгоритмов составления расписания в вычислительно-распределенных средах, но вопрос о планировании в GRID до сих пор остается открытым. В работе рассматривается применение метода поиска расписания заданий на основе генетических алгоритмов (ГА) [1]. Данный вид алгоритмов является одним из способов стохастической оптимизации, хорошо зарекомендовавших себя при решении сложных задач. ГА сочетают в себе элементы детерминированности и случайности и являют собой подход к задаче поиска лучшего решения, а не четко определенный алгоритм. Задачи представляют собой программы с приблизительно известными заранее параметрами, такими, как предполагаемое время расчета, пиковое потребление оперативной памяти, требуемые объемы пространств на жестких дисках, и другими, указываемыми пользователями на этапе постановки заданий на счет. Составив расписание на основе этих предварительных данных, мы можем получить по той или иной причине превышение времени расчета и тогда нам будет необходимо произвести корректировки в составлении дальнейших расписаний. В случае использования генетических алгоритмов, при добавлении задач нам не понадобиться выполнять поиск с самого начала, достаточно лишь воспользоваться особями последней полученной популяцией, попарно скрещивая их с особями новой популяции, которая уже учитывает недостающие в расписании задания. Так же необходимо отметить, что вычислительная сложность ГА составляет $O(n)$. Таким образом, для нахождения оптимального расписания было решено применить алгоритм поиска на основе ГА.

Рассматривая существующие решения для планирования в GRID, можно выделить алгоритмы, разработанные специально для GRID, например, алгоритм экономического планирования [4] или метод опережающего планирования [6], а так же алгоритмы планирования заданий в вычислительных кластерах и операционных системах, адаптированные позже под планирование в GRID, такие, как алгоритм равномерного планирования Round-Robin, алгоритм обратного заполнения Backfill [2], алгоритмы, основанные на приоритетах заданий, очередей и ресурсов (включая FIFO-алгоритмы), алгоритм Fair Share, алгоритм имитации отжига [3], алгоритм составления прогноза по данным статистического использования ресурсов [5].

В настоящий момент применение эволюционных методов для составления расписаний заданий исследуются как для многопроцессорных машин [7], в том числе и систем реального времени [8], так и для вычислительных кластеров [9]. В данной работе предпринята попытка

адаптировать генетический алгоритм для нахождения оптимального расписания в вычислительных GRID-сетях на базе инструментария Globus Toolkit (http://www.globus.org/toolkit).

2. Теоретическая часть

2.1. Генетический алгоритм

Генетический алгоритм (Genetic Algorithm) представляет собой вариант стохастического поиска, в котором особи-преемники формируются путем изменения или комбинирования двух и более родительских особей. Под особью в данном случае подразумевается расписание заданий в GRID. Работа генетического алгоритма начинается с генерирования множества сформированных случайным образом особей, называемых популяцией. Каждая особь представлена в виде строки символов и классифицируется с помощью целевой функции, или (в терминологии ГА) функции пригодности (fitness function). Функция пригодности должна возвращать более высокое (в случае поиска расписания - более низкое) значение для лучших особей-расписаний.

В условиях вычислительной GRID-сети необходимо составлять расписание на величину времени, соответствующую времени окончания расчета всех заданий. В реальных условиях такого события может не произойти никогда, так как к окончанию расчета начального множества задач могут поступать в очередь следующие задания.

Алгоритм не различает процессоры и ядра процессора - для него они представляют собой одну и ту же вычислительную единицу, на которой можно запустить вычислительный процесс. Далее под процессором подразумевается как процессор с одним ядром однопроцессорного вычислительного узла, так и одно из ядер многоядерного процессора.

Для генетического алгоритма необходимо, чтобы расписание было представлено в закодированном виде - в форме хромосом. Для наглядного описания этой формы рассмотрим простейшее расписание, в требования которого включим всего два параметра: количество процессов и предположительное максимальное процессорное время расчета каждого процесса. Задачи кратко описаны в таблице 1.

Таблица 1. Описание примерных заданий

Требуемые параметры	Задача A	Задача B	Задача C
Число процессов	8	2	1
Процессорное время (ч)	1	3	3

Была выбрана следующая форма представление расписания для алгоритма. Хромосома состоит из нескольких блоков, каждый из которых представляет собой описание выполнения задач на одном конкретном процессоре в формате пары последовательных ячеек <идентификатор задачи, процессорное время>. Если в расписании есть окна, в которых тому или иному процессору приходится простаивать, то такая задача простоя имеет идентификатор равный нулю. Генами в данном случае будут являться пары <идентификатор задачи, процессорное время>. Таким образом, наименьшей неделимой для генетических операторов единицей, будет пара числовых значений (далее - ген). Тип данных гена в нотации языка C++ будет std::pair<unsigned short, unsigned long>. Наглядно форму одной из возможных хромосом для задачи из таблицы 1 можно видеть на Рис. 1.

Для селекции особей (то есть выбора хромосом для участия в новом поколении) выбран вероятностный процесс, происходящий методом рулетки (roulette wheel selection). В этом виде селекции каждой из особей популяции ставится в соответствие сектор круга, размером, пропорциональным значению функции пригодности данной хромосомы и делается

Рис. 1. Формат хромосомы для примерной задачи

N выборок, где N - неизменный размер популяции. Вероятность включения той или иной хромосомы в следующую популяцию становится пропорциональной оценки ее пригодности. Математически процесс селекции описывается формулой 1.

$$p_s(h_i) = \frac{F(h_i)}{\sum_{j=1}^{N} F(h_j)}, \tag{1}$$

где $F(h_i)$ - значение функции пригодности хромосомы h_i, $p_s(h_i)$ - вероятность отбора хромосомы h_i в результате селекции. По окончанию процесса селекции создается популяция, называемая родительским пулом с численностью N.

В текущей реализации алгоритма находят применение два классических генетических оператора: кроссинговера (crossover operator) для обмена генетического материала между особями и мутации (mutation operator) для предотвращения скатывания алгоритма в локальный экстремум. Вероятности применения обеих операторов устанавливается в файле конфигурации алгоритма и не меняется в течении его работы.

Под схемой S будем понимать некоторую хромосому с зафиксированными в определенных позициях генами. Количество таких фиксированных ген назовем порядком схемы $o(S)$, а расстояние между первым и последним зафиксированными генами - длиной схемы $d(S)$.

В [1] показано, что ожидаемое количество хромосом, соответствующих схеме S в следующем поколении зависит от фактического количества хромосом, соответствующих схеме, относительной пригодности схемы, а так же порядка и длины схемы. А именно, схемы с пригодностью выше средней, а так же малым порядком и длиной характеризуются возрастанием количества своих представителей в последующих популяциях. Таким образом, с каждой итерацией, в результате действия селекции и генетических операторов происходит увеличение значений пригодности особей в популяции.

Генетический алгоритм предполагает псевдослучайное смешивание хромосом. Но предварительные (заказанные пользователем) условия какой-либо из задач могут не соответствовать аппаратным или программным ресурсам, которыми оперирует тот или иной процессор. Для того, чтобы в популяции в результате действия на хромосомы генетических операторов не образовались расписания с заведомо невозможными соответствиями задача-процессор, введем понятие фильтрующей функции. Фильтрующей назовем функцию, разрешающую размещение процесса задачи на процессоре только в том случае, если запрашиваемые ресурсы для размещаемой задачи соответствуют возможностям процессорного узла. Фильтрующая функция должна вызываться генетическими операторами, перед тем, как произвести действие над хромосомами для проверки корректности применения оператора.

Для описанной задачи критерий отбора реализуется целевой фнкцией (или функцией приспособленности). Эта функция позволяет оценить особи в популяции на каждой из итераций и выбрать лучшие из них, которые будут включены в следующую популяцию (новое поколение). Для описания работы функции пригодности рассмотрим одну хромосому. На каждом из ее участков, описывающих последовательность заданий на одном конкретном процессоре, возьмем сумму значений процессорного времени каждой из рассчитывающейся на этом процессоре задач. Максимальное значение всех полученных сумм является тем временным интервалом, в котором хотя бы один из процессоров будет недоступен для постановки на него новой задачи, ввиду загруженности задачами из предыдущего расписания.

Назовем такую максимальную сумму - временем расписания и обозначим за M. Построим график загруженности (рис. 2) процессоров (для наглядности рассмотрим тестовую хромосому из рис. 1). На осях графика отложим процессорное время расчета и порядковый номер процессора.

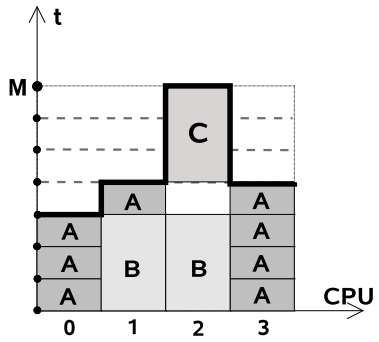

Рис. 2. График загруженности для тестового расписания

Верхняя граница загруженности (на графике 2 - линия, отмеченная жирным) определяет время, когда тот или иной процессор переходит в режим ожидания и готов принять для расчета другие задания (отсутствующие в данном расписании). Время, когда процессор простаивает, ввиду нехватки на узле каких либо из ресурсов (например, оперативной памяти, которую потребовала другая задача, считающаяся на втором процессоре двухпроцессорного SMP-узла), назовем окном расписания. Величина суммы количества времени простаивания, ограниченного сверху числом M (временем расписания) будет критерием эффективности расписания. Функция пригодности, следуя принципу минимума времени простаивания при минимуме времени окон отбирает особи, которые, пройдя процедуру селекции, будут включены в следующее поколение. Другими словами, чем плотнее запланированы задания на каждом из процессоров (т.е. чем меньше окон простаивания и быстрее наступает время высвобождения всех процессоров), тем эффективнее расписание.

Процесс поиска останавливается, когда изменение значения функции пригодности от поколения к поколению становятся незначительными. Величина, которая определяет незначительность таких изменений, а так же максимальное число поколений до остановки алгоритма определяется параметрами конфигурационного файла.

По окончанию поиска расписания существует вероятность того, что в очереди образовались не учтенные в расписании задания. При необходимости произвести повторное планирование генетические алгоритмы позволяют ускорить нахождение нового расписания. Для этого начальная популяция генерируется на основании последней полученной популяции из предыдущего поиска. Затем происходит применение генетического оператора кроссинговера попарно к особям обеих популяций. Таким образом, полученное новое поколение содержит в каждой из особей информацию о новых заданиях.

2.2. Формальная постановка задачи

Пусть $G = \{P, M, S, H\}$ - географически распределенная вычислительная GRID-сеть, представленная множеством программно и аппаратно разнародных процессорных узлов $P = \{P_1, P_2, ..., P_n\}$ с доступной на них оперативной памятью $M(P_i)$, постоянной памятью $H(P_i)$ и характеристиками программной среды $S(P_i) = \{S_n, S_v, R, Q\}$. Здесь S_n - имя операционной системы, S_v - версия операционной системы, R - тип локальной системы управления ресурсами, $Q = \{Q_1, Q_2, ..., Q_n\}$ - множество локальных очередей с такими характеристиками, как количество доступных очереди узлов, количество свободных узлов, максимальное время расчета задачи, максимальное процессорное время расчета, максимальное количество находящихся в очереди и запущенных задач, приоритет очереди.

Через F обозначим значение целевой функции. Рассмотрим задачу отыскания глобального минимума функции F, определенной на множестве Ω и представленной уравнением 2.

$$F = \sum_{i=1}^{N} T(P_i), \tag{2}$$

где N - общее количество всех процессоров в GRID-сети, $T(P_i)$ - время, в течении которого процессор загружен.

Через $X^* \subset \Psi$ обозначим множество решений этой задачи, принадлежащих множеству всевозможных расписаний Ψ. Через F^* обозначим минимальное значение целевой функции $F(x)$.

$$F^* = \min_{x \subset \Psi} F(x). \tag{3}$$

Необходимо с заданной точностью ε определить величину глобального минимума функции F и найти хотя бы одну точку, где это приближенное значение достигается. Введем множество ε-оптимальных решений задачи 3.

$$X^* = \left\{ x \subset \Psi : F(x) \leq F^* + \varepsilon; \widehat{F} \neq 0 \right\}, \tag{4}$$

где $\widehat{F}$ - значение фильтрующей функции, показанной в уравнении 5.

Пусть $J = \{j_k\}$ - множество задач, тогда

$$\widehat{F} = \begin{cases} 1 & , \text{ если } M(P_i) \geq M(j_k), H(P_i) \geq H(j_k) \\ 0 & , \text{ в противном случае} \end{cases}, \tag{5}$$

где $M(j_k)$ и $H(j_k)$- количество оперативной и постоянной памяти соответственно, запрошенные задачей j_k. Таким образом, условие 3 должно выполняться только при наличии достаточных для выполнения задачи ресурсов.

3. Реализация

Базовая часть алгоритма реализована на языке C++ и представляет собой исполняемую программу с CUI интерфейсом. На стандартном входе исполняемый файл получает (помимо численных параметров алгоритма) путь к директории, в которой располагаются файлы описания задач. Каждый файл описывает одну задачу пользователя. Все задачи равнозначны и не имеют приоритетов, то есть задача, поставленная в очередь первой, будет рассматриваться с таким же приоритетом, что и поставленная в очередь последней на определенный момент времени. Входными данными алгоритма является множество описаний заданий с параметрами, описывающими требуемые параметры процессора, операционной системы, оперативную и постоянную память, и так далее.

Базовая часть алгоритма не представляла бы большого практического значения без ее интеграции в GRID. Для этих целей был написан web-сервис WS-Geneur, работающий в окружении контейнера Globus Toolkit 4.x и являющийся, своего рода, интерфейсом к базовому модулю алгоритма. Необходимая информация о состоянии и параметрах ресурсов GRID-сети WS-Geneur получает от стандартного web-сервиса Globus Toolkit WS-MDS (Monitoring and Discovery System). Последний в свою очередь может черпать информацию у таких систем мониторинга ресурсов, как Ganglia (http://ganglia.info) или Nagios (http://www.nagios.org). На основании данной информации строиться таблица описания ресурсов. Она используется как при генерировании первой популяции, так и при работе фильтрующей функции, которая не дает образоваться заведомо противоречивым особям в популяции.

Для управления WS-Geneur и размещения новых заданий в очередь на планирование, а так же отладки написаны специальные консольные утилиты. При необходимости размещения нового задания пользователь должен создать файл описания задания и с помощью

консольного клиента поставить задание в очередь. Описываемые утилиты написаны на языке C++ и не требуют установленный Globus Toolkit, что позволяет работать пользователю удаленно, находясь за пределами GRID-сети.

4. Заключение

В статье описан алгоритм стохастического поиска оптимального расписания заданий в GRID на основе генетических алгоритмов. Использование подобных алгоритмов сокращает время простаивания вычислительных ресурсов в GRID, увеличивая качество расписания. При этом остается возможность уменьшения времени перепланирования за счет использования последней полученной популяции.

Литература

1. Д. Рутковская, М. Пилиньский, Л. Рутковский. Нейронные сети, генетические алгоритмы и нечеткие системы: Пер. с польск. И. Д. Рудковского. - М.: Горячая линия - Телеком, 2006 г.

2. В.Н. Коваленко, Д.А. Семячкин. Использование алгоритма Backfill в ГРИД, Труды международной конференции "Распределенные вычисления и Грид-технологии в науке и образовании"(Дубна, 29 июня-2 июля 2004 г.).-Дубна: 2004, сс. 139-144.

3. S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, Optimization by Simulated Annealing. Science, Vol 220, Number 4598, pages 671-680, 1983.

4. C. Ernemann. Economic scheduling in Grid computing; C. Ernemann,V. Hamscher,R.Yahyapour. In D. Feitelson and L. Rudolph, editors, Job Scheduling Strategies for Parallel Processing (Proceedings of the Eighth International JSSPP Workshop; LNCS 2537). Springer-Verlag, 2002. p. 129–152.

5. J.B. Weissman. Gallop: The Benefits of Wide-Area Computing for Parallel Processing. Journal of Parallel and Distributed Computing, V. 54(2), November 1998.

6. В.Н. Коваленко, Е.И. Коваленко, Д.А. Корягин, Э.З. Любимский. Метод опережающего планирования для грид. Препринт ИМП. 112. М.: 2005г., 33с.

7. E.S.H. Hou, N. Ansari, H. Ren. A Genetic Algorithm for Multiprocessor Scheduling. IEEE Transactions on Parallel and Distributed Systems, vol. 05, no. 2, pp. 113-120, Feb., 1994. ISSN: 1045-9219

8. В.А. Костенко, Р.Л. Смелянский, А.Г. Трекин. Синтез структур вычислительных систем реального времени с использованием генетических алгоритмов. Программирование, 2000, н.5, С.63-72.

9. Michelle Moore. An Accurate and Efficient Parallel Genetic Algorithm to Schedule Tasks on a Cluster. International Parallel and Distributed Processing Symposium (IPDPS'03), 2003. ISBN: 0-7695-1926-1