Генетический алгоритм составления расписания выполнения параллельных заданий в распределенной вычислительной системе*

С. И. Смагин, Т. С. Шаповалов
Вычислительный центр ДВО РАН, Хабаровск, Россия
e-mail: smagin@as.khb.ru, taras@as.khb.ru

Предложен генетический алгоритм для решения задачи составления расписания выполнения параллельных заданий в распределенной вычислительной системе. Представлен механизм учета ресурсных ограничений, а также зависимостей между заданиями в расписаниях при применении генетического алгоритма. Описан модифицированный алгоритм обратного заполнения с учетом ресурсных ограничений, адаптированный для составления расписаний начальной популяции в генетическом алгоритме. Представлены результаты численных экспериментов.

Ключевые слова: расписание, генетический алгоритм, параллельные задания, распределенная вычислительная система.

Введение

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

Примерами РВС могут служить системы, построенные по технологии Grid [1, 2]; Clouds Computing [3,4]; системы, использующие вычислительное время простаивающих рабочих станций [5, 6]. С начала 2000 г. были введены в эксплуатацию ряд крупных Grid: EGEE (Enabling Grids for E-sciencE), NorduGrid, Open Science Grid, TeraGrid и др. Выполняются проекты по распределению вычислительной работы между простаивающими рабочими станциями с использованием сети Интернет: Folder@Home, GIMPS (Great Internet Mersenne Prime Search), LHC@Home и др.

Эффективность работы РВС напрямую зависит от методов составления расписаний выполнения заданий. В общем случае задача составления расписания является NP-полной. Ранее в работе [7] рассматривалась задача составления расписания выполнения непараллельных заданий, сделан обзор методов составления расписаний для РВС. Данная работа является продолжением [7]. В ней рассматривается задача со-

*Работа выполнена в рамках ФЦП "Научные и научно-педагогические кадры инновационной России" (проект № 02.740.11.0626) и поддержана грантами ДВО РАН № 09-I-П1-01 и 10-III-В-01И-009.

ставления расписания параллельных заданий для РВС с применением генетического алгоритма (ГА).

Под процессом будем понимать поток инструкций для процессора ЭВМ с единым адресным пространством, значениями регистров процессора, стеком, открытыми файлами, глобальными переменными и т. п. Параллельными будем считать задания с числом процессов $N_p > 1$, которые потенциально могут обмениваться данными между собой, и, таким образом, эти процессы должны быть спланированы на одновременный запуск [8–10]. Такой вид планирования называется совместным (синонимы: связанным, комплектным, gang-scheduling). Процессы одного задания должны всегда выполняться в пределах одной группы процессоров, например, многопроцессорной системы с общей памятью или вычислительного кластера [11].

В качестве ресурсов будем рассматривать совокупность программных и аппаратных средств для выполнения процессов. Примеры ресурсов: процессор, среда передачи данных, прикладное программное обеспечение, система хранения данных и т. д. Виртуальный узел (ВУ) — один вычислительный элемент (ядро процессора), рассматриваемый в связке с сопоставленными ему ресурсами. Если несколько процессоров используют одни и те же ресурсы, характеризуемые объемом (оперативная память, постоянная память и т. п.), то будем считать, что объем этих ресурсов равномерно делится на все виртуальные узлы, соответствующие данным процессорам.

1. Постановка задачи составления расписания выполнения параллельных заданий в распределенной вычислительной системе

Разделим отрезок времени T, в течение которого поступают задания пользователей, на $(Q+1)$ последовательных отрезков (периодов) планирования: $T = \bigcup T_q$, где q — порядковый номер периода планирования, $q = 0, 1, \ldots, Q$. В каждый из периодов T_q накапливается некоторое множество параллельных заданий. Каждое параллельное задание включает запрос на выполнение нескольких одновременно выполняемых процессов.

Пусть $U_q = \{u_{iq} : \ i = 1, 2, \ldots, N_q\}$ — множество процессов, принадлежащих заданиям, планируемым в период T_q, N_q — число процессов, принадлежащих всем заданиям, планируемым в период T_q. Для выполнения задания пользователь заказывает количество требуемых ресурсов определенных видов. Данные требования не меняются во время выполнения процессов. Если пользователь не указал для задания требование к ресурсу определенного вида, то для данного вида ресурса заказывается значение по умолчанию, заданное администратором РВС.

Пусть $\Phi = \{\phi_m : \ m = 1, 2, \ldots, M\}$ — неизменяемое в течение всего отрезка времени T множество всех виртуальных узлов в РВС, где $M \geq 1$ — число виртуальных узлов в РВС. Тогда расписание S_q выполнения процессов $u_{iq} \in U_q$, планируемых в период T_q в РВС, определяется как множество отрезков времени (слотов) s_{imq}, зарезервированных для выполнения этих процессов на ВУ $\phi_m \in \Phi$:

$$S_q = \{s_{imq} : \ i \in I_q, \ m \in M_q\}, \tag{1}$$

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

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

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

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

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

Введем разбиение множества Φ всех виртуальных узлов в РВС на непересекающиеся подмножества $\Phi_g = \{\phi_m : \ m \in M_g\}$ — группы ВУ:

$$\Phi = \bigcup_{g=1}^{N_g} \Phi_g, \quad \Phi_g \bigcap \Phi_p = \varnothing, \quad g, p = 1, 2, \ldots, N_g, \quad g \ne p, \tag{3}$$

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

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

$$\bigcup_{p=1}^{P_q} U_{pq} = U_q, \quad U_{lq} \bigcap U_{pq} = \varnothing, \quad l, p = 1, 2, \ldots, P_q, \quad l \ne p, \tag{4}$$

где P_q — количество заданий, поступивших в период планирования T_q. Тогда под непротиворечивостью расписания S_q выполнения процессов из множества U_q понимается выполнение следующих условий.

1. Каждому процессу из множества U_q, который может быть запущен в РВС, должен быть сопоставлен один слот в расписании S_q. Пусть N'_q — число слотов в расписании S_q, соответствующих планируемым в период T_q процессам, N_q — число процессов всех заданий, планируемых в период T_q, которые могут выполняться в РВС. Тогда должно быть справедливо равенство $N'_q = N_q$.

2. Никакие два слота $s_{imq} \in S_q$ и $s_{jmq} \in S_q$ $(i \ne j)$ не должны быть сопоставлены одному ВУ ϕ_m одновременно: $|s_{imq} \bigcap s_{jmq}| = 0$, где $|\cdot|$ — длина соответствующего отрезка времени.

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

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

В большинстве случаев для одного множества процессов U_q можно составить множество расписаний $\widetilde{S}_q = \{S_{\alpha q} : \alpha = 1, 2, \ldots, A'_q\}$, где A'_q — число возможных расписаний. Введем для расписаний $S_{\alpha q} \in \widetilde{S}_q$ целевую функцию $f : \widetilde{S}_q \to R_+$, где R_+ — множество действительных положительных чисел. Оптимальным будем считать расписание $S^*_q \in \widetilde{S}_q$, удовлетворяющее условию

$$f(S^*_q) = \min_{\alpha=1,2,\ldots,A'_q} f(S_{\alpha q}).$$

Выбор подходящей целевой функции является важным компонентом алгоритма составления расписания выполнения заданий в РВС. Необходимо выбрать целевую функцию, которая позволяла бы учесть приоритеты процессов и равномерно загрузить работой ВУ.

В связи с тем что задача составления расписания в общем случае является NP-полной, на практике приходится ограничиваться поиском субоптимального решения [12].

Сформулируем задачу составления расписания выполнения параллельных заданий в РВС.

Задача. Найти субоптимальное расписание S_q выполнения процессов $u_{iq} \in U_q$ вида (1). При этом S_q должно удовлетворять условиям ресурсных ограничений (2) и условиям непротиворечивости 1–4.

2. Генетический алгоритм

На рис. 1 представлена схема генетического алгоритма для решения задачи составления расписаний выполнения параллельных заданий.

Далее номер периода планирования q опускается, так как он не требуется для понимания работы алгоритма составления одного расписания. Для описания деталей разработанного алгоритма будут использоваться следующие понятия и обозначения.

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

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

1. Создание начальной популяции. С помощью модифицированного алгоритма обратного заполнения для заданий из множества V создается начальная популяция — множество $X = \{X_\alpha : \alpha \in A\}$, где X_α — хромосомы.

2. Вычисление значений функции пригодности f_α по формуле (6) (см. ниже) для всех хромосом из X.

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

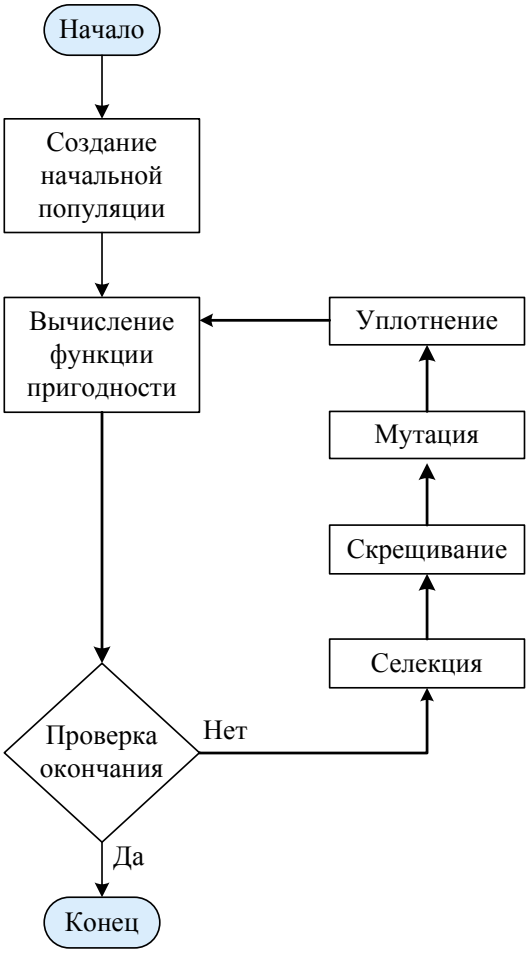

Рис. 1. Схема генетического алгоритма составления расписания выполнения параллельных заданий в распределенной вычислительной системе

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

5. Селекция. Создается подмножество $X' \subseteq X$ хромосом, выбранных методом рулетки [13].

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

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

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

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

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

Длина $|s_{im\alpha}|$ (продолжительность) слота $s_{im\alpha}$ рассчитывается исходя из параметров, указанных пользователем в файле описания соответствующего задания. При составлении расписания выполнения заданий для гетерогенной РВС необходимо масштабировать длину слотов в соответствии с фактически используемым типом процессора. Новое значение длины слота вычисляется по формуле

$$|s_{im\alpha}| = |s_{ih\alpha}|\frac{\kappa_h}{\kappa_m}, \tag{5}$$

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

3. Функция пригодности

Часто в качестве критериев оценки пригодности расписаний при использовании ГА применяются критерии длины расписания и среднего взвешенного времени прохождения заданий. Оба критерия имеют существенные достоинства и недостатки при их применении для поиска расписания выполнения параллельных заданий в РВС. Длина расписания учитывает лишь последнее по времени завершения задание и плохо "справляется" с уплотнением процессов в расписании. Критерий среднего взвешенного времени прохождения заданий "справляется" с уплотнением расписания. Тем не менее значение функции пригодности с данным критерием зависит от порядка следования заданий. Для равнозначных расписаний это ведет (при большом числе сильно различающихся по длительности заданий) к существенному разбросу значений функции пригодности, что мешает работе ГА.

Для устранения вышеописанных недостатков, а также в целях учета приоритета заданий используется следующая функция пригодности для оценки хромосомы X_α:

$$f_\alpha = \sqrt[\widetilde{N}]{\prod_{m\in M_\alpha}\sum_{i\in I_{m\alpha}}\lambda_{im\alpha}|s_{im\alpha}|}, \tag{6}$$

где M_α — множество номеров ВУ, которым сопоставлены слоты из хромосомы X_α; $I_{m\alpha}$ — множество номеров слотов, соответствующих ВУ ϕ_m; $\widetilde{N}$ — число всех виртуальных узлов в РВС; $|s_{im\alpha}|$ — длина слота $s_{im\alpha} \in X_\alpha$; $\lambda_{im\alpha}$ — вес процесса u_i при запуске его на ВУ ϕ_m. Вес $\lambda_{im\alpha}$ процесса u_i на ВУ ϕ_m необходим для учета его приоритета: $\lambda_{im\alpha} = (p_{i\alpha})^{d_{im\alpha}}$, где $p_{i\alpha}$ — приоритет процесса u_i, $d_{im\alpha}$ — порядковый номер следования слота $s_{im\alpha}$ на ВУ ϕ_m.

4. Учет зависимостей между заданиями

Между заданиями могут существовать зависимости, возникающие при ожидании заданием результатов выполнения других заданий. В общем случае такие зависимости

можно представить в виде направленного ацикличного графа. Будем рассматривать ситуацию, когда каждое задание может зависеть только от одного задания, что в большинстве случаев достаточно. Если i-е задание зависит от j-го, то должно выполняться условие $t_{im\alpha} \geq t_{jm\alpha} + |s_{jm\alpha}|$, где $t_{im\alpha}$ и $t_{jm\alpha}$ — моменты времени запуска соответствующих заданий, $|s_{jm\alpha}|$ — длительность выполнения процесса j-го задания на ВУ $\phi_m \in \Phi$, $m \in M_\alpha$, M_α — множество номеров ВУ, которым соответствуют слоты в расписании, кодируемом хромосомой X_α, $i, j \in I_\alpha$, I_α — множество номеров слотов в хромосоме X_α.

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

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

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

$$s_{in\alpha} \subseteq [t_{jn\alpha}, t^{\max}_{in\alpha}], \quad s_{jm\alpha} \subseteq [t_{im\alpha}, t^{\max}_{jm\alpha}]. \tag{7}$$

5. Учет ресурсных ограничений

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

Важная задача при применении ГА к составлению расписания для РВС — разработка механизма учета указанных ограничений. Представляемый алгоритм является ресурсно-зависимым — т. е. учитываются ограничения (2) на ресурсы, необходимые заданиям. Его идея заключается в сокращении пространства выбора ВУ перед применением генетических операторов.

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

1. Обозначим через $B^e_{i\alpha} = \{b^e_{in\alpha} :\ n \in M\}$ булевый вектор, значение компоненты $b^e_{in\alpha}$ которого равно 1, если для номера ресурса e, ВУ ϕ_n и слота $s^*_{im\alpha}$ выполняется неравен-

ство (2), и $b^e_{in\alpha} = 0$ — в противном случае. Элемент $b^e_{in\alpha}$ данного вектора устанавливает возможность сопоставить слот $s^*_{im\alpha}$ с ВУ ϕ_n без нарушения ресурсного ограничения для ресурса номер e.

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

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

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

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

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

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

6. Создание начальной популяции

Для создания хромосом начальной популяции применялся модифицированный алгоритм обратного заполнения (Backfill), часто используемый при составлении расписаний как для вычислительных кластеров [14], так и для РВС типа Grid [15] и на сегодня эталонный при составлении расписания параллельных заданий для РВС.

Входными данными алгоритма являются: $V = \{v_p : \ p = 1, 2, \ldots, P\}$ — множество заданий, удовлетворяющих условию (4), P — число заданий, для которых составляется расписание; Ф — объединение групп ВУ вида (3). Результат работы алгоритма — хромосома $X_\alpha \in X$, где X — множество хромосом создаваемой популяции.

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

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

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

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

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

2.2. Если $i = P$, то работа алгоритма завершается.

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

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

$$t^* = t_{jm\alpha} + |s_{jm\alpha}|, \tag{8}$$

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

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

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

2.5. Создается маска ресурсов B^*_α для задания v^*.

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

2.7. К хромосоме X_α добавляются слоты для процессов задания v^* с моментом начала, равным началу холостых слотов окна w^*. Длина холостых слотов этого окна уменьшается на длину слотов задания v^* либо эти холостые слоты удаляются из хромосомы X_α, если их длина совпадает с длиной вставляемых слотов. Если на шаге 2.3 по формуле (8) был установлен нижний предел планирования t^* для задания v^*, то соответствующие слоты должны начинаться в момент t^*. Для получившихся отрезков времени простаивания ВУ в хромосоме X_α создаются холостые слоты.

7. Оператор скрещивания

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

На рис. 2 схематично показан процесс “сборки” дочерней хромосомы из двух родительских (у слотов $s_{im\alpha}$ опущен индекс m (номер ВУ)). Точки разрыва оператора скрещивания выбираются строго между множествами слотов, сопоставленных различным группам ВУ.

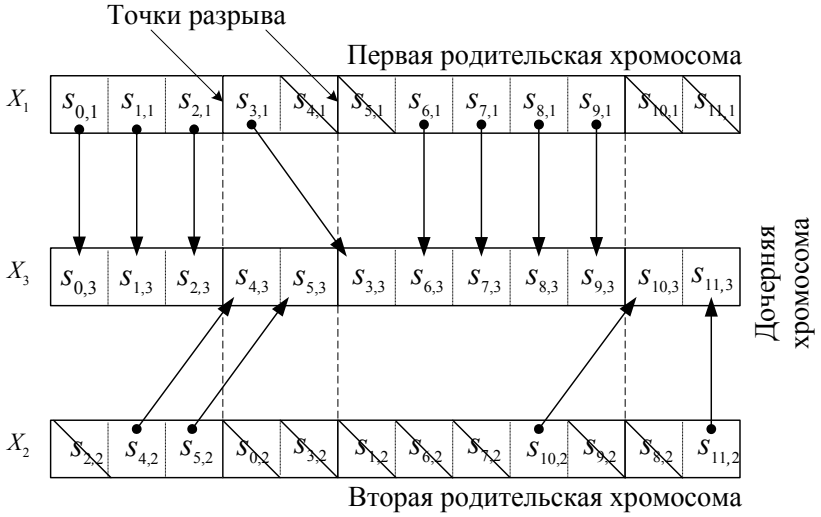

Рис. 2. Схема скрещивания двух хромосом

Алгоритм оператора скрещивания следующий.

1. Множества слотов X_1 и X_2 разбиваются на непересекающиеся подмножества

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

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

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

8. Оператор мутации

Ниже описан алгоритм оператора мутации в ГА составления расписания выполнения параллельных заданий в РВС, разбитый на две части. В первой части рассмотрена последовательность действий для выбора слотов двух заданий, во второй — описываются действия по обмену слотов этих заданий.

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

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

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

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

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

5. Определяется подмножество $Y_\alpha \subseteq X'_\alpha$ слотов, чьи маски ресурсов $B_{k\alpha}$ включают элемент $b_{km\alpha} = 1$, где m — номер ВУ первоначально выбранного слота $s_{jm\alpha}$. Таким образом, Y_α — подмножество слотов, на которые выбранный на первом шаге слот $s_{jm\alpha}$ может быть заменен без нарушения определяемых неравенством (2) ресурсных ограничений.

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

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

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

9. Определяется подмножество $Z_\alpha \subseteq Y'_\alpha$, состоящее только из таких слотов $s_{in\alpha}$, что для $s_{jm\alpha}$ и $s_{in\alpha}$ соблюдаются условия коридора мутации (7).

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

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

соответствующих слотам $s_{im\alpha} \in Z_\alpha$. Определяется подмножество Z'_α слотов $s_{in\alpha} \in Z_\alpha$, для которых $N_{1\alpha} \le M'_i$ и $N'_{i\alpha} \le M_j$.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

9. Оператор уплотнения

В результате переупорядочивания слотов в хромосоме оператором мутации могут появляться множества холостых слотов. Часть хромосом в популяции после мутации можно сжимать, уменьшая длину холостых слотов или исключая эти слоты, если их длину можно уменьшить до нуля. Входным параметром алгоритма оператора уплотнения является хромосома $X_\alpha \in X$. Оператор уплотнения описывается в следующем виде.

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

$$t_{pm\alpha} = t^0_{im\alpha} + |s^0_{im\alpha}|, \quad m \in M, \quad i \in I^0_\alpha, \tag{9}$$

где $t^0_{im\alpha}$ и $|s^0_{im\alpha}|$ — начало и длина холостого слота $s^0_{im\alpha} \in X_\alpha$, $t_{pm\alpha}$ — момент начала задания $v_{p\alpha}$ на ВУ ϕ_m, I^0_α — множество номеров холостых слотов из хромосомы X_α, M — множество всех ВУ в РВС. Холостые слоты, удовлетворяющие равенству (9) формируют множество Y_α.

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

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

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

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

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

7. Осуществляется переход на шаг 1.

10. Результаты численных экспериментов

Решение задачи составления расписания выполнения заданий с использованием ГА считалось найденным, если максимальное значение функции пригодности в популяции отличалось от наилучшего (заранее известного) решения не более чем на 5 %. Генерировалось множество заданий для расписания с известным значением функции пригодности. Длина оптимального расписания — 1000 ч. В качестве описания ресурсов РВС для тестирования были взяты два кластера Вычислительного центра ДВО РАН. Краткие характеристики кластеров приведены в таблице.

На практике алгоритм обратного заполнения при планировании параллельных заданий с небольшим числом процессов является достаточно эффективным и быстрым.

Генетический алгоритм может уступать ему в скорости. При применении ГА важным является знание условий, при которых он может составлять более эффективные расписания по сравнению с общепринятым алгоритмом. В данном случае учет этих условий необходимо осуществлять при принятии решения: оптимизировать очередь заданий с помощью ГА или остановиться на более быстром алгоритме обратного заполнения.

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

Необходимо отметить, что значение N_p ограничивает лишь верхнюю границу числа процессов каждого задания, в то время как нижняя граница равнялась 1. Для каждого прогона задания генерировались заново случайным образом с числом процессов от 1 до N_p. Учитывались средние значения по 30 прогонам для каждого сочетания факторов. Все задания имели одинаковый приоритет, равный 1.

Одним из параметров, позволяющим оценить работу алгоритма, является загруженность w_j, представляющая собой отношение времени всех ВУ под нагрузкой выполнения заданий к максимально возможному уровню загруженности для заданной длины расписания в поколении j генетического алгоритма:

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

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

На рис. 3 представлена зависимость выигрыша в загруженности $w_I - w_0$ от общего числа процессов N. Здесь w_0 — загруженность ресурсов согласно расписанию, полученному алгоритмом обратного заполнения (использовалось лучшее расписание в начальной популяции), w_I — загруженность ресурсов согласно расписанию, оптимизированному ГА (бралось лучшее расписание в последней популяции номер I). По приведенным на рисунке зависимостям видно, что при $N \ge 128$ ГА может оптимизировать расписание, увеличивая загруженность рассматриваемых ресурсов. При этом больший выигрыш по загруженности наблюдается при относительно низких значениях общего числа процессов N. Резкое увеличение разницы между эффективностью двух алгоритмов при небольших N объясняется тем, что при большем числе процессов алгоритм обратного заполнения имеет возможность оперировать большим числом окон, что ведет к более плотной загруженности ресурсов.

Среднее время простаивания вычислительных ресурсов Δ_j в поколении j генетического алгоритма есть величина, показывающая среднее время ожидания всех ВУ в РВС

Характеристики вычислительных кластеров ВЦ ДВО РАН

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

после выполнения всех запланированных расписанием процессов до момента времени, соответствующего длине расписания:

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

где l_{mj} — длина подрасписания на ВУ $\phi_m \in \Phi$ в поколении j. Тогда сэкономленное время простаивания есть разность $\Delta_0 - \Delta_I$, где Δ_0 — среднее время простаивания ресурсов в лучшем расписании, сгенерированном алгоритмом обратного заполнения для популяции номер 0, Δ_I — среднее время простаивания ресурсов в лучшем расписании в последнем поколении ГА.

Практический выигрыш наблюдается на зависимости сэкономленного времени простаивания от N и N_p (рис. 4). По этим зависимостям видно, что в данном случае можно сэкономить порядка 8–12 ч процессорного времени на каждый ВУ. Однако при $N \leq 128$ включать оптимизацию расписания, уже полученного с помощью алгоритма обратного заполнения, нецелесообразно.

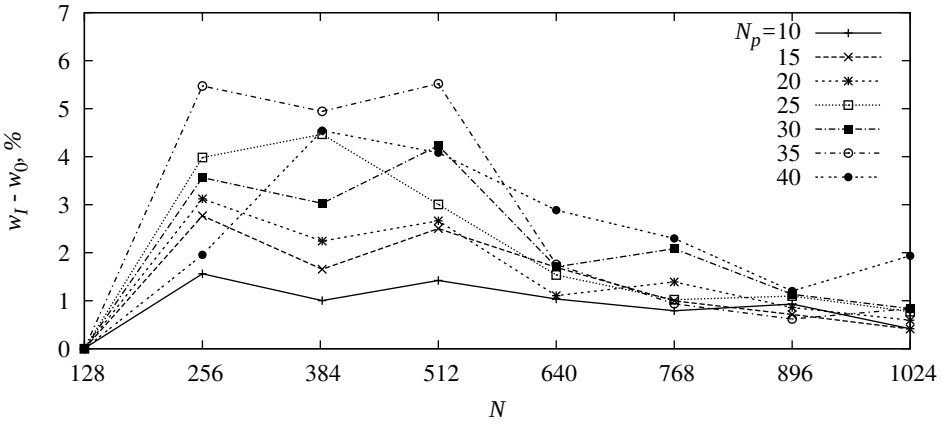

Рис. 3. Зависимость выигрыша в загруженности ресурсов $w_I - w_0$ от числа процессов N и максимального числа процессов одного задания N_p

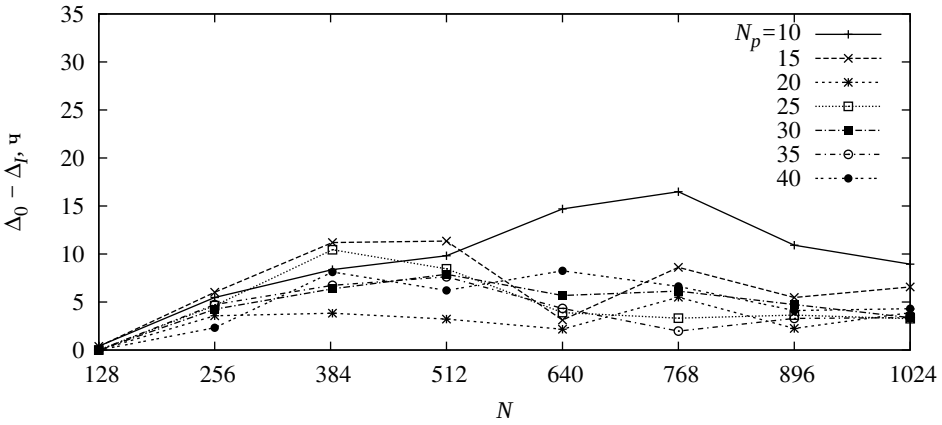

Рис. 4. Зависимость сэкономленного времени простаивания $\Delta_0 - \Delta_I$ от числа процессов N и максимального числа процессов одного задания N_p

Значения длин l расписаний, возникающих при различном значении N_p, можно оценить по зависимостям, приведенным на рис. 5. С ростом N_p возрастает и длина расписания. Это происходит в силу увеличения отрезков времени простаивания ВУ, которые возникают в силу условия одновременности запуска всех процессов параллельного задания.

Другой характеристикой, позволяющей оценить работу алгоритма, является среднее время ожидания запуска заданий

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

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

На рис. 6 показана зависимость $\delta_0 - \delta_I$ от числа процессов N всех заданий. Здесь δ_0 и δ_I — значения среднего времени ожидания в расписаниях, полученные соответственно алгоритмом обратного заполнения и ГА в последнем поколении. Как и в случае с зависимостью $w_I - w_0$ от N, на рис. 6 при небольших N наблюдается увеличение времени ожидания запуска заданий $\delta_0 - \delta_I$. С ростом N выигрыш от использования ГА уменьшается и затем стабилизируется в некотором коридоре значений.

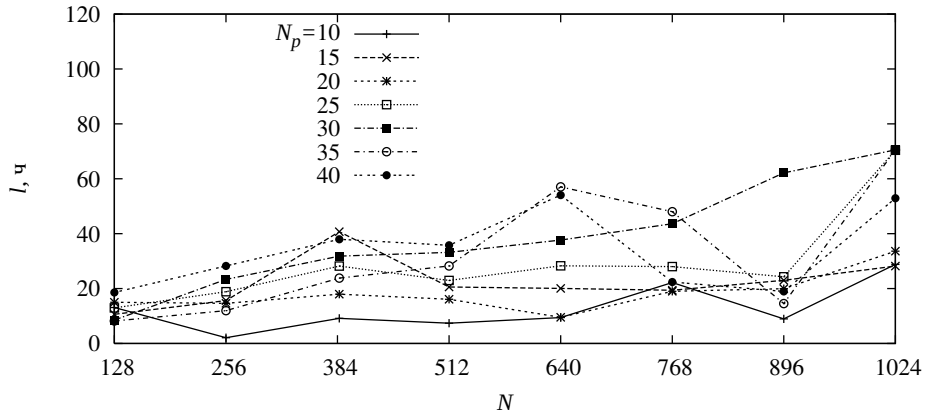

Рис. 5. Зависимость длины l расписания от числа процессов N и максимального числа процессов одного задания N_p

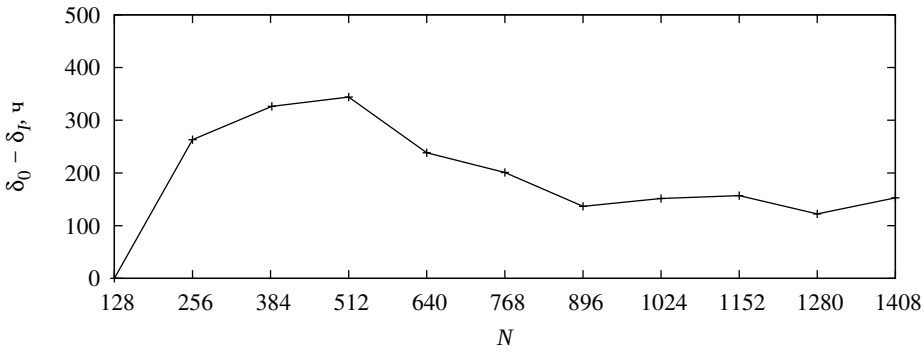

Рис. 6. Зависимость разницы значений среднего времени ожидания запуска заданий $\delta_0 - \delta_I$ от числа процессов N

Таким образом, для задачи составления расписания выполнения параллельных заданий в РВС предложен и программно реализован ГА, учитывающий особенности предметной области. Проведены экспериментальные исследования разработанного ГА на различных множествах заданий. Исследования показали, что в результате применения ГА вместо алгоритма обратного заполнения для составления расписания выполнения параллельных заданий в РВС можно более эффективно загружать вычислительные ресурсы полезной работой.

Список литературы

[1] Foster I., Kesselman C., Nick J., Tuecke S. The physiology of the Grid: An open Grid services architecture for distributed dystems integration // Comput. Networks: The Intern. J. Comput. and Telecommunicat. Networking. 2002. Vol. 40, No. 1. P. 5–17.

[2] Foster I., Kesselman C., Tuecke S. The anatomy of the Grid: Enabling scalable virtual organizations // Intern. J. High Performance Comput. Appl. 2001. Vol. 15, No. 3. P. 200–222.

[3] Brian H. Cloud computing // Communicat. ACM. 2008. Vol. 51, No. 7. P. 9–11.

[4] Luis V., Juan C., Maik L. A break in the clouds: towards a cloud definition // ACM SIGCOMM Comput. Communicat. Rev. 2008. Vol. 39, No. 1. P. 50–55.

[5] Филамофитский М.П. Система поддержки метакомпьютерных расчетов X-Com: Архитектура и технология работы // Вычисл. методы и программирование. 2004. Т. 5, № 2. С. 123–137.

[6] Heien E.M., Anderson P.D., Hagihara K. Computing low latency batches with unreliable workers in volunteer computing environments // J. Grid Comput. 2009. Vol. 7, No. 4. P. 501–518.

[7] Шаповалов Т.С., Пересветов В.В. Генетический алгоритм составления расписаний для распределенных гетерогенных вычислительных систем // Вычисл. методы и программирование. 2009. Т. 10, № 1. С. 159–167.

[8] Feitelson D.G., Jette M.A. Improved utilization and responsiveness with gang scheduling in job scheduling strategies for parallel processing // Lecture Notes Comput. Sci. 1997. Vol. 1291. P. 238–261.

[9] Yoo A.B., Jette M. An efficient and scalable coscheduling technique for large symmetric multiprocessor clusters // Ibid. 2001. Vol. 2221. P. 21–40.

[10] Zhang Y., Franke H., Moreira J., Sivasubramaniam A. An integrated approach to parallel scheduling using gang-scheduling, backfilling and migration // Ibid. 2001. Vol. 2221. P. 133–158.

[11] Baker M. Cluster Computing White Paper. Portsmouth: Univ. Portsmouth, 2000. 119 p.

[12] Коффман Э.Г. Теория расписаний и вычислительные машины. М.: Наука, 1984. 336 с.

[13] Holland J.H. Adaptation in Natural and Artificial Systems: An Introductory Analysis With Applications to Biology, Control, and Artificial Intelligence. Cambridge: The MIT Press, 1992. 228 p.

[14] Топорков В.В. Модели распределенных вычислений. М.: Физматлит, 2004. 320 с.

[15] Коваленко В.Н., Семячкин Д.А. Использование алгоритма Backfill в Грид // Распределенные вычисления и Грид-технологии в науке и образовании. Дубна: ОИЯИ, 2004. С. 139–144.

Поступила в редакцию 1 июля 2010 г.