

РЕШЕНИЕ ЗАДАЧИ ДИСПЕТЧЕРИЗАЦИИ В РАСПРЕДЕЛЕННЫХ ТЕЛЕКОММУНИКАЦИОННЫХ СИСТЕМАХ

Введение

Системы распределенной обработки, или распределенные системы (РС), функционирующие в компьютерных и телекоммуникационных сетях, являются одной из наиболее перспективных и быстро развивающихся областей информатики [1]. Такое место они заняли благодаря их существенным преимуществам по сравнению с изолированными системами, функционирующими на базе отдельных компьютеров. РС характеризуются потенциально более высокой надежностью, гибкостью использования вследствие представления пользователю широкого спектра информационных и вычислительных услуг, высокой степенью параллелизма обработки данных, то есть высокой производительностью. Однако достижение отмеченных достоинств сопряжено с решением комплекса проблем, связанных со значительным усложнением механизмов управления информационными процессами, особенно при децентрализации последнего. При этом особое значение имеет обеспечение целостности и непротиворечивости распределенных баз данных (РБД) – одного из основных функциональных компонентов РС [1].

Рассмотрим решение задачи диспетчеризации в распределенных телекоммуникационных системах. Для обеспечения практической применимости GRID систем должна быть решена проблема обеспечения качества обслуживания (QoS – quality of service) пользователей [2]. Качество обслуживания – многоаспектное понятие, включающее: безопасность участвующих в GRID ресурсов и безопасность выполняющихся заданий, надежность и постоянную доступность ресурсов, а также требуемую оперативность выполнения заданий. Системы диспетчеризации реализуются с помощью инструментальной среды Globus Toolkit [2], являющейся стандартом де-факто в области создания GRID систем. Globus Toolkit позволяет использовать систему диспетчеризации в имеющихся GRID -инфраструктурах. Так, например, рассмотрим программный комплекс GrAS (Grid Advanced Scheduler) [3] – разработанный в ИПМ им. М.В. Келдыша и реализующий централизованный подход к диспетчеризации заданий в GRID с использованием локального прогнозирования. GrAS создан с использованием инструментальной среды Globus Toolkit. Коммуникации распределенных компонентов друг с другом реализованы с помощью аппарата GRID -служб – стандартным средством дистанционного взаимодействия в GRID. GrAS состоит из трех основных компонент (рис.1):

- интерфейса пользователя, который позволяет отправить задание в GrAS, узнать статус уже отправленного задания, изменить его параметры, отменить его и получить результаты;
 - кластерного Агента, который составляет прогноз занятия/освобождения ресурсов в кластере;
 - серверной части GrAS, которая осуществляет прием сообщений от Агентов и пользователей, распределение заданий по ресурсам и их запуск.
- Серверная часть GrAS в свою очередь содержит в себе:
- ядро, которое обрабатывает поступающие события и осуществляет планирование;
 - утилиту запуска заданий в кластер и управления запущенными заданиями – Job Control;
 - грид-службы, принимающие сообщения от Агентов и пользователей;
 - базу данных, в которой хранится очередь заданий и информация о ресурсах;
 - очередь сообщений.

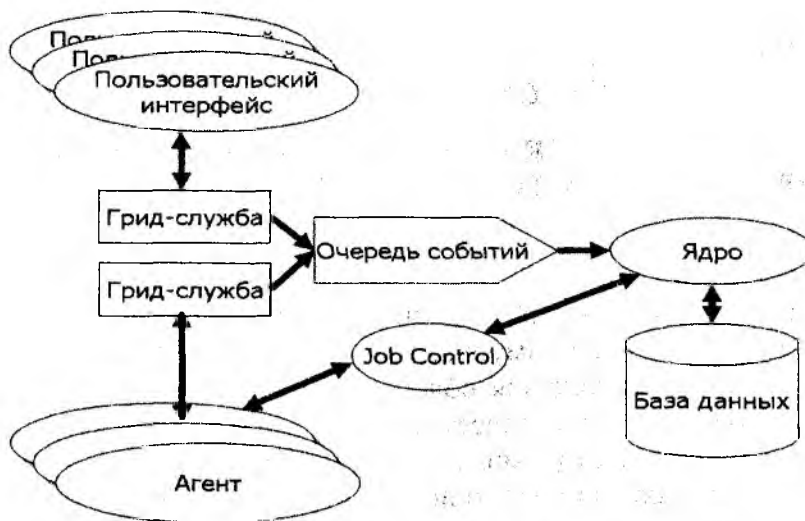


Рис.1. Структурная схема GrAS

Очередь предназначена для буферизации сообщений, приходящих от Агентов и пользователей, перед тем, как они будут обработаны ядром GrAS. Буферизация сообщений в очереди позволяет работать пользователям и Агентам с диспетчером в асинхронном режиме. Информация в очереди хранится в порядке поступления, однако поддерживается механизм позволяющий выбрать из любого места очереди сообщения определенного типа. Все содержимое очереди физически хранится в базе данных.

В базе данных GrAS содержится информация, имеющаяся в распоряжении системы диспетчеризации. В качестве СУБД используется система PostgreSQL. Выбор в пользу данной СУБД был сделан по нескольким причинам. Во-первых, это открытость PostgreSQL, а во-вторых, поддержка механизма хранимых процедур, позволяющих повысить скорость работы с базой данных на несколько порядков. К тому же PostgreSQL является реляционной базой данных, что соотносится с характером хранящейся в ней информации. В GRID системах возможно возникновение пиковых нагрузок, а именно большого количества запросов на входе СУБД, возникают отказы в обслуживании запросов [4, 5]. Поэтому актуальным является разработка наиболее эффективного метода разрешения очереди запросов.

Формализация задачи обработки запросов в СУБД

В общем виде модель СУБД представлена на рис.2.

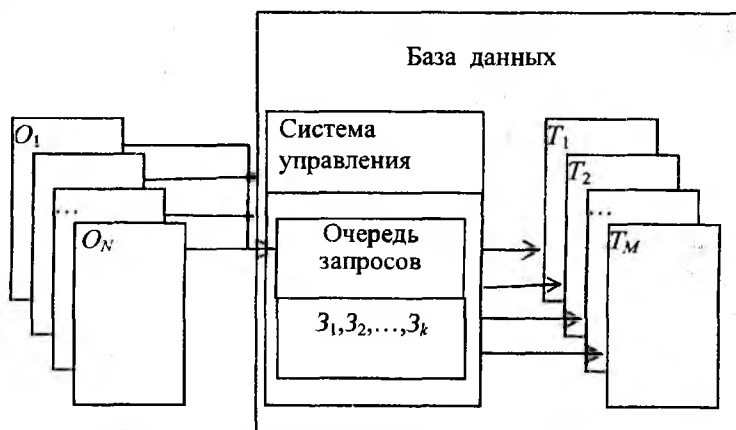


Рис. 2. Схема управления базой данных

База данных состоит из M таблиц T_i , $i = \overline{1..M}$, содержащих информацию и системы управления базой данных. К базе данных имеют доступ N клиентов O_j , $j = \overline{1..N}$, которые формируют запросы к базе данных [1, 4]. Каждый запрос Z_k от любого клиента O_j имеет свой приоритет C_k , зависящий от уровня привилегий клиента. Момент времени, в который от клиента поступает запрос, – величина случайная. Поэтому на входе системы управления при высокой интенсивности запросов образуются очереди запросов. Для математического описания указанной схемы управления базой данных целесообразно использовать теорию массового обслуживания, в которой под заявкой необходимо понимать запрос оператора к базе данных.

При наличии ограничений на размер хранимой очереди в системе управления может возникнуть ситуация, при которой очередной запрос, поступивший на вход системы управления, не будет принят к обслуживанию, что приведет к его потере или к задержке в обслуживании. Поэтому остро стоит вопрос разработки системы управления базой данных с приемлемой скоростью обслуживания запросов.

Следует отметить дополнительное требование к системе управления базой данных в GRID системах – СУБД имеет одновременный доступ к любому количеству таблиц. Наиболее перспективными вариантами метода обслуживания запросов являются [6 – 10]:

- метод групповой выборки;
- метод групповой выборки с индивидуальным сегментированием.

Метод групповой выборки – метод, при реализации которого из очереди запросов обслуживаются несколько запросов одновременно. Выбираются запросы, которые требуют информацию из разных таблиц и чтобы сумма их приоритетов была максимальной. В случае наличия равнозначных запросов выбирают более «старые».

Поэтому необходимо выбрать из очереди как можно большее количество запросов, которые обращаются к разным таблицам, и чтобы сумма приоритетов выбранных запросов была бы максимальна. Стремление к максимуму суммы приоритетов выбранных запросов является главным критерием при выборе запросов из очереди.

Пусть $\{\bar{X}\}$ – множество всех вариантов выбора запросов из очереди, \bar{X} – один из вариантов выбора запросов. Причем $\bar{X} = \{x_1, x_2, \dots, x_k, \dots, x_p\}$, где $k = \overline{1..p}$, p – количество запросов в очереди, x_k – булева переменная равная 1 если запрос Z_k выбран в этом варианте и 0 если нет, C_k – приоритет запроса Z_k .

Для описания суммы приоритетов выбранных запросов используем функционал

$$F = \sum_{k=1}^p C_k x_k \rightarrow \max \quad (1)$$

Пусть A_{kg} – булева переменная равная 1, если Z_k использует таблицу T_g , и 0 – если нет. B_g – количество копий таблицы T_g . Тогда, исходя из условия, что в любой момент времени любая таблица может быть использована для одного запроса, получаем M ограничений вида

$$\sum_{k=1}^p A_{kg} x_k \leq B_g, g = \overline{1..M} \quad (2)$$

Следовательно, необходимо найти такую выборку \bar{X} из множества $\{\bar{X}\}$, для которой функционал (1) примет максимальное значение при выполнении всех ограничений (2). Таким образом, получили задачу линейного программирования с булевыми переменными.

Разрешение очереди запросов при такой формализации происходит поэтапно. Каждый этап состоит из нахождения оптимальной выборки \bar{X} , ее обслуживания и изменения функционала (1) и ограничений (2) с учетом изменений в очереди после обслуживания выборки.

Рассмотрим метод групповой выборки на следующей задаче:

Имеется очередь из 7 запросов, каждый из которых имеет свой приоритет и требует информацию из определенных таблиц. Эти исходные приведены в табл. 1.

Составим табл. 2 с обратными исходными данными по данным табл. 1, в которой покажем для каждой таблицы, какие запросы обращаются к ее данным.

1 этап. Запишем функционал (1) подставив значения из табл. 1.

$$F = z_1 + 3z_2 + 2z_3 + 2z_4 + 4z_5 + z_6 + z_7 \rightarrow \max.$$

Ограничения (2) примут вид:

$$z_1 + z_3 + z_5 \leq 1; z_1 + z_7 \leq 1; z_3 + z_4 + z_7 \leq 1. \quad (3)$$

Таблица 1

Таблица 2

Запросы	z_1	z_2	z_3	z_4	z_5	z_6	z_7
Приоритеты	1	3	2	2	4	1	1
Таблицы	$T_1 T_3$	T_2	$T_1 T_4$	T_4	T_1	T_5	$T_3 T_4$

Таблицы	T_1	T_2	T_3	T_4	T_5
Запросы	$z_1 z_3 z_5$	z_2	$z_1 z_7$	$z_3 z_4 z_7$	z_6

Ограничение (3) составлено для таблицы T_1, T_3, T_4 соответственно.

После решения полученной задачи линейного программирования с булевыми переменными получаем, что на первом этапе могут быть решены запросы z_2, z_4, z_5, z_6 .

2 этап. Запишем функционал (1) с учетом результатов первого этапа:

$$F = z_1 + z_3 + z_7 \rightarrow \max. \quad (4)$$

Ограничения (2) примут вид:

$$z_1 + z_3 \leq 1; z_1 + z_7 \leq 1; z_3 + z_7 \leq 1. \quad (5)$$

По результатам решения данной задачи получаем, что необходимо обслужить запрос z_3 .

3 этап. Аналогично получаем, что на третьем этапе необходимо обслужить запрос z_1 .

4 этап. На четвертом – z_7 .

Следовательно, этим вариантом очередь разрешится за четыре этапа.

Метод групповой выборки с индивидуальной сегментацией – это такой метод, при реализации которого запросы, находящиеся в очереди, разбиваются на подзапросы. Из очереди полученных подзапросов выбираются подзапросы, требующие информацию из различных таблиц и чтобы сумма их приоритетов была максимальной. При наличии равнозначных подзапросов выбирают более «старые».

Подзапрос – это часть запроса, которая запрашивает информацию из одной конкретной таблицы. Если запроса требует информацию из R таблиц, то он разбивается на R подзапросов.

В этом варианте нам надо выбрать из очереди как можно большее количество подзапросов, которые обращаются к разным таблицам, и сумма приоритетов выбранных подзапросов была бы максимальной.

Пусть Z_{kg} – подзапрос запроса Z_k , обращающийся к таблице T_g . C_{kg} – приоритет запроса. $Z_{kg} \{ \bar{X} \}$ – множество всех вариантов выбора подзапросов из очереди, \bar{X} – один из вариантов выбора подзапросов. Причем $\bar{X} = \{ x_{11}, x_{12}, \dots, x_{kg}, \dots, x_s \}$, где $k = \overline{1..p}$, $g = \overline{1..M}$, p – количество запросов в очереди, M – количество таблиц, x_{kg} – булева переменная равная 1, если соответствующий подзапрос Z_{kg} выбран в этом варианте, и 0 – если нет, тогда функционал (1) примет вид

$$F(x) = \sum_{j=1}^{p_1} C_{1j} S_1(C_n^1) + \sum_{j=1}^{p_2} C_{2j} S_2(C_n^2) + \dots + \sum_{j=1}^{p_k} C_{kj} S_k(C_n^k) + \dots + \sum_{j=1}^{p_n} C_{nj} S_n(C_n^n) \rightarrow \max, \quad (6)$$

где $S_r(C_n^r) = S_1 + S_2 + \dots + S_{p_r}$ – сумма всех возможных сочетаний произведений переменных, содержащихся в каждом произведении $S_r = X_p X_k \dots X_m$ r различных переменных;

$p_r = \frac{n!}{r!(n-r)!}$; C_{rj} – коэффициенты, стоящие в произведениях S_r содержащих r переменных.

Ограничения (3) примут вид

$$\sum_{k=1}^p A_{kg} x_{kg} \leq B_g, \quad g = \overline{1..M} \quad (7)$$

Получаем задачу нелинейного программирования с булевыми переменными. Метод разрешения очереди запросов при таком методе отличается только тем, что для решения соответствующей задачи линейного или нелинейного программирования с булевыми переменными используются разные методы.

Для рассмотренного ранее примера, функционал (6) примет вид

$$F = Z_{11} + Z_{13} + 3Z_{21} + 2Z_{31} + 2Z_{34} + 2Z_4 + 4Z_5 + Z_6 + Z_{73} + Z_{74} + Z_{11}Z_{13} + 2Z_{31}Z_{34} + Z_{73}Z_{74} \rightarrow \max,$$

а ограничения (3) и учетом (7) примут вид

$$Z_{11} + Z_{31} + Z_5 \leq 1; \quad Z_{13} + Z_{73} \leq 1; \quad Z_{34} + Z_4 + Z_{74} \leq 1.$$

И при методе групповой выборки с индивидуальной сегментацией эта очередь разрешится за три этапа.

В результате проведенного анализа необходимо отметить:

1) использование метода групповой выборки с индивидуальной сегментацией позволяет сократить количество этапов, за которые разрешается очередь. Это позволяет уменьшить частоту отказов в обслуживании запросов на входе СУБД, при пиковой нагрузке;

2) для обеспечения эффективной работы предложенных методов необходимо в качестве математического аппарата использовать методы решения задач линейного и нелинейного программирования с булевыми переменными обладающие малой временной сложностью и погрешностью.

Формализация решения задачи нелинейного и линейного программирования с булевыми переменными

Обозначим через H множество всех функций, которое можно породить на основе $F(x)$, полагая равными нулю различные сочетания C_{ij} в (6). Множество H является полным в том смысле, что содержит в себе все возможные нелинейности, состоящие из всех возможных сочетаний переменных, образующих эти нелинейности, которые можно постро-

ить на основе данного подмножества переменных $\{X_1, X_2, \dots, X_n\}$. Мощность данного множества очень велика, но конечна и равна 2^{P_Σ} , где

$$P_\Sigma = 1 + \frac{1}{2} \left[\frac{n!}{1!(n-1)!} \left(\frac{n!}{1!(n-1)!} + 1 \right) + \frac{n!}{2!(n-2)!} \left(\frac{n!}{2!(n-2)!} + 1 \right) + \frac{n!}{k!(n-k)!} \left(\frac{n!}{k!(n-k)!} + 1 \right) + \dots + \frac{n!}{(n-1)!!} \left(\frac{n!}{(n-1)!!} + 1 \right) \right].$$

Следует отметить, что с помощью соотношения (6) может быть определен класс задач дискретной оптимизации, в которых решение определяется только сочетанием переменных и не зависит от перестановки переменных в $S_r(C_n^r)$. То есть значение C_{ij} в этих задачах зависит только от сочетания переменных в $S_r(C_n^r)$. В общем случае задачу булевого программирования можно представить в виде

$$f(X_1, X_2, \dots, X_n) \Rightarrow \max; g_j(X_1, X_2, \dots, X_n) \leq b_j; \quad j = \overline{(1, m)},$$

где

$$f(X_1, X_2, \dots, X_n) \in H; \quad g_j(X_1, X_2, \dots, X_n) \in H; \quad b_j \in Z; \quad Z - \text{множество целых чисел}; \quad X_i \in \{0, 1\}.$$

Рассмотрим граф $G(X, E)$ рис. 3, в котором вершины X_i и X_j соединены ребром (i, j) , если они могут быть объединены в клику. В графе $G(X, E)$ каждой вершине X_i соответствует переменная X_i .

Выделим в графе G произвольную клику $Q = X_p X_r \dots X_m$, состоящую из r вершин, где $r < n$, и рассмотрим ее пересечения с $S_r(C_n^r) \in f(X_1, X_2, \dots, X_n)$, а также с $S_r(C_n^r) \in g_j(X_1, X_2, \dots, X_n)$. Каждое пересечение можно охарактеризовать суммами коэффициентов C_{ij} , стоящими при $S_r(C_n^r)$ в функционале $f(X_1, X_2, \dots, X_n)$ и ограничениях $g_j(X_1, X_2, \dots, X_n)$, при этом в общем случае произвольная клика Q всегда будет характеризоваться соответствующим весом по функционалу $f(X_1, X_2, \dots, X_n)$ и не более чем m весами по ограничениям $g_j(X_1, X_2, \dots, X_n)$. Таким образом, произвольная задача булевого программирования может рассматриваться как задача нахождения клики Q^* максимального веса по весам функционала, в графе G , у которой все m весов по весам ограничений не превышают соответственно b_j . Если решается задача линейного программирования

$$\begin{aligned} f(x) &= C_1 X_1 + C_2 X_2 + C_3 X_3 + C_4 X_4 \rightarrow \max; \\ B_1 X_1 + B_2 X_2 + B_3 X_3 + B_4 X_4 &< b_1; \\ K_1 X_1 + K_2 X_2 + K_3 X_3 + K_4 X_4 &\leq b_2, \end{aligned} \quad (8)$$

то граф G для задачи линейного программирования будет иметь вид (рис.4).

Как видно из табл. 3, каждая вершина графа G характеризуется тремя весами (C_i, B_i, K_i) . Для решения задачи в графе G (рис.4) следует найти клику Q^* из взвешенных вершин, такую, чтобы ее суммарный вес по весам $\{C_i\}$ был максимален и при этом суммарные веса по весам $\{B_i\}$ и $\{K_i\}$ не превышали соответственно величин b_1 и b_2 . Следует отме-

титель, что в задачах линейного программирования и с нелинейностью, выше второй (т.е. при наличии в функции цели или ограничениях произведений, состоящих из числа переменных, более двух), весовые характеристики ребер графа G полагаются равными нулю. В случае решения задач квадратичного программирования удобно вводить и весовые характеристики ребер.

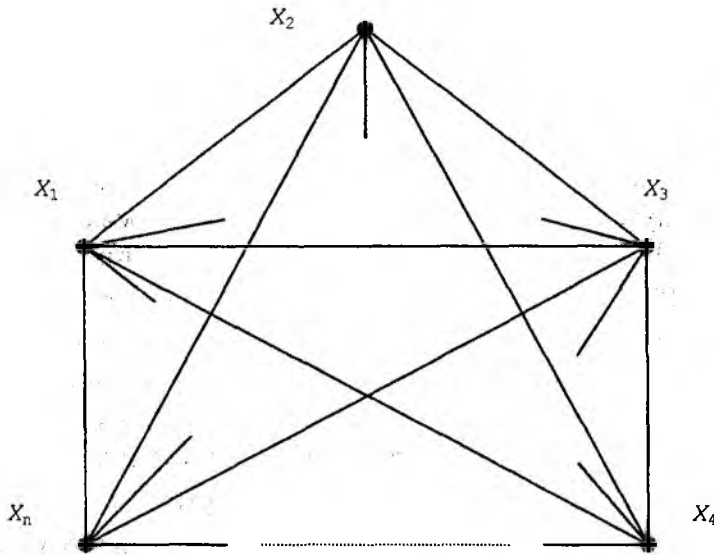


Рис. 3. Граф G

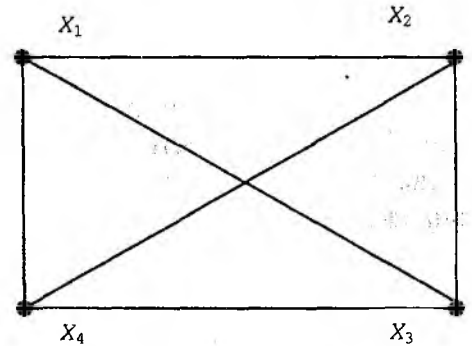


Рис.4. Граф G

Рассмотрим задачу квадратичного программирования следующего вида:

$$f(x) = C_1X_1 + C_2X_2 + C_3X_3 + C_4X_4 + C_{12}X_1X_2 + C_{13}X_1X_3 + C_{34}X_1X_4 + C_{23}X_2X_3 + C_{24}X_2X_4 + C_{34}X_3X_4 \rightarrow \max;$$

$$B_1X_1 + B_2X_2 + B_3X_3 + B_4X_4 < b_1; K_1X_1 + K_2X_2 + K_3X_3 + K_4X_4 \leq b_2; \quad (9)$$

Для нее можно поставить в соответствие следующий граф G для задачи квадратичного программирования (рис.5).

Как видно из рис.5 и табл.4, вершинам графа, как и в задаче линейного программирования, соответствуют веса (C_i, B_i, K_i) , а ребрам веса $\{C_{ij}\}$. Для решения данной задачи в графе G требуется найти клику Q^* максимального суммарного веса по C_i и C_{ij} и при этом необходимо, чтобы суммарные веса по весам $\{B_i\}$ и $\{K_i\}$ не превышали соответственно величин b_1 и b_2 . В этом случае граф G является взвешенным и по вершинам, и по ребрам, аналогично взвешенными являются и все клики данного графа. Следует иметь в виду, что в квадратичной задаче $C_i = C_{ij}$. Таким образом, для решения произвольной задачи булевого программирования необходимо построить алгоритм, позволяющий находить в заданном графе G , взвешенном по вершинам или по ребрам, клику Q^* максимального суммарного веса по весам функционала $f(X_1, X_2, \dots, X_n)$, у которой все m весов по весам ограничений $g_j(X_1, X_2, \dots, X_n)$ не превышают соответствующих b_j .

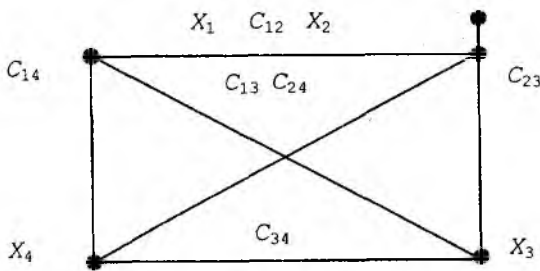


Рис. 5

Таблица 3

X_1	C_1	B_1	K_1
X_2	C_2	B_2	K_2
X_3	C_3	B_3	K_3
X_4	C_4	B_4	K_4

Таблица 4

X_1	C_1	B_1	K_1
X_2	C_2	B_2	K_2
X_3	C_3	B_3	K_3
X_4	C_4	B_4	K_4

Решение задачи. Для решения задачи воспользуемся представлением исходного графа G (рис. 5) в виде симметричного дерева путей предложенного в работе [14]. Смысл такого представления заключается в следующем. Пусть все возможные состояния некоторой системы определяются графом $G(V, E)$ с n вершинами, где вершины соответствуют возможным состояниям системы. Перейдем к пространству с $(n-1)^2$ состояниями. Для этого каждому из n состояний поставим в соответствие еще $(n-1)$ состояние, характеризующее способ достижения состояния из множества $\{1, 2, \dots, n\}$. При этом в качестве добавляемых состояний определим ранг пути в графе $G(V, E)$. Т.е. из вершины s графа $G(V, E)$ в произвольную вершину j можно попасть путем ранга $r=1$, используя одно ребро, путем ранга $r=2$, используя 2 ребра и т.д. путем ранга $r=n-1$, используя $n-1$ ребро. Такое пространство состояний можно представить в виде стянутого дерева путей D графа $G(V, E)$, графически оно может быть изображено следующим образом (рис.6).

Дерево всех путей D содержит $(n-1)$ горизонтальную линейку и $(n-1)$ ярус. Для прочтения путей на каждой горизонтальной линейке можно бывать только один раз. Исходя из стянутого дерева путей, для произвольной вершины j множество путей, ведущих в эту вершину из некоторой вершины s , можно представить в следующем виде

$$m_s(j) = m_{sj}^{r=1} \cup m_{sj}^{r=2} \cup \dots \cup m_{sj}^{r=n-1}; j = \overline{(1, n-1)}, \quad (10)$$

где $m_{sj}^r = \{\mu_{sj}^r\}$ – подмножества путей из произвольной вершины s в некоторую вершину j графа $G(V, E)$, ранга r . Следует отметить, что дерево всех путей D может строиться и от конкретной вершины i графа, в этом случае вершина $s=i$ и i -я горизонтальная линейка исключается в D . Например, при $i=2$ дерево D будет иметь вид рис.7. В дальнейшем стянутое дерево путей, приведенное на рис.6, будет использоваться для построения однопроходных алгоритмов решения задачи, стянутое дерево всех путей D графа от вершины $s=2$ $G(V, E)$ (рис. 7) – для построения n -проходных алгоритмов.

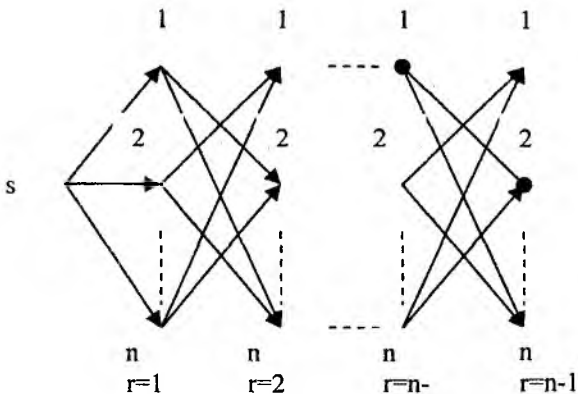


Рис. 6

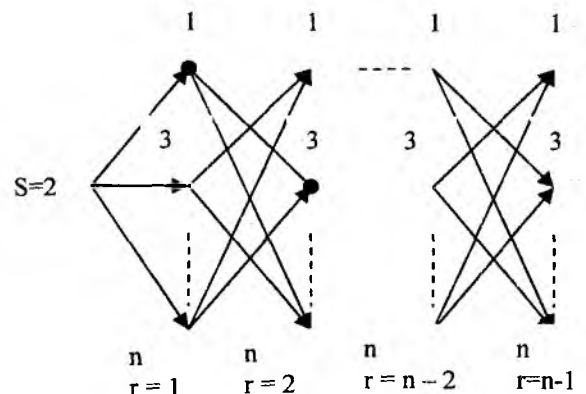


Рис. 7

Таким образом, используя граф D и введя правила формирования путей следующего ранга, можем из произвольной вершины s поэтапно строить пути $\{\mu_{sj}^r\}$ произвольного ранга вплоть до ранга $r = n - 1$. В нашей задаче под состоянием системы будем подразумевать различные способы объединения вершин графа D в клики. Тогда каждому пути $\{\mu_{sj}^r\}$ ранга r в графе D , проходящем через вершины (v_h, v_k, \dots, v_p) в исходном графе G решаемой задачи, соответствует клика из $r - 1$ вершины $(X_h X_k \dots X_p)$, характеризующаяся соответствующим весом по функционалу $f(X_1, X_2, \dots, X_n)$ и не более чем m весами по ограничениям $g_j(X_1, X_2, \dots, X_n)$. Весовые характеристики $\{d_{sj}^{r-1}\}$ произвольной клики $Q^{r-1}(j)$, состоящей из $r - 1$ вершины и определяемой одним из путей $\mu_{sj}^r \in m_{sj}^r$ ранга r в графе D , вычисляются по весам функционала путем суммирования коэффициентов подмножества $L_f = \{C_{rj}\}$, стоящих при $S_{r-1}(C_n^{r-1}) \in P_f$, где P_f – все подмножества $\{S_{r-1}(C_n^{r-1})\}_f$, удовлетворяющие условию

$$S_{r-1}(C_n^{r-1})_f \cap Q^{r-1}(j) \neq \emptyset,$$

а $S_{r-1}(C_n^{r-1})_f$ определяется функционалом $f(X_1, X_2, \dots, X_n)$. Аналогично определяются весовые характеристики по весам ограничений путем суммирования коэффициентов подмножества $L_B = \{C_{rj}\}$, стоящих при $S_{r-1}(C_n^{r-1}) \in P_B$, где P_B – все подмножества $\{S_{r-1}(C_n^{r-1})\}_B$, удовлетворяющие условию

$$S_{r-1}(C_n^{r-1})_B \cap Q^{r-1}(j) \neq \emptyset,$$

а $S_{r-1}(C_n^{r-1})_B$ определяется ограничениями $g_j(X_1, X_2, \dots, X_n); j = \overline{(1, m)}$. Таким образом, весовые характеристики клик $Q^{r-1}(j)$, характеризующихся множеством путей m_{sj}^r по весам функционала и ограничений, определяются соответственно

$$d_{sj} f(r-1) = \sum_{C_{rj} \in L_f} C_{rj}; \quad d_{sj} B(r-1) = \sum_{C_{rj} \in L_B} C_{rj}. \quad (11)$$

Итак, в графе D каждый путь имеет в общем случае $m + 1$ длину, одну по весам функционала и m – по весам ограничений, и для решения поставленной задачи нам в графе D нужно построить путь максимальной длины по весам функционала от вершин $1, 2, \dots, n$ ко все остальным вершинам графа; при этом его длины по весам ограничений не должны превышать соответствующей величины b_j . Если на основе подмножеств путей $m_{sj}^{r=1}$ в графе D строить подмножества $m_{sj}^{r=2}$ и так далее до $m_{sj}^{r=n-1}$, то вынуждены будем построить $(n-1)!$ путей, поэтому для формирования путей вводится процедура A , позволяющая отсекалть неперспективные пути. Для отсекалния неперспективных вариантов в процедуре A предлагается использовать принцип оптимизации по направлению к произвольной вершине p , при формировании путей следующего ранга m_{sp}^{r+1} на основе путей предыдущего ранга m_{sj}^r , предложенных в работах [11 – 13], который для рассматриваемой задачи определяется следующим рекуррентным соотношением

$$\mu_{sp}^{r+1} = \max\{\{\mu_{sj}^r\} \cup (j, p)\}; j = \overline{(1, n)}; p = \overline{(1, n)}; j \neq p \quad (12)$$

где (j, p) – ребро графа D ; n – число различных вершин в графе D .

Рассмотрим возможность построения n – проходных и однопроходных процедур решения задачи (7) соответственно на стянутых деревьях приведенных на рис.6 и 7.

Процедура A_1 с n проходами.

Перед началом работы процедуры A_1 переменной $i := 1$.

Шаг 1. Переменной $s := i$ и из вершины s строятся все возможные пути ранга $r = 1$ ко всем вершинам графа D (рис.7), удовлетворяющие ограничениям $g_j(X_1, X_2, \dots, X_n); j = \overline{(1, m)}$, при этом длины по весам функционала и ограничениями вычисляются в соответствии с соотношениями (11).

Шаг 2. С использованием путей текущего ранга r строятся все возможные пути ранга $r := r + 1$, удовлетворяющие ограничениям $g_j(X_1, X_2, \dots, X_n); j = \overline{(1, m)}$ с использованием рекуррентного соотношения (12). При этом проверка ограничений и выбор пути максимального по весам функционала осуществляются на основе вычислений длин путей по весам функционала и ограничений в соответствии с соотношениями (11). (Следует отметить, что если в процессе применения рекуррентного соотношения (12) возникают несколько путей одинаковой длины, то необходимо их продлевать на следующем ранге.)

Шаг 3. Проверяем $m_{sj}^{r+1} = \emptyset$, если да, то путь μ_{sj}^{*r} максимальной длины, полученный на ранге r , является локальным экстремумом решаемой задачи, иначе выполняем следующий шаг.

Шаг 4. Проверяем $i := n - 1$, если нет, то $i := i + 1$, и переходим к выполнению шага 1, иначе процедура A_1 заканчивает работу, при этом из множества локальных экстремумов $\{\mu_{sj}^{*r}\}$ выбирается глобальный μ_{sj}^{**r} , соответствующий оптимальному решению задачи (7).

Для снижения временной сложности работы алгоритма возможно использовать однопроходный вариант реализации данной процедуры на основе стянутого дерева путей приведенного на рис.6. При этом однопроходная процедура A_2 имеет такой вид.

Процедура A_2 с 1 проходом.

Шаг 1. Из вершины s строятся все возможные пути ранга $r = 1$ ко всем вершинам графа D (рис.6), удовлетворяющие ограничениям $g_j(X_1, X_2, \dots, X_n); j = \overline{(1, m)}$, при этом длины по весам функционала и ограничениям вычисляются в соответствии с соотношениями (11).

Шаг 2. Используя пути текущего ранга r , строят все возможные пути ранга $r := r + 1$ удовлетворяющие ограничениям $g_j(X_1, X_2, \dots, X_n); j = \overline{(1, m)}$ с использованием рекуррентного соотношения (12). При этом проверка ограничений и выбор пути максимального по весам функционала осуществляются на основе вычислений длин путей по весам функционала и ограничений в соответствии с соотношениями (11). (Следует отметить, что если в процессе применения рекуррентного соотношения (12) возникают несколько путей одинаковой длины то необходимо их все продлевать на следующем ранге.)

Шаг 3. Проверяем $m_{sj}^{r+1} = \emptyset$, если да, то путь μ_{sj}^{*r} максимальной длины, полученный на ранге r , является локальным экстремумом решаемой задачи, иначе выполняем следующий шаг.

Шаг 4. Проверяем ранг $r = n$, если нет, то переходим к выполнению шага 2, иначе процедура A_2 заканчивает работу, и путь μ_{sj}^{*r} максимальной длины, полученный на ранге $r = n$ соответствует оптимальному решению задачи (7).

Еще одним вариантом уменьшения временной сложности алгоритмов на основе процедур A_1 и A_2 могут являться процедуры A' и A'' , отличающихся от A_1 и A_2 тем, что на каждом ярусе формирования путей на основе процедур A_1 и A_2 локальные экстремумы будут выделяться не в каждом множестве, а выделяются глобальные экстремумы на ярусе и на основе пути, соответствующего глобальному экстремуму на ярусе, формируются пути следующего яруса, удовлетворяющие ограничениям, при этом процедуры A' и A'' будут иметь вид.

Процедура A' . Перед началом работы процедуры A' переменной $i := 1$.

Шаг 1. Переменной $s := i$ и из вершины s строятся все возможные пути ранга $r = 1$ ко всем вершинам графа D (рис.7), удовлетворяющие ограничениям $g_j(X_1, X_2, \dots, X_n)$; $j = \overline{(1, m)}$, при этом длины по весам функционала и ограничениям вычисляются в соответствии с соотношениями (11). Далее выделяется самый длинный путь на ярусе.

Шаг 2. Используя самый длинный путь текущего ранга r построенный на предыдущем шаге, строятся все возможные пути ранга $r := r + 1$ удовлетворяющие ограничениям $g_j(X_1, X_2, \dots, X_n)$; $j = \overline{(1, m)}$ с использованием рекуррентного соотношения (12). При этом проверка ограничений и выбор пути максимального по весам функционала осуществляется на основе вычислений длин путей по весам функционала и ограничений в соответствии с соотношениями (11).

Шаг 3. Проверяем $m_{sj}^{r+1} = \emptyset$, если да, то путь μ_{sj}^{*r} максимальной длины, полученный на ранге r , является локальным экстремумом решаемой задачи, иначе выполняем следующий шаг.

Шаг 4. Проверяем $i := n - 1$, если нет, то $i := i + 1$ и переходим к выполнению шага 1, иначе процедура A' заканчивает работу, при этом из множества локальных экстремумов (полученных за один проход) $\{\mu_{sj}^{*r}\}$ выбирается глобальный μ_{sj}^{**r} , соответствующий оптимальному решению задачи (7).

Процедура A'' .

Шаг 1. Из вершины s строятся все возможные пути ранга $r = 1$ ко всем вершинам графа D (рис.6), удовлетворяющие ограничениям $g_j(X_1, X_2, \dots, X_n)$; $j = \overline{(1, m)}$, при этом длины по весам функционала и ограничениям вычисляются в соответствии с соотношениями (11). Далее выделяется самый длинный путь на ярусе.

Шаг 2. Используя самый длинный путь текущего ранга r построенный на предыдущем шаге, строятся все возможные пути ранга $r := r + 1$, удовлетворяющие ограничениям $g_j(X_1, X_2, \dots, X_n)$; $j = \overline{(1, m)}$ с использованием рекуррентного соотношения (12). При этом проверка ограничений и выбор пути максимального по весам функционала осуществляется на основе вычислений длин путей по весам функционала и ограничений в соответствии с соотношениями (11).

Шаг 3. Проверяем $m_{sj}^{r+1} = \emptyset$, если да, то путь μ_{sj}^{*r} максимальной длины, полученный на ранге r , является локальным экстремумом решаемой задачи, иначе выполняем следующий шаг.

Шаг 4. Проверяем ранг $r = n$, если нет, то переходим к выполнению шага 2, иначе процедура A'' заканчивает работу, и путь μ_{sj}^{*r} максимальной длины, полученный на ранге $r = n$ соответствует оптимальному решению задачи (7).

Оценка сложности процедур $\{A\}$

Поскольку число путей, строящихся на произвольном ранге r , не может превысить $(n-1)(n-1)$, максимальный ранг r произвольного пути не превышает $(n-1)$, а число циклов выполняемое процедурой A_1 , равно n , то после n циклов число путей, которое построит процедура A_1 , не может превзойти $(n-1)(n-1)(n-1)n \approx n^4$, а число обработанных векторов n^5 . С учетом число слагаемых (k) в функционале и числа ограничений (m) временная сложность алгоритма не превысит в худшем случае $O(n^5 k(m+1))$. В случае, когда решение задачи осуществляется за один проход процедуры A_2 или за один проход процедуры A'' , но с выделением наиболее длинного пути на ярусе, сложность процедур A_2 и A'' не превысит соответственно $O(n^4 k(m+1)), O(n^3 k(m+1))$. Итак, алгоритмы A_5, A_4, A_3 имеют соответственно временную сложность, не превышающую в худшем случае $O(n^5 k(m+1)), O(n^4 k(m+1)), O(n^3 k(m+1))$.

Таким образом, предложенный подход решения произвольных задач булевого программирования позволяет на основе предложенных алгоритмов решать с единых позиций любые задачи линейного и нелинейного программирования за полиномиальное время с требуемой точностью. Если в соотношении (6) ввести обозначение $f_k(x) = \sum_{j=1}^{P_k} C_{kj} S_k(C_n^k)$, то оно примет вид $F(x) = \sum_k f_k(x)$. В общем случае, если функционал и ограничения представляют собой произвольную нелинейную функцию от $f_k(x)$, то и такая задача нелинейного программирования также может быть эффективно решена с помощью предложенного подхода и при этом будут изменяться только правила весов вершин клик.

Экспериментальное исследование алгоритмов. Исследовались следующие алгоритмы: алгоритм A_5 на основе многопроходной процедуры A_1 , алгоритм A_4 на основе однопроходной процедуры A_2 и алгоритм A_3 на основе однопроходной процедуры A'' . При исследовании коэффициенты в функционале и ограничениях генерировались по равномерному закону распределения в функционале в диапазоне от 0 до 10, а в ограничениях от 0 до 20. На каждую точку при оценке временной сложности алгоритмов в среднем и погрешности алгоритмов решалось не менее 50 тестовых задач, результаты получены с доверительной вероятностью 0,95. В качестве точного алгоритма использовался разработанный алгоритм для задачи квадратичного и линейного программирования, скомбинированного на основе использования для прогнозирования идей рангового подхода, а для отсева неперспективных путей метода ветвей и границ, позволившего снять погрешности для задач до размерности, не превышающей $n = 70$. Погрешность алгоритмов с увеличением числа ограничений m асимптотически уменьшается, с увеличением n возрастает и $m \geq 50$, погрешность алгоритмов стабилизируется и для задач линейного программирования не превышает 2%, а для задач квадратичного 5–10%.

Решение тестовых задач показало, что увеличение диапазона изменения коэффициентов в функционале и ограничениях приводит к резкому снижению погрешностей алгоритмов, переход от нелинейностей одного порядка к нелинейностям более высокого порядка может

приводить к незначительному возрастанию погрешностей при небольшом числе ограничений, а с возрастанием числа ограничений возрастание погрешности очень быстро компенсируется. Экспериментальное исследование временной сложности показало, что число обрабатываемых векторов не зависит от числа ограничений и в среднем для алгоритмов A_5, A_4, A_3 временная сложность не превышает соответственно

$$O(0,1n^{4,9}), O(0,3n^{3,7}), O(0,4n^{2,8}).$$

Заключение

Таким образом, как видно из экспериментальных исследований предложенных алгоритмов, их можно эффективно использовать для планирования обработки запросов в СУБД методами групповой выборки с индивидуальной сегментацией, используемых в GRID системах, где требуется обеспечить такое планирование в масштабе реального времени.

Предлагаемые алгоритмы позволяют решать однообразно как задачи линейного булевого программирования, так нелинейного программирования.

Список литературы: 1. *Мирошник, М.А., Котух, В.Г., Селевко, С.Н.* Отказоустойчивость распределенных телекоммуникационных систем // Радиотехника : Всеукр. межвед. науч.-техн. сб. – 2012. – Вып. 168. – С. 191–195. 2. *Foster, C. Kesselman, S. Tuecke.* The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International J. Supercomputer Applications, 15(3), 2001. 3. *Коваленко, В.Н., Коваленко, Е.И., Корягин, Д.А., Любимский, Э.З., Хухлаев, Е.В., Шорин, О.Н.* Грид-диспетчер: реализация службы диспетчеризации заданий в грид // Гр. междунар. конф: "Распределенные вычисления и Грид-технологии в науке и образовании". – Дубна, 2004. – С. 133-139. 4. *Мирошник, М.А., Котух, В.Г.* Разработка методов довышения отказоустойчивости и надежности функционирования компонентов телекоммуникационных систем и сетей // Радиотехника : Всеукр. межвед. науч.-техн. сб. – 2011. – Вып. 164. – С. 190–197. 5. *Мирошник, М.А.* Подход к проектированию компьютерных систем с интеллектуальной диагностической инфраструктурой / С.Г. Карпенко, М.А. Ковалева, С.В. Панченко // Інформаційно-керуючі системи на залізничному транспорті. – 2011. – №6. – С. 51-59. 6. *Гуль, А.Ю., Никитина, Т.С.* Подходы к организации планирования распределения ресурсов в Grid системах // 36. наук. праць укр. держ. академії залізничного транспорту. – 2010. – Вип. 116. – С.88-97. 7. *Герц Хансен, Джеймс Хансен.* Базы данных: разработка и приложение : пер. с англ. – М. : БИНОМ, 1999. – 704 с. 8. *Дейт, К.* Введение в системы баз данных – 6-е изд. : пер. с англ. – К.; М.; С.-Пб. : Вильямс, 2000. – 848 с. 9. *Мейер, М.* Теория реляционных баз данных. – М. : Мир, 1987. – 608 с. 10. *Хаббард, Дж.* Автоматизированное проектирование баз данных. – М. : Мир, 1984. – 294 с. 11. *Listrovoy, S.V., Golubnichiy, D.Yu., Listrovaya, E.S.* Solution method on the basis of rank approach for integer linear problems with boolean variables // Engineering Simulation. – 1999. – Vol.16. – P. 707–725. 12. *Listrovoy, S.V., Tretjak, V.F., Listrovaya, A.S.* Parallel algorithms of calculation process optimization for the boolean programming problems // Engineering Simulation. – 1999. – Vol.16. – P. 569–579. 13. *Listrovoy, S.V., Gul, Yu.* Method of Minimum Covering Problem Solution on the Basis of Rank Approach // Engineering Simulation. – 1999. – Vol.17. – P. 73–89. 14. *Методы моделирования и дискретной оптимизации вычислительных систем реального времени / Жихарев В.Я., Илюшко В.М., Кравец Л.Г., Листровой С. В., Харченко В.С. / под ред. В.Я. Жихарева. – Харьков; Житомир : ЖГУ, 2004. – 494 с.*

Харьковский национальный
университет радиоэлектроники

Поступила в редколлегию 05.04.2012