ИЗВЕСТИЯ РАН. ТЕОРИЯ И СИСТЕМЫ УПРАВЛЕНИЯ, 2019, № 3, с. 77–86

СИСТЕМНЫЙ АНАЛИЗ И ИССЛЕДОВАНИЕ ОПЕРАЦИЙ

УДК 004.031.43

ПОТОКОВЫЕ АЛГОРИТМЫ ПЛАНИРОВАНИЯ ВЫЧИСЛЕНИЙ В ИНТЕГРИРОВАННОЙ МОДУЛЬНОЙ АВИОНИКЕ[1]

© 2019 г. В. А. Костенко[a],*, А. С. Смирнов[a]

[a]МГУ, Москва, Россия

**e-mail: kost@cs.msu.su*

Поступила в редакцию 26.12.2018 г.
После доработки 25.01.2019 г.
Принята к публикации 28.01.2019 г.

Предложены алгоритмы, основанные на нахождении максимального потока в транспортной сети для планирования выполнения задач в системах реального времени с интегрированной модульной архитектурой. Приведены результаты их экспериментального исследования для однопроцессорного и многопроцессорного вариантов задачи планирования вычислений.

DOI: 10.1134/S0002338819030119

Введение. Для поддержки требований реального времени и изолированности программ различных подсистем информационно-управляющих систем реального времени (ИУС РВ) с интегрированной модульной архитектурой [1] был разработан стандарт ARINC 653 [2]. Наиболее широко используемый подход к построению ИУС РВ с интегрированной модульной архитектурой известен как интегрированная модульная авионика (ИМА). Российской операционной системой, соответствующей стандарту ARINC 653, является ОС РВ Багет 3.0 [3, 4]. В операционной системе реального времени (ОС РВ) Багет 3.0 стандарт ARINC 653 выбран в качестве основного. Реализованы все обязательные функции ARINC 653. Стандарт POSIX используется в той мере, в какой это не противоречит ARINC 653.

Изолированность программ различных подсистем обеспечивается введением разделов и окон. Для программ каждой подсистемы выделяется свой раздел и набор временных окон (непересекающихся интервалов времени). Программы раздела могут выполняться только в рамках своих окон и каждому разделу выделяется необходимая память, к которой не могут обращаться программы других разделов. Программы раздела внутри окна запускаются на выполнение динамически, например, по мере готовности данных в соответствии с приоритетами. Допустимо прерывание программы и ее последующее выполнение в этом окне или в одном из следующих окон раздела. Программы различных разделов могут взаимодействовать лишь путем передачи сообщений, т.е. прикладные программы выполняются в соответствии со статико-динамическим расписанием. Статико-динамическое расписание построено, если определена привязка разделов к процессорам вычислителя, для каждого раздела найден набор окон и для каждого окна – время его открытия и время закрытия.

Различные варианты задачи построения статико-динамических расписаний задаются динамическими планировщиками программ в разделах, способом задания исходных данных и набором технологических ограничений. Примерами технологических ограничений являются максимально допустимый размер окна, минимально необходимый промежуток времени между окнами различных разделов для переключения контекста. Однако понятия раздела и окна являются общими для всех операционных систем, соответствующих стандарту ARINC 653.

В данной работе для построения многопроцессорных статико-динамических расписаний предложены алгоритмы, основанные на нахождении максимального потока в транспортной сети.

1. Близкие задачи и известные алгоритмы. В работах [5, 6] были предложены муравьиные алгоритмы для построения статико-динамических расписаний. Данные алгоритмы применимы

[1] Работа выполнена при финансовой поддержке РФФИ (грант № 17-07-01566).

только для варианта задачи, когда прерывание программ не допустимо. Это означает, что программа может выполняться только в одном окне своего раздела, что существенно ограничивает практическое применение алгоритмов.

В [7, 8] были рассмотрены алгоритмы, основанные на декомпозиции задачи на две подзадачи: привязка разделов к процессорам и построение набора окон для каждого процессора. Недостатком этих алгоритмов является их ориентация на учет специфических для конкретной ИУС РВ ограничений на корректность статико-динамического расписания, а также зависимость их точности от класса исходных данных.

Наиболее близкими задачами, для которых известны алгоритмы, основанные на нахождении максимального потока в транспортной сети, являются задачи построения расписаний с прерываниями работ. Методика использования алгоритмов нахождения максимального потока в сети для построения расписаний с прерываниями была предложена в работах [9, 10].

В [11] был рассмотрен алгоритм построения расписаний для неоднороднородной многопроцессорной системы (процессоры имеют разную производительность). Прерывания и переключения работ не требуют временных затрат. В [12] был описан алгоритм для задачи, в котором учитывался ограниченный объем памяти процессоров. В [13] предложен алгоритм для случая, когда длительности выполнения работ линейно зависят от количества выделенного им ресурса.

Основными проблемами, препятствующими непосредственному применению известных алгоритмов, основанных на нахождении максимального потока в транспортной сети, для построения статико-динамических расписаний являются: учет принадлежности работ к разделам и построение набора окон.

В [14] был рассмотрен алгоритм, основанный на поиске максимального потока в транспортной сети для однопроцессорного варианта задачи построения статико-динамического расписания. Однако для многопроцессорного варианта задачи данный алгоритм может быть использован только совместно с алгоритмом привязки разделов к ядрам. Для достижения высокой точности такой композиции алгоритмов может потребоваться существенная адаптация в алгоритме привязки разделов к ядрам критериев распределения раздела на ядро таким образом, чтобы на каждом ядре получать набор разделов (класс исходных данных), для которого потоковый алгоритм показывает высокую точность.

2. Задача построения статико-динамического расписания. Для задачи построения статико-динамических расписаний для систем реального времени с архитектурой интегрированной модульной авионики задаются следующие исходные данные: n – число работ; p – число процессоров; q – число разделов; c – время на переключение окна; $A = \left\{a_k \middle| k = \overline{1,n}\right\}$ – набор работ.

Требования к выполнению работ в реальном времени могут задаваться двумя способами:

Способ 1: $a_k = \langle s_k^A, f_k^A, t_k^A, d_k^A\rangle$, где s_k^A – директивный срок начала выполнения работы k (выполнение работы должно начаться не раньше этого срока); f_k^A – директивный срок завершения выполнения работы k (выполнение работы должно завершиться до наступления этого срока), t_k^A – время выполнения работы; d_k^A – раздел, к которому относится работа.

Способ 2: $a_k = \langle F_k^A, t_k^A, d_k^A\rangle$, где F_k^A – частота ($1/F_k^A$ – период) выполнения работы; t_k^A – время выполнения работы; d_k^A – раздел, к которому относится работа. Частота определяет набор отрезков времени выполнения работы, длина которых равна ее периоду.

Второй способ задания требований к выполнению работ в реальном времени может быть сведен к первому. Вычисляется большой цикл как наименьшее общее кратное периодов выполнения работ. Количество запусков работы в большом цикле равно количеству ее периодов в большом цикле. Директивные сроки каждого запуска работы определяются временем начала и завершения соответствующего периода.

Требуется построить статико-динамическое расписание, содержащее максимальное число работ из заданного набора. Расписание построено, если определен набор окон для каждого процессора l: $W_l = \{w_i = \langle s_i^{W_l}, f_i^{W_l}, d_i^{W_l}, A_i^{W_l}\rangle i = \overline{1,m_l}\}$, где m_l – число окон; $s_i^{W_l}$ – время открытия окна; $f_i^{W_l}$ – время закрытия окна; $d_i^{W_l}$ – номер раздела, которому принадлежит окно; $A_i^{W_l} = \{(a_j, t_j) | a_j \in A, j \subset \overline{1,n}, t_j \le t_j^A\}$ – набор работ, выполняемых в окне (с указанием времени, отведенном работе на исполнение в данном окне).

Расписание должно удовлетворять условиям корректности для каждого набора окон W_l.

1. Окна не перекрываются и учтено минимальное время переключения:

$$c \le s_{i+1}^{W_l} - f_i^{W_l} \quad \forall i \in \overline{1, m_l - 1}. \tag{2.1}$$

2. Сумма значений длительностей частей работ, размещенных в одном окне, не превышает значения длительности окна:

$$\sum_{a_j \in A_i^{W_l}} t_{ji} \le f_i^{W_l} - s_i^{W_l} \quad \forall i \in \overline{1, m_l}. \tag{2.2}$$

3. В окне могут выполняться только части работы или целые работы из того раздела, к которому это окно привязано:

$$\forall a_j, a_k \in A_i^{W_l} \rightarrow d_j^{W_l} = d_k^{W_l} \quad \forall i \in \overline{1, m_l}. \tag{2.3}$$

4. Работа размещена, если она выполнена полностью:

$$\sum_{a_k \in A_j^{W_l}} t_{kj} = t_k^A \quad \forall a_k \in \bigcup_{i=1}^{m_l} A_i^{W_l}. \tag{2.4}$$

5. Работы одного раздела выполняются только на одном процессоре.

3. Алгоритм построения статико-динамического расписания на основе алгоритма поиска максимального потока в сети. Алгоритм состоит из трех основных этапов:

1) построение транспортной сети (ориентированного графа),

2) нахождение потока в транспортной сети,

3) восстановление расписания из полученного потока.

В соответствии с директивными сроками работ (время начала и время завершения директивного интервала) строится упорядоченный по времени открытия набор непересекающихся временных интервалов. Работа может выполняться только в интервалах, которые пересекаются с ее директивным интервалом. Построение транспортной сети осуществляется выделением вершин-работ, выделением вершин-интервалов-процессоров, связаных с номером процессора, добавлением стока и истока. Вершина исток соединяется с вершинами-работами ребрами пропускной способности, равной времени выполнения данной работы. Вершины-работы соединяются с вершинами-интервалами-процессорами, на которых соответствующая работа доступна для выполнения, ребрами пропускной способности, равной длительности данного интервала. Вершины-интервалы соединяются с вершиной стоком пропускными способностями, равными длительности соответствующего интервала.

Поиск потока осуществляется на основе алгоритма “поднять-и-в-начало” как наиболее простого по вычислительной сложности и организации проверки корректности расписания. В нем определены три основные операции:

1) подъем вершины,

2) проталкивание из вершины в вершину,

3) разрядка вершины.

Алгоритм “поднять-и-в-начало” существенно модифицирован, так как в транспортной сети из-за необходимости учитывать время переключения между окнами возможны изменения пропускных способностей ребер. Операция проталкивания учитывает, поток какого раздела был направлен в вершину-интервал-процессор или был удален из нее, а также его количество. На основе этих данных определяется: требуется ли выделить время на переключение между окнами в этой вершине, т.е. изменить пропускную способность ребра, ведущего от вершины-интервала-процессора до стока. Подъем находит минимальную по высоте вершину, такую, что существует предпоток в эту вершину и он меньше пропускной способности этой дуги, и делает высоту исходной вершины на единицу больше найденной. Высота определяет порядок применения операции проталкивания. Для получения корректного статико-динамического расписания операции разрядки вершин упорядочиваются и имеют два режима работы. Первый режим – с просмотром возможности проталкивания полученного потока в сток, второй – без этой возможности. Первое проталкивание из вершины-работы в вершину-интервал-процессор является размещением раздела на данном процессоре и все пропускные способности дуг в другие процессоры становятся равными нулю. При невозможности разместить работу раздела требует-

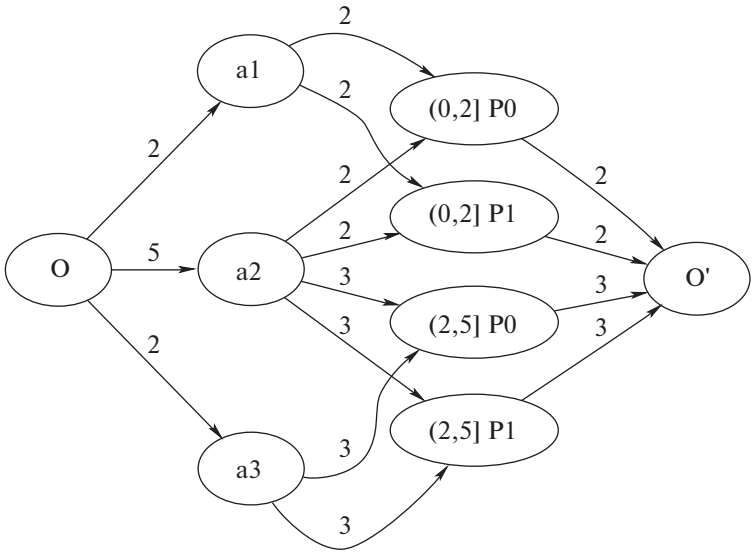

Рисунок. Транспортная сеть

ся вернуть поток от всех работ данного раздела и сделать пропускные способности всех дуг, ведущих от этих вершин-работ в вершины-интервалы-процессоры данного процессора, равными нулю.

Вершины разбиты на группы – вершины-работы каждого раздела и вершины-интервалы-процессоры, всего $q+1$ группа. Пока есть переполненные вершины, алгоритм производит операцию разрядки вершин. Порядок разрядки следующий: сначала выбирается первый раздел, выполняется разрядка всех его вершин с просмотром возможности проталкивания потока в сток, после этого выполняется разрядка вершин-интервалов. Пока не останется переполненных вершин группы этого раздела и группы вершин-интервалов разрядка осуществляется по следующему правилу: разряжается вершина-работа, если это вызвало появление переполненных вершин-интервалов, то разряжаются они, а потом опять разряжаются вершины-работы раздела.

Восстановление расписания заключается в рассмотрении вершин-интервалов-процессоров по порядку и анализу входящих в них потоков от работ разных разделов и числа переключений окон в вершине-интервале-процессоре.

Рассмотрим каждый из этапов алгоритма подробнее.

3.1. Построение транспортной сети. Пусть $\tau_0 < \ldots < \tau_s$ – все различные значения s_k^A и f_k^A, $k = 1, \ldots, n$, и $I_i = (\tau_{i-1};\ \tau_i]$, $i = \overline{1,s}$. Работа k доступна для выполнения (выполнима) в момент времени t_0, если $s_k^A < t_0 < f_k^A$.

Сеть представляет собой двудольный граф с дополнительными вершинами: источником O и стоком O' (рисунок). Первый уровень графа состоит из вершин, каждая из которых взаимно однозначно соответствует какой-либо работе (вершины-работы), второй уровень состоит из вершин, соответствующих паре интервал-процессор (вершины-интервалы-процессоры). Источник соединен со всеми вершинами-работами, причем дуга, ведущая к вершине k-й работы, имеет пропускную способность, равную t_k^A, $k = \overline{1,n}$. Вершина-работа соединяется со всеми вершинами-интервалами-процессорами, на которых эта работа выполнима дугами пропускной способности t^A. Все вершины-интервалы-процессоры соединены со стоком O' дугами пропускной способности, равной длине интервала.

3.2. Поиск потока в транспортной сети. Для построения статико-динамического расписания требуются следующие данные: *dist* – вершина-сток; s – вершина-источник; пропускные способности $c(u,v)$ всех дуг в сети; функция предпотока $f(u,v)$ для всех дуг в сети на шаге алгоритма. Каждая вершина имеет целочисленные параметры: избыточный поток e_u; высота h_u; список вершин-работ по разделам; вершины-работы также имеют номер раздела pt_u, к ко-

торому эта работа прикреплена. Вершины-интервалы-процессоры знают предшествующий и последующий интервалы ni_u, pi_u; имеют целочисленный параметр cw_u – количество переключений окон в интервале, параметры ir_u и il_u показывают, есть ли переключение окна на правой и левой границах соответственно. Вершины-интервалы содержат структуру для запоминания входящих потоков по разделам PT_u^i, где $i=\overline{1,q}$, (т.е. можно узнать, сколько времени отведено в данном интервале данному разделу), множество разделов PS_u, потоки которых входят в вершину, первый fp_u и последний lp_u разделы окна, длительность интервала dur_u. Также для всех вершин-интервалов v, $proc_v$ – номер процессора, с которым ассоциирован данный интервал времени, и максимальное число попыток перераспределений процессоров K. Привязка каждого раздела к ядру есть PR_{part}.

Рассмотрим алгоритмы выполнения основных операции для поиска потока в транспортной сети.

Инициализация сети.

1. Изначально предпоток равен пропускной способности для всех ребер, выходящих из источника, и противоположен для обратных пар вершин:

$$f(s,u) = c(s,u),$$

$$f(u,s) = -c(s,u),$$

$$e_u = f(s,u).$$

2. Для остальных пар вершин предпоток равен нулю.

3. Начальная высота равна 10 в сети для источника, 1 – для вершин-работ, 0 – для вершин-интервалов и стока.

Подъем вершины (u).

Из всех доступных вершин u (существует дуга (u,v) и $f(u,v) < c(u,v)$) выбирается вершина v (v отлична от $dist$) с минимальной высотой, после чего высота вершины u становится равной $h_v + 1$.

Добавление переключения окна в (v).

В этой операции v является вершиной-интервалом-процессором.

1. Пропускная способность $c(v,dist)$ уменьшается на c.

2. Число переключений окон cw_v увеличивается на 1.

3. Если $f(v,dist) > c(v,dist)$, то изменяется избыточный поток.

4. Переполненный поток $e_v = e_v + f(v,dist) - c(v,dist)$, и поток $f(v,dist)$ становится равным $c(v,dist)$.

5. Если $e_v > 0$, то вершина v добавляется к списку переполненных вершин.

Удаление переключения окна в (v).

В данной операции v является вершиной-интервалом.

1. Пропускная способность $c(v,dist)$ увеличивается на c.

2. Если $e_v \geq c$, то

$$f(v,dist) = f(v,dist) + c,$$

$$f(dist,v) = -f(v,dist),$$

$$e_v = e_v - c,$$

$$e_{dist} = e_{dist} + c.$$

3. Если $e_v = 0$, то убрать вершину из списка переполненных вершин.

4. Число переключений окон cw_v уменьшается на 1.

Учет раздела при добавлении потока размера *value* раздела *part* из вершины-интервала v.

1. Добавляется поток в структуру входящих потоков вершины:

$$PT_{part} = PT_{part} + value.$$

2. Если поток этого раздела впервые вошел в вершину-интервал-процессор, то он добавляется в множество PS_v.

2.1. Если это самый первый раздел, то $fp_v = part$ и $lp_v = part$.

2.2. Если это второй раздел, то он сравнивается с первым разделом следующего интервала:

2.2.1. если разделы равны, то $lp_v = part$;

2.2.2. если разделы не равны, то $fp_v = part$.

2.3. Если это третий и более раздел, то он сравнивается с первым разделом следующего интервала:

2.3.1. если разделы равны, то $lp_v = part$;

2.3.2. если нет, то он сравнивается с первым разделом следующего интервала, и если они равны, то $fp_v = part$.

Учет раздела при удалении потока размера *value* раздела *part* из вершины-интервала-процессора v.

1. Убирается поток из структуры входящих потоков вершины:

$$PT_{part} = PT_{part} - value.$$

2. Если поток этого раздела равен нулю (равносильно тому, что этот раздел больше не входит в данный интервал), то он убирается из множества PS_v.

3. Если не осталось разделов, то $fp_v = 0$ и $lp_v = 0$.

4. Если остался один раздел pt, то $fp_v = pt$ и $lp_v = pt$.

5. Если осталось более одного раздела, то:

5.1. если удаленный раздел равен последнему разделу в интервале, то (pt_{nf} – раздел, не равный первому разделу в интервале) $lp_v = pt_{nf}$;

5.2. если удаленный раздел равен первому разделу в интервале, то (pt_{nl} – раздел, не равный последнему разделу в интервале) $fp_v = pt_{nl}$.

Корректировка окон в вершине-интервале-процессоре (v).

1. Производится корректировка внутренних переключений. Вычисляется число внутренних переключений $cw_{in} = cw_v - ir_v - il_v$. Выполняются необходимое число раз операции удаление переключения окна в (in) или добавление переключения окна в (in), чтобы число учтенных переключений было равно cw_{in}.

2. Производится корректировка правого переключения.

2.1. Если есть справа непустая вершина-интервал-процессор ni, то:

2.1.1. если $fp_{ni} \neq lp_v$ и нет переключений справа в v ($ir_v = false$), и нет переключения слева в ni ($ir_{ni} = false$):

2.1.1.1. если $c(ni,dest) - f(ni,dest) \geq cw$ (в следующем интервале есть место для переключения) и $e_v + f(v,dest) > c(v,dest) - cw$ (в текущем нет), то выполняется операция добавление переключения окна в (ni) и $il_{ni} = true$;

2.1.1.2. иначе выполняется операция добавление переключения окна в (v), $ir_v = true$.

2.1.2. Если $fp_{ni} = lp_v$ и есть переключение справа в (v) и $(ir_v = true)$, то выполнить операцию удаление переключения окна в (v) и $ir_v = false$, если есть переключения слева в ni ($il_{ni} = true$), то выполнить операцию удаление переключения окна в (ni) и $il_{ni} = false$.

2.2. Если справа все вершины пустые и есть переключение справа в v, то выполнить операцию удаление переключения окна в (v) и $ir_v = false$.

3. Производится корректировка левого переключения (аналогично с правым).

3.1. Если есть слева непустая вершина-интервал pi, то:

3.1.1. если $lp_{pi} \neq fp_v$ и нет переключений слева в v ($il_v = false$), и нет переключения справа в pi ($ir_{pi} = false$):

3.1.1.1. если $c(pi,dest) - f(pi,dest) \geq cw$ (в предыдущем интервале есть место для переключения) и $e_v + f(v,dest) > c(v,dest) - cw$ (в текущем нет), то выполняется операция добавление переключения окна в (pi) $ir_{pi} = true$;

3.1.1.2. иначе выполняется операция добавление переключения окна в (v) и $il_v = true$;

3.1.2. если $lp_{pi} = fp_v$ и есть переключение слева в v ($il_v = true$), то выполнить операцию удаление переключения окна в (v) и $il_v = false$, если есть переключения справа в pi ($ir_{pi} = true$), то выполнить операцию удаление переключения окна в (pi) и $ir_{pi} = false$.

3.2. Если справа все вершины пустые и есть переключение справа в v, то выполнить операцию удаление переключения окна в (v) и $il_v = false$.

Удаление работы u из сети.

1. Для всех пар вершина-работа–вершина-интервал-процессор (u,v), где вершина-работа u соответствует не полностью размещенной работе.

2. Убирается поток от вершины-работы к вершине-интервалу-процессору, такой же поток вычесть из потока от вершины-интервала-процессора до стока. Убирается поток от источника до вершины-работы:

$$value = f(u,v),$$
$$e_{dist} = e_{dist} - f(u,v),$$
$$f(v,dist) = f(v,dist) - f(u,v),$$
$$f(dist,v) = -f(v,dist),$$
$$f(u,v) = 0, \quad f(v,u) = 0, \quad f(0,u) = 0, \quad f(u,0) = 0.$$

3. Выполняется операция учет раздела при удалении потока размера value раздела pt_u из вершины-интервала v.

4. Выполняется операция корректировка окон в v.

5. Пропускная способность от источника до вершины-работы становится равной нулю: $c(0,u) = 0$.

Размещение раздела *part* на процессор *proc*.

Для всех вершин-работ u, таких, что $pt_u = part$ и для всех ребер (u, v), таких, что $proc_v \neq proc$, пропускная способность $c(u,v) = 0$ и $PR_{pt_u} = proc$.

Удаление раздела *part* с процессора *proc*.

Для всех вершин-работ u, таких, что $pt_u = part$.

1. Для всех ребер (u, v), таких, что $proc_v \neq proc$: $c(u,v) = dur_v$.

2. Для всех ребер (u, v), таких, что $proc_v = proc$:

2.2. выполняется операция учет раздела при удалении потока размера *value* раздела pt_u из вершины-интервала-процессора v;

2.3. выполняется операция корректировка окон в v;

2.4. пропускная способность от вершины-работы до вершины-интервала-процессора становится равной нулю: $c(u,v) = 0$.

Проталкивание (u, v).

1. Поток $f(u,v)$ увеличивается на величину $\delta f(u,v) = \min(e_u, c(u,v) - f(u,v))$.

2. На $\delta f(u,v)$ увеличивается избыточный поток e_v.

3. Обратный поток $f(v,u)$ и избыточный поток e_u уменьшаются на $\delta f(u,v)$.

4. Если u – вершина-работа, v – вершина-интервал-процессор и раздел еще не размещен ни на один процессор, то выполнить операцию размещение раздела ptu на процессор $proc_v$; выполнить операцию учет раздела при добавлении потока размера $\delta f(u,v)$ раздела pt_u в вершину-интервал-процессор v и выполнить операцию корректировка окон в (v).

5. Если u – вершина-интервал-процессор, v – вершина-работа, то выполнить операцию учет раздела при удалении потока размера $\delta f(u,v)$ раздела pt_v в вершину-интервал-процессор и выполнить операцию корректировка окон в (u).

6. Если $e_v = 0$, то вершина убирается из списка переполненных вершин соответствующего раздела.

7. Если $e_v > 0$, то вершина добавляется в список переполненных вершин соответствующего раздела.

Проталкивание (u,v) в вершину-интервал-процессор с учетом остаточного потока $(v, dist)$.

В этой операции v является вершиной-интервалом-процессором.

1. Поток $f(u,v)$ увеличивается на величину:

$$\delta f(u,v) = \max(0, \min(e_u, c(u,v) - f(u,v), c(v,dist) - f(v,dist) - ccw - e_v)). \quad (3.1)$$

2. Избыточный поток e_v увеличивается на $\delta f(u,v)$. Обратный поток $f(v, u)$ и избыточный поток e_u уменьшаются на $\delta f(u,v)$.

3. Если u – вершина-работа, v – вершина-интервал и раздел еще не размещен ни на один процессор, то выполнить операцию размещение раздела pt_u на процессор $proc_v$; выполнить операцию учет раздела при добавлении потока размера $\delta f(u,v)$ раздела pt_u в вершину-интервал v и выполнить операцию корректировка окон в (v).

4. Если $e_v = 0$, то вершина убирается из списка переполненных вершин соответствующего раздела.

5. Если $e_v > 0$, то вершина добавляется в список переполненных вершин соответствующего раздела.

Разрядка вершины u.

Пока $e_u > 0$.

1. По порядку рассматриваем все доступные вершины для u.

2. При первом рассмотрении:

2.1. если v – вершина-интервал-процессор, то cw – число переключений окон, которые появятся при проталкивании потока в вершине-интервале-процессоре v;

2.2. вычисляется величина проталкиваемого потока, при условии, что $h_u = h_v + 1$, по формуле (3.1), и если она не равна нулю, то выполнить операцию проталкивание (u, v) в вершину-интервал с учетом остаточного потока $(v, dist)$.

3. При последующих рассмотрениях:

3.1. если $h_u = h_v + 1$, v – источник и $K > 0$, то $K = K - 1$ и выполнить операцию удаление раздела pt_u с процессора PR_{pt_u}, иначе выполнить операцию проталкивание (u, v).

4. Выполнить операцию подъем u.

5. Вернуться на первый шаг.

Возможная разрядка вершины u.

В данной операции u является вершиной-работой.

Для всех существующих ребер (u, v).

1. Если v является источником: если $h_u = h_v + 1$ и если $K = 0$, то $K = K - 1$ и выполнить операцию удаление раздела pt_u с процессора PR_{pt_u}, иначе выполнить операцию проталкивание (u, v).

2. Если v является вершиной-интервалом-процессором, то: если $h_u = h_v + 1$, то вычисляется величина проталкиваемого потока по формуле (3.1), и если она не равна нулю, то выполнить операцию проталкивание (u, v) в вершину-интервал с учетом остаточного потока $(v, dist)$.

Восстановление расписания.

Последовательно для всех процессоров выполняются следующие операции.

1. Если есть переключение окна слева, то текущее окно закрывается, учитывается время переключения, открывается окно первого раздела в этом интервале, вычитается единица из количества переключений окон в этом интервале.

2. Если есть переключение окон в середине, то (пока есть переключения в середине) окно закрывается через время, равное потоку этого раздела в эту вершину. Разделы перебираются, начиная с первого, заканчивая последним (неважно, какой порядок в центре), учитывается переключение, открывается окно следующего раздела.

3. Если есть переключение окна справа, то текущее окно закрывается, учитывается время переключения, открывается окно первого раздела следующего интервала.

Чтобы указать работы, которые выполняются в окне, а также их время выполнения, выделяемое в каждом окне, достаточно для каждой вершины-работы взять поток в соответствующий интервал окна. Величина потока и будет временем выполнения работы в данном окне. Если потока нет, то его величина равна нулю и работа в данном окне не выполняется.

Общая схема алгоритма.

1. Выполняется операция инициализация сети.

2. Строится поток.

2.1. Выбирается раздел, список переполненных вершин которого не пуст (если такого раздела нет, то переход к п. 3, далее для этих вершин u выполняются операции: возможная разрядка вершины u.

2.2. Пока списки переполненных вершин раздела и вершин-интервалов не пусты:

2.2.1. для всех переполненных вершин-интервалов-процессоров v выполняется операция разрядка вершины v;

2.2.2. для одной переполненной вершины раздела u выполняется операция разрядка вершины u.

2.3. Переход к п. 2.

3. Если есть не полностью размещенные работы, то для первой такой работы выполняется операция удаление работы из сети и переход к п. 2.

4. Выполняется операция восстановление расписания.

4. Свойства алгоритма построения статико-динамического расписания на основе алгоритма поиска максимального потока в сети. Результат работы алгоритма удовлетворяет условиям корректности расписания. В алгоритме основной операцией является операция разрядки вершины, которая в свою очередь состоит из операций проталкивания из вершины в вершину и операций подъема вершины. Операция подъема влияет только на очередность проталкиваний и не влияет на корректность расписания.

Операция проталкивания устроена таким образом, что при отсутствии избыточного потока в сети будут выполнены все условия корректности расписания.

Операция проталкивания, связанная с вершиной-интервалом, выполняет операции корректировки окон, которая добавляет в соответствующую вершину переключения окон между разделами – резервирует время для переключения между разделами, чтобы другие работы не могли его использовать. Окна строятся по длительностям работ одного раздела, следующих подряд. Повторный анализ сети на не полностью размещенные задачи исключает таковые из сети.

Так как размещение происходит сразу после первого проталкивания работы на вершину интервал-процессор данного процессора, то дальнейшее размещение работ этого раздела возможно только на этом процессоре (так как пропускные способности к остальным процессорам равны 0) до того момента, пока работа не захочет сделать проталкивание в исток, что равносильно тому, что не удается разместить весь раздел на этом процессоре. Тогда выполняется вторая операция. Поток отзывается из всех вершин от всех вершин-работ данного раздела (корректность расписания остается, так как учитываются все переключения окон). Пропускные способности до остальных процессоров восстанавливаются. Поэтому не может быть нарушено условие: работы одного раздела выполняются на одном процессоре. Построение сети можно осуществить за $O(pn^2)$. Восстановление расписания осуществляется за $O(pn)$.

Для одного процессора верно следующее: если n – число работ, то $V \le 3n + 2$ и $E \le 3n + 2n^2$. Первый шаг алгоритма заключается в возможной разрядке всех вершин одного раздела, который в худшем случае занимает $2n^2$ операций проталкивания. За этим следует разрядка всех вершин-интервалов, что занимает в худшем случае $2n(n+1)$ операций проталкивания. Эти действия выполняются для каждого раздела, значит нужно $n(2n^2 + 2n(n+1)$ операций проталкивания. Далее покажем, что для каждой пары вершина-работа – вершина-интервал возможны не более 10 проталкиваний. Рассмотрим функцию высоты для вершин-работ, она всегда больше 0. И если она равна 11, то проталкивание осуществляется в исток и этот поток больше в сети не появляется. Высота истока, равная 10, обеспечивает пять попыток проталкивания из одной и той же вершины-работы в вершину-интервал. Так как высоты вершин-интервалов не уменьшаются (когда проталкивание возвращает поток в вершину-работу), то высота становится больше высоты вершины-работы, из которой было осуществлено проталкивание. Следовательно, для всех пар вершин-работ и вершин-интервалов-процессоров возможно не больше 10 проталкиваний. При удалении работы нужно повторить алгоритм сначала, максимум можно удалить n работ, что соответствует n итерациям алгоритма – $n(2n^3 + 2n(n+1) + 20n^2)$. Общая сложность поиска потока для одного процессора алгоритмом равна $O(n^4)$, если требуется удаление работ, и $O(n^3)$, если получается спланировать все работы. Тогда сложность для всех процессоров составляет $O(pn^3)$.

Так как имеется только K попыток перераспределения раз дела, то сложность равна $O(Kpn^3)$ при полном планировании расписания и $O(Kpn^4)$ – при удалении некоторых работ.

Экспериментальное исследование проводилось в системе с ОС Windows 7 64bit, процессором Intel Core i3-2370M 2.39GHz, ОЗУ 8Гб.

На сгенерированных исходных данных алгоритм построения однопроцессорного статико-динамического расписания разместил все работы. При этом время построения решения не сильно зависит от загрузки процессора и не превышает 1.2 с для 1000 работ. Также число разделов почти не влияет на точность и время решения.

На сгенерированных исходных данных алгоритм продемонстрировал полную планируемость работ при загрузке процессоров до 90% для двух процессоров. Увеличение числа процессоров плохо влияет на число планируемых работ, так для трех процессоров планируемость всех работ возможна при загрузках процессора до 70% при больших значениях – возможно размещение 99% работ. Увеличение числа процессоров до восьми приводит к размещению 99% работ при загрузке до 70% и к размещению 90% работ при загрузке до 90%. Время выполнения алгоритма построения многопроцессорного статико-динамического расписания растет с увеличением загрузки процессоров для восьми процессоров. Это связано с тем, что число размещенных работ меньше 100% и требуется дополнительное время на исключение работ, для которых не нашлось места в статико-динамическом расписании. Время построения такого расписания может занимать 20 мин. Для числа процессоров от двух до четырех время построения статико-динамического расписания не превосходит 4 мин и точность находится на уровне 99–100% при загрузках процессора до 90%.

Заключение. Задача построения статико-динамических расписаний возникает при проектировании информационно-управляющих систем реального времени с архитектурой интегрированной модульной авионики.

Для ряда задач построения расписаний с прерываниями высокую эффективность по критериям точность и вычислительная сложность показали алгоритмы, основанные на нахождении максимального потока в транспортной сети. Основными проблемами, препятствующими применению известных алгоритмов, основанных на нахождении максимального потока в транспортной сети, для построения статико-динамических расписаний являются: проблема учета принадлежности работ к разделам и проблема построения набора окон.

Проблемы учета принадлежности работ к разделам и построения набора окон в предложенном алгоритме решаются за счет модификации алгоритма нахождения максимального потока в сети.

Экспериментальное исследование свойств алгоритма показало его высокую эффективность по критериям точность и вычислительная сложность на многих классах исходных данных.

СПИСОК ЛИТЕРАТУРЫ

1. *Костенко В.А.* Архитектура программно-аппаратных комплексов бортового оборудования // Изв. вузов. Приборостроение. 2017. Т. 60. № 3. С. 229–233.
2. Arinc Specification 653. Airlines Electronic Engineering Committee. (http://www.arinc.com).
3. *Годунов А.Н.* Операционные системы реального времени Багет 3.0 // Программные продукты и системы, 2010. № 4. С. 15–19.
4. *Годунов А.Н., Солдатов В.А.* Операционные системы семейства Багет (сходство, отличия и перспективы) // Программирование. 2014. № 5. С. 69–76.
5. *Балаханов В.А., Костенко В.А.* Способы сведения задачи построения статико-динамического однопроцессорного расписания для систем реального времени к задаче нахождения на графе маршрута // Программные системы и инструменты. 2007. № 8. С. 148–156.
6. *Балаханов В.А., Кокарев В.А., Костенко В.А.* Возможность применения муравьиных алгоритмов для решения задачи построения статико-динамических расписаний // Тр. V Московской междунар. конф. по исследованию операций (ORM2007). М.: МАКС Пресс, 2007. С. 238–240.
7. *Balashov V.V., Balakhanov V.A., Kostenko V.A.* Scheduling of Computational Tasks in Switched Network-based IMA Systems // Proc. Intern. Conf. on Engineering and Applied Sciences Optimization. National Technical University of Athens (NTUA), Athens, Greece, 2014. P. 1001–1014.
8. *Балашов В.В.* Семейство систем автоматизации проектирования бортовых вычислительных систем реального времени // Программные продукты, системы и алгоритмы. 2017. № 4. С. 1–19.
9. *Federgruen A., Groenevelt H.* Preemptive Scheduling of Uniform Machines by Ordinary Network Flow Technique // Management Science. V. 32. № 3. 1986.
10. *Gonzales T., Sanhi S.* Preemptive Scheduling of Uniform Processor Systems // J. Association for Computing Machinery. V. 25. № 1. 1978.
11. *Фуругян М.Г.* Планирование вычислений в многопроцессорных АСУ реального времени с дополнительным ресурсом // АиТ. 2015. № 3. С. 144–150.
12. *Фуругян М.Г.* Планирование вычислений в многопроцессорных системах с несколькими типами дополнительных ресурсов и произвольными процессорами // Вестн. МГУ. Сер. 15. Вычислительная математика и кибернетика. 2017. № 3. С. 38–45.
13. *Фуругян М.Г.* Составление расписаний в многопроцессорных системах с дополнительными ограничениями // Изв. РАН. ТиСУ. 2018. № 2. С. 52–59.
14. *Костенко В.А., Смирнов А.С.* Алгоритм построения однопроцессорных статико-динамических расписаний // Вестн. МГУ. Сер. 15. Вычислительная математика и кибернетика. 2018. № 1. С. 45–52.