УДК 681.518.54 doi: 10.17586/2226-1494-2020-20-5-677-682

СИНТЕЗ ИЕРАРХИЧЕСКОЙ ДИАГНОСТИЧЕСКОЙ МОДЕЛИ ПОТОКОВОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ РЕАЛЬНОГО ВРЕМЕНИ

Е.В. Лукоянов, А.М. Грузликов

АО «Концерн «ЦНИИ «Электроприбор», Санкт-Петербург, 197046, Российская Федерация
Адрес для переписки: lukoyanov.egor@mail.ru

Информация о статье
Поступила в редакцию 09.07.20, принята к печати 15.08.20
Язык статьи — русский

Ссылка для цитирования: Лукоянов Е.В., Грузликов А.М. Синтез иерархической диагностической модели потоковой вычислительной системы реального времени // Научно-технический вестник информационных технологий, механики и оптики. 2020. Т. 20. № 5. С. 677–682. doi: 10.17586/2226-1494-2020-20-5-677-682

Аннотация

Предмет исследования. Рассмотрены вопросы проектирования средств диагностирования нарушений в адресации информационных обменов между программными модулями для потоковых вычислительных систем реального времени. Несмотря на декомпозицию процессов проектирования в таких системах, вопросы диагностирования и повышения отказоустойчивости остаются актуальными для каждого уровня иерархии. **Метод.** Предлагаемые процедуры синтеза иерархической модели потоковой вычислительной системы являются развитием метода тестового диагностирования на основе использования параллельной модели. **Основные результаты.** Приведено краткое описание подхода к тестовому диагностированию на основе параллельной модели. Разработан алгоритм синтеза иерархической диагностической модели. Модель обеспечивает минимизацию количества диагностической информации, передаваемой по каналам обмена, снижая уровень вводимой избыточности и повышая тем самым уровень надежности. **Практическая значимость.** Разработанная иерархическая модель позволяет существенно сокращать время проектирования средств диагностирования за счет снижения необходимого количества диагностических алгоритмов.

Ключевые слова
параллельная модель, потоковая вычислительная система, тестовое диагностирование, периодически нестационарные динамические системы

Благодарности
Работа выполнена при поддержке гранта Российского фонда фундаментальных исследований № 19-08-00052.

doi: 10.17586/2226-1494-2020-20-5-677-682

HIERARCHICAL DIAGNOSTIC MODEL SYNTHESIS FOR DATAFLOW REAL-TIME COMPUTING SYSTEM

E.V. Lukoyanov, A.M. Gruzlikov

JSC, CSRI “Elektropribor”, Saint Petersburg, 197046, Russian Federation
Corresponding author: lukoyanov.egor@mail.ru

Article info
Received 09.07.20, accepted 15.08.20
Article in Russian

For citation: Lukoyanov E.V., Gruzlikov A.M. Hierarchical diagnostic model synthesis for dataflow real-time computing system. *Scientific and Technical Journal of Information Technologies, Mechanics and Optics*, 2020, vol. 20, no. 5, pp. 677–682 (in Russian). doi: 10.17586/2226-1494-2020-20-5-677-682

Abstract

Subject of Research. The paper considers design issues for diagnostic tools of fault detection in addressing information exchanges between software modules for real-time dataflow computing systems. Despite the decomposition of the design processes in such systems, the issues of diagnostics and fault tolerance remain relevant for each hierarchy level. **Method.** The proposed synthesis procedures for the hierarchical model of a dataflow computing system are the result of the test diagnostics method development based on the parallel model application. **Main Results.** The paper presents a brief description of the test diagnostics method based on the parallel model. An algorithm for hierarchical diagnostics

model synthesis is developed. The model minimizes the amount of diagnostic data transmitted through the exchange channels, reducing the redundancy level introduced into the system and thereby increasing the level of reliability. **Practical Relevance.** The developed hierarchical model reduces significantly the design time for diagnostic tools as a result of reducing the required number of diagnostic modules included in it.

Keywords
parallel model, dataflow computing system, test diagnosis, periodical non-stationary dynamic systems

Acknowledgments
This work was supported by the project No. 19-08-00052 of the Russian Foundation for Basic Research, Russian Federation.

Введение

Рассмотрение вопросов диагностирования занимает важное место в процессе проектирования систем обработки информации и управления, поскольку от качества их решения зависит надежность и отказоустойчивость систем. Применяемые на практике решения основываются на техниках функционального и тестового диагностирования [1–3]. Объектом рассмотрения в настоящей статье является произвольная бортовая потоковая вычислительная система (ПВС) реального времени, представленная набором функционально связанных программных модулей (ПМ), которые исполняются либо на одном вычислительном устройстве (ВУ), либо путем распределенных вычислений на совокупности ВУ. В рамках тестового диагностирования одним из возможных подходов для математического описания рассматриваемого класса систем является подход с использованием так называемой параллельной модели [4]. Модель встраивается в программное обеспечение системы, параллельно основным функциональным алгоритмам, и предназначена для тестового диагностирования нарушений в адресации обменов между ПМ системы. Известен также подход с конечно-автоматными моделями [5, 6]. Однако в этом случае асимптотическая сложность алгоритма построения теста для системы, определяющая зависимость роста времени его исполнения от размерности модели, характеризуется экспоненциальной зависимостью от размерности модели. В случае использования параллельной модели разработчик сам выбирает алгоритмы обработки тестовых последовательностей в каждом из ПМ, причем делает это так, чтобы упростить и алгоритм построения теста и сам тест, а именно, выбирает линейную модель, которая позволяет использовать алгоритмы построения тестов с асимптотической сложностью, характеризующейся полиномиальной зависимостью от размерности модели. Использование эффективных алгоритмов с точки зрения асимптотической сложности построения тестов становится особенно актуально в случае ПВС реального времени.

В работе предлагается алгоритм синтеза иерархической диагностической модели ПВС реального времени, которая является параллельной по отношению к системе. Использование иерархической модели позволяет уменьшать размерность модели, тем самым сокращая объем диагностической информации, передаваемой между ПМ, улучшая показатели надежности и времени работы алгоритмов построения тестов.

Подход к тестовому диагностированию с параллельной моделью

Проиллюстрируем процедуру построения модели на простом примере (рис. 1), где приведен пример системы реального времени, заданной графом межмодульных связей. В системе реализуются три функционально связанных ПМ: $ПМ_1$, $ПМ_2$ и $ПМ_3$, которые могут быть размещены как на разных процессорах системы, так и на одном. Каждый из ПМ на основе входных данных (u_1 — для $ПМ_1$, u_2 и y_3 — для $ПМ_2$, y_1 и y_2 — для $ПМ_3$) формирует выходные данные (y_1, y_2 и y_3 соответственно). Входные данные поступают и обрабатываются в реальном времени с некоторым заданным периодом. В системе все входные потоки конкретного ПМ являются аргументами реализуемой им функции, необходимыми для ее вычисления.

На рис. 2 система приведена совместно с параллельной моделью и средствами диагностирования (СД). Стоит отметить, что модель является вводимой избыточностью для системы и представлена диагностическими алгоритмами: π_1, π_2 и π_3, связи между которыми обозначены зеленым цветом. СД реализованы в виде отдельного ПМ, выделенного штрихпунктирной линией, и состоят из генератора тестов (ГТ), генератора эталонных реакций (ГЭР) и компаратора (К). Тестовые данные u_1' и u_2', формируемые СД, дополняют входные данные для новых $ПМ_1'$ и $ПМ_2'$, также выделенных штрихпунктирной линией и получившихся в результате объединения исходных ПМ с диагностическими алгоритмами. Таким образом, в каждый $\{ПМ_i\}_{i=1}^{3}$ по каналам обмена, выделенным черным цветом, поступает информация, которая обрабатывается штатными алгоритмами. Примером штатного алгоритма может служить обработка потока цифровых данных с массива датчиков или решение задач ориентации и навигации в робототехнических комплексах [7], в общем случае являющиеся нелинейными. Параллельно с этим, отсюда и название модели, тестовые данные обрабатываются диагностическими алгоритмами, реагирующими на события приема/выдачи информации. Поскольку механизм обмена реальными $\{y_i\}_{i=1}^{3}$ и тестовыми данными

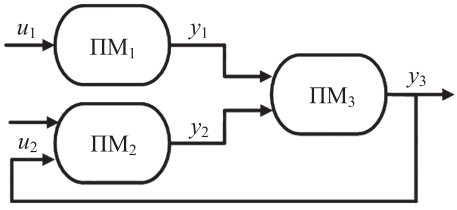


Рис. 1. Граф межмодульных связей системы

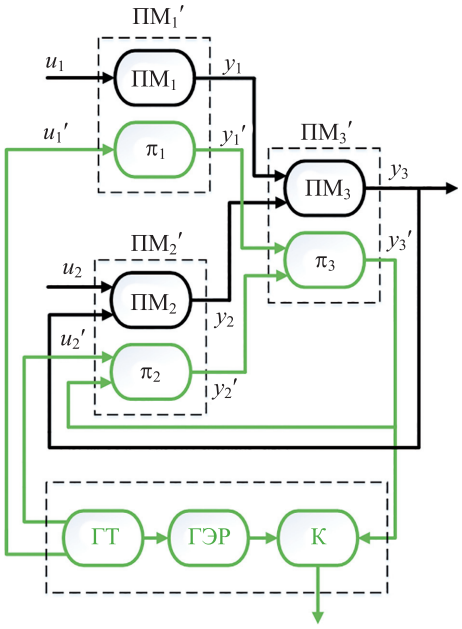


Рис. 2. Граф межмодульных связей избыточной системы и средств тестового диагностирования

$\{y_i'\}_{i=1}^{3}$ в системе является общим, возникает возможность по наблюдаемым в процессе работы тестовым данным делать вывод о наличии или отсутствии нарушений в адресации обменов. Заметим, что искажения реальных данных в процессе обмена при сохранении графа межмодульных связей не входят в этот класс рассматриваемых нарушений.

Таким образом, процесс синтеза параллельной модели заключается в построении диагностических алгоритмов обработки тестовых последовательностей. Модель, как будет показано в разделе о синтезе параллельной модели с независимыми цепями, имеет вид линейной дискретной периодически нестационарной системы. В терминах этой модели класс рассматриваемых нарушений определяется как всевозможные искажения матриц этой модели.

Параллельная модель с независимыми цепями

Одним из возможных вариантов параллельной модели является модель с независимыми цепями [8]. На первом этапе формируется структура модели, которая представляет собой множество вычислительных путей (трасс), составляющих покрытие дуг графа межмодульных связей исходной системы. При этом под вычислительным путем понимаем последовательность срабатывающих ПМ, соединяющую некоторый вход с выходом. Затем с каждым из полученных путей сопоставляется цепь $l_j = \{v_i\}_{i=1}^{n_j}$, из такого числа динамических звеньев v_i, через сколько ПМ проходит данный путь, n_j — общее количество звеньев в j-ой цепи. Структура модели для абстрактного примера, состоящая из множества независимых цепей $\{l_j\}_{j=1}^{m}$, где m — общее количество независимых цепей в модели, приведена на рис. 3, *а*. На рисунке изображены динамические звенья $v_{i,j}$, где i — номер звена в цепи; j — номер цепи, а также диагностические алгоритмы $\{\pi_i\}_{i=1}^{p}$, где p — количество ПМ в системе.

На втором этапе формирования модели определяется вид динамических звеньев. При этом учитывается, что искомая динамическая модель системы далее используется для построения тестов и что процедура построения тестов упрощается, если модель системы, во-первых, линейна, а во-вторых, управляема и наблюдаема [3]. Отсюда можно сформулировать требование к динамическим звеньям – они должны быть линейны, т. е.

$$\begin{aligned} \mathbf{x}_{i,j}'(k+1) &= \mathbf{f}_{i,j}\mathbf{x}_{i,j}'(k) + \mathbf{g}_{i,j}\mathbf{u}_{i,j}'(k), \\ \mathbf{y}_{i,j}'(k) &= \mathbf{h}_{i,j}\mathbf{x}_{i,j}'(k),\ i = \overline{1,n},\ j = \overline{1,m}, \end{aligned} \qquad (1)$$

где $\mathbf{x}_{i,j}'(k) \in F^n$, $\mathbf{u}_{i,j}'(k) \in F^q$, $\mathbf{y}_{i,j}'(k) \in F^p$ — векторы состояния, входа и выхода соответственно для i-го звена модели j-й цепи; n — размерность вектора состояния, q — размерность входного вектора; p — размерность выходного вектора; $\mathbf{f}_{i,j} \in F^{n\times n}$, $\mathbf{g}_{i,j} \in F^{n\times q}$, $\mathbf{h}_{i,j} \in F^{n\times p}$ — матрицы динамики, входа и выхода соответственно; $F = \{0,1\}$ — двоичное множество. Кроме того, звенья

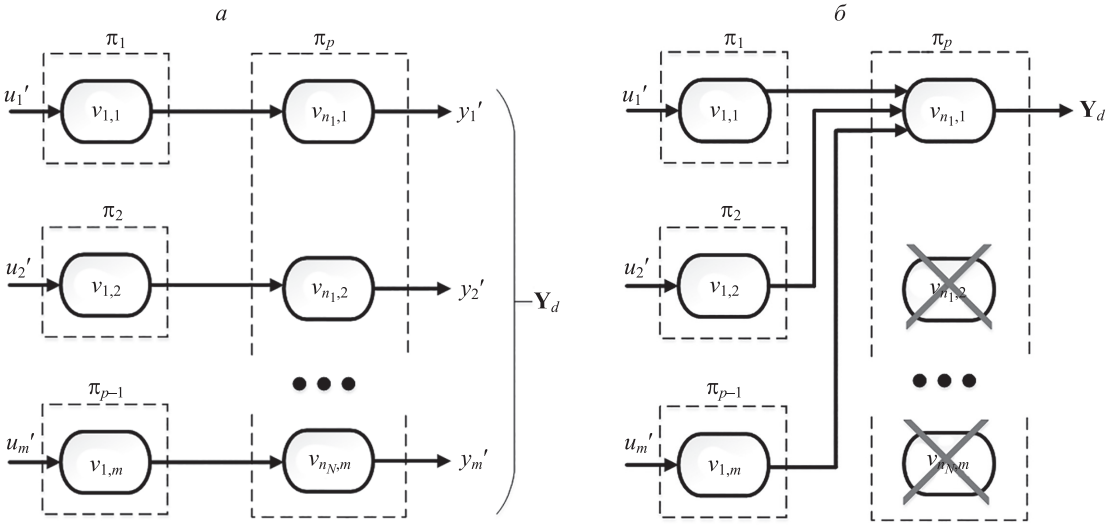


Рис. 3. Структура модели для системы: с независимыми цепями (*а*); со слиянием цепей (*б*)

должны быть таковы, чтобы модель системы была наблюдаемой и управляемой.

Динамическое описание цепи, состоящей из звеньев, получается по следующим правилам. Вектор состояния $\mathbf{x}_j'(k)$ цепи формируется из векторов состояния звеньев $\mathbf{x}_{i,j}'(k)$, в нее входящих. Предположим, что в каждый момент времени в системе происходит лишь один обмен, тогда перенос информации между ПМ и СД может быть описано с помощью блочных матриц $\mathbf{F}_j(k)$, $\mathbf{G}_j(k)$, $\mathbf{H}_j(k)$, составленных из матриц динамики, входа и выхода звеньев, описываемых моделью (1). На практике данное предположение не выполняется, однако, как показано в работах [3, 9, 10], это не является препятствием для использования таких моделей при построении тестов. Для удобства описания свяжем с каждой последовательностью матриц на интервале, равном периоду, свою последовательность индексов, полученных из множества $\{1, 2, \ldots, N\}$ в результате циклического сдвига. Например, при $N = 3$ получаем множество, состоящее из трех последовательностей индексов $\Gamma = \{1, 2, 3; 2, 3, 1; 3, 1, 2\}$. Обозначим за γ_r элемент множества Г. Тогда

$$\mathbf{x}_j'(k+1) = \mathbf{F}_j(\gamma_r(k))\mathbf{x}_j'(k) + \mathbf{G}_j(\gamma_r(k))\mathbf{u}_j'(k),$$

$$\mathbf{y}_j'(k) = \mathbf{H}_j(\gamma_r(k))\mathbf{x}_j'(k),\ k = \overline{1,N},$$

где $\mathbf{u}_j'(k)$ — вектор входных воздействий цепи; $\mathbf{y}_j'(k)$ — вектор выходной реакции цепи; N — число обменов на периоде работы системы. Таким образом, модель цепи описывается с помощью периодически нестационарной линейной динамической системы.

Однако применение параллельной модели с независимыми цепями может в некоторых случаях потребовать передачи через каналы обмена большого количества диагностической информации, что не всегда допустимо. В подобных ситуациях целесообразно воспользоваться приемом, заключающимся в обработке нескольких массивов информации одним звеном (слияние вычислительных путей). Этот вариант иллюстрируется на рис. 3, *б*, где выходные массивы информации звеньев $v_{1,2}$ и $v_{1,m}$ обрабатываются звеном $v_{n_1,1}$, а звенья $v_{n_2,2}$ и $v_{n_N,m}$ исключаются. В результате сокращается количество диагностических звеньев, размерность диагностического вектора $\mathbf{Y}_d$, а следовательно, и объем выдаваемой из ПМ в каждом такте диагностической информации.

Алгоритм синтеза иерархической модели

В рассмотренный в предыдущем разделе алгоритм синтеза параллельной модели предлагается включить дополнительный (промежуточный) этап, когда в полученную структуру модели вводятся точки слияния цепей.

Приведем формальную постановку задачи процедуры поиска точек слияния на множестве независимых цепей и алгоритм ее решения, при этом воспользуемся терминологией, принятой в [11] для описания алгоритмов и структур данных. Для работы алгоритма используются две промежуточные структуры данных: двунаправленный список для хранения независимых цепей модели и хеш-таблица для выполнения операций со списками.

Постановка задачи. Дано множество двусвязных списков $L = \{l_i\}_{i=1}^{m}$, где m — общее количество списков; $l_i = \langle V_i \rangle$; $V_i = \{v_j\}_{j=1}^{n_i}$ — множество узлов i-го списка; n_i — количество узлов в i-ом списке; $N = \left(\sum_1^m n_i\right)$ — общее количество узлов всех списков из множества L. Пусть задано отображение $\rho_i: V_i \to P$ множества V_i узлов i-го списка на множество программных модулей P исходной рассматриваемой системы. Требуется разработать алгоритм поиска точек слияния списков.

Алгоритм предполагает рекурсивное слияние очередного списка со структурой из списков, объединенных на предыдущих шагах. На каждом шаге формируется не более одного узла слияния. Обнаружение точки слияния осуществляется по правилу: если $\rho(v_k) = \rho(v_h)$ и $\rho(v_{k-1}) \neq \rho(v_{h-1})$, где $v_k \in V_i$ и $v_h \in V_j$ узлы списков l_i и l_j соответственно, то узел v_k является точкой слияния. Индексы h и k в начальный момент имеют значения, соответствующие хвостовому узлу списка.

Входные данные: L — множество двусвязных списков; $M = \varnothing$ — множество узлов слияния; P — множество ПМ исходной системы.

Выходные данные: количество двусвязных списков, которые не удалось слить в общую структуру; M — множество узлов слияния.

Псевдокод алгоритма, обеспечивающего поиск точек слияния списков, представлен на рис. 4, *а*.

Суть алгоритма заключается в следующем:

1) на первом этапе (строки 6–14) хвостовые узлы всех списков по ключу заносятся в хеш-таблицу. При этом проводится проверка на наличие в ячейке хеш-таблицы более чем одного узла. В случае обнаружения вызывается процедура *NodeMerge* (рис. 4, *б*, строки 17–34), которая возвращает номер узла слияния. Получившийся узел заносится в множество M (процедура *PushBack*);
2) процедура слияния списков *NodeMerge* является рекурсивной, так как в процессе процедуры слияния происходит спуск по имеющемуся списку (структуре) до тех пор, пока вершины объединяемых списков совпадают. Когда неравенство выполняется, списки (структуры) объединяются, т. е. происходит перенаправление указателей, а дублирующие узлы удаляются с помощью функции *DeleteNodes*;
3) в конце алгоритма по разности мощностей исходного множества L и получившегося в результате слияния множества M, вычисляется количество списков, которое не участвовало в слиянии.

Пример. Рассмотрим применение алгоритма поиска точек слияния, предварительно расширив граф межмодульных связей, представленный на рис. 1. В результате получаем граф, представленный на рис. 5.

Граф имеет семь вершин, которые пронумерованы в соответствии с ПМ, которому они принадлежат. На первом этапе выделяем независимые цепи (списки): $l_1 = \{4, 2, 1, 2, 1\}$, $l_2 = \{5, 2, 1\}$, $l_3 = \{6, 3, 1, 3, 1\}$, $l_4 = \{7, 3, 1\}$. На втором этапе применим процедуру поиска точек слияний. Рассмотрим пару списков (l_1, l_2). Начиная с концов списков последовательно

а

```
1: procedure ChainMerge(L, M, P)
2:     T ← MakeHash(|P|, 0)
3:     M ← MakeVector()
4:     key ← 0
5:     node ← 0
6:     for all l ∈ L do
7:         key ← HashSearch(T, Tail(l))
8:         if T[key] ≠ 0 then
9:             node ← NodeMerge(T[key], Tail(l))
10:            PushBack(M, node)
11:        else
12:            HashInsert(T, Tail(l))
13:        end if
14:    end for
15:    return |L| − |M| − 1
16: end procedure
```

б

```
17: procedure NodeMerge(N1, N2)
18:     mergednode ← 0
19:     if N1.listnum ≠ N2.listnum then
20:         if N1.nodenum == N2.nodenum then
21:             for all prev ∈ N1.prev[] do
22:                 if prev.nodenum == N2.prev.nodenum then
23:                     mergednode ← NodeMerge(prev, N2.prev)
24:                     return mergednode
25:                 end if
26:             end for
27:             N1.prev[] ← PushBack(N2.prev)
28:             N2.prev.next ← N1
29:             DeleteNodes(N2)
30:             mergednode ← N1
31:             return mergednode
32:         end if
33:     end if
34: end procedure
```

Рис. 4. Алгоритм поиска точек слияния списков (*а*); процедура объединения двух вершин разных цепей (*б*)

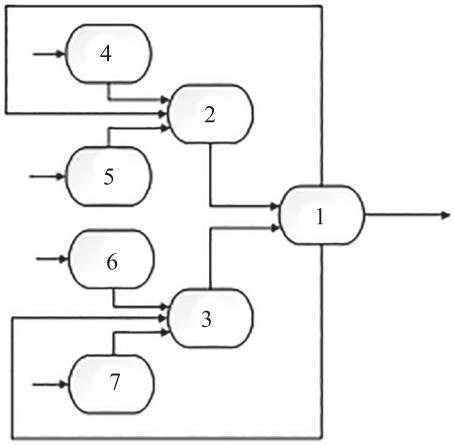


Рис. 5. Расширенный граф межмодульных связей системы

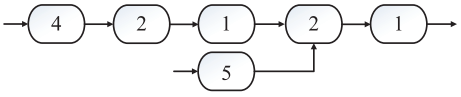
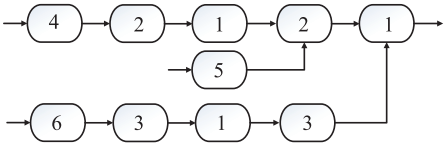
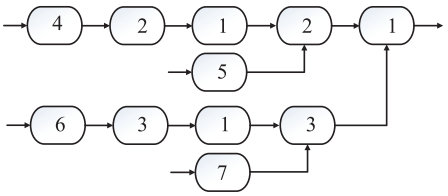
начинаем искать совпадающие по значению узлы, вплоть до вершины со значением (2), так как для этого узла выполняется условие $\rho(v_k) = \rho(v_h) \Leftrightarrow 2 = 2$, $\rho(v_{k-1}) \neq \rho(v_{h-1}) \Leftrightarrow 1 \neq 5$. Вершина (2) и будет являться узлом слияния. Отразим этот факт в таблице. Далее аналогичным способом получим узлы слияния для списков l_3 и l_4.

Таким образом, выполнение алгоритма может привести к следующим ситуациям:

1) структура, состоящая из независимых цепей (совпадает с исходной), если $|M| = 0$;
2) структура, состоящая из множества независимых цепей и множества структур с несколькими точками слияния, если $|L| - |M| \neq 0$;
3) полностью иерархическая структура, если $|L| - |M| = 0$.

Результирующая асимптотическая сложность алгоритма является логарифмической с полиномиальным коэффициентом $O(|L|^2 \log_{|L|} N)$, однако при условии $|L| << N$, когда количество звеньев в исходной модели много больше количества независимых цепей (случай малого количества длинных вычислительных путей в системе), сложность становится логарифмической $O(\log_{|L|} N)$ и зависит от общего количества звеньев в модели N.

Таблица. Перечень списков после слияния

Номера списков	Структура после слияния	Точка слияния
$l_1 \cup l_2$		2
$l_1 \cup l_2 \cup l_3$		1
$l_1 \cup l_2 \cup l_3 \cup l_4$		3

Заключение

Представлены результаты исследования иерархической диагностической модели потоковой вычислительной системы реального времени. Модель встраивается в программное обеспечение системы и предназначена для тестового диагностирования нарушений в адресации обменов между программными модулями системы. Предложен алгоритм синтеза структуры модели для минимизации количества диагностических алгоритмов и, как следствие, уменьшения диагностической информации, передаваемой по каналам обмена. Асимптотическая сложность алгоритма позволяет реализовать его в любом современном бортовом вычислителе общего назначения с целью оперативной реконфигурации модели под изменения, происходящие в системе в процессе ее эксплуатации.

Однако, по-прежнему, остаются вопросы, требующие дополнительного исследования, в частности, доказательство оптимальности алгоритма поиска точек слияния, а также сбор статистики в условиях применения модели в составе средств диагностирования реальной бортовой потоковой вычислительной системы.

Литература

1. Isermann R. Fault-Diagnosis Applications: Model-Based Condition Monitoring: Actuators, Drives, Machinery, Plants, Sensors, and Fault-Tolerant Systems. Springer Science & Business Media, 2011. XVI, 354 p. doi: 10.1007/978-3-642-12767-0
2. Issues of Fault Diagnosis for Dynamic Systems / ed. by R.J. Patton, P.M. Frank, R.N. Clark. Springer Science & Business Media, 2000. 597 p. doi: 10.1007/978-1-4471-3644-6
3. Колесов Н.В., Толмачева М.В., Юхта П.В. Системы реального времени. Планирование, анализ, диагностирование. СПб.: ОАО «Концерн «ЦНИИ «Электроприбор», 2014. 180 с.
4. Грузликов А.М., Колесов Н.В., Лукоянов Е.В. Тестовое диагностирование нарушений адресации информационных обменов в вычислительных системах с использованием параллельной модели // Известия Российской академии наук. Теория и системы управления. 2018. № 3. С. 76–89. doi: 10.7868/S0002338818030071
5. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Теория соответствия для систем с блокировками и разрушением. М.: Физматлит, 2008. 412 с.
6. Бурдонов И.Б., Косачев А.С., Кулямин В.В. Использование конечных автоматов для тестирования программ // Программирование. 2000. Т. 26. № 2. С. 12–28.
7. Ривкин Б.С. Первые отечественные навигационные комплексы для гражданских судов // Гироскопия и навигация. 2018. Т. 26. № 4. С. 96–104. doi: 10.17285/0869-7035.2018.26.4.096-104
8. Грузликов А.М., Колесов Н.В. Дискретно-событийная диагностическая модель распределенной вычислительной системы. Независимые цепи // Автоматика и телемеханика. 2016. № 10. С. 140–155.
9. Introduction to Discrete Event Systems / ed. by C.G. Cassandras, S. Lafortune. 2nd ed. New York: Springer, 2008. 770 p. doi: 10.1007/978-0-387-68612-7
10. Zaytoon J., Lafortune S. Overview of fault diagnosis methods for Discrete Event Systems // Annual Reviews in Control. 2013. V. 37. N 2. P. 308–320. doi: 10.1016/j.arcontrol.2013.09.009
11. Кормен Т., Лейзерсон Ч., Ривест Р., Штайн К. Алгоритмы. Построение и анализ. М.: Издательский дом «Вильямс», 2009. 1296 с.

References

1. Isermann R. *Fault-Diagnosis Applications: Model-Based Condition Monitoring: Actuators, Drives, Machinery, Plants, Sensors, and Fault-Tolerant Systems*. Springer Science & Business Media, 2011, XVI, 354 p. doi: 10.1007/978-3-642-12767-0
2. *Issues of Fault Diagnosis for Dynamic Systems*. Ed. by R.J. Patton, P.M. Frank, R.N. Clark. Springer Science & Business Media, 2000, 597 p. doi: 10.1007/978-1-4471-3644-6
3. Kolesov N.V., Tolmacheva M.V., Yukhta P.V. *Real-Time Systems. Planning, Analysis, Diagnostics*. St. Petersburg, Concern CSRI Elektropribor, 180 p. (in Russian)
4. Gruzlikov A.M., Kolesov N.V., Lukoyanov E.V. Test-based diagnosis of faults in data exchange addressing in computer systems using parallel model. *Journal of Computer and Systems Sciences International*, 2018, vol. 57, no. 3, pp. 420–433. doi: 10.1134/S1064230718030024
5. Burdonov I.B., Kosachev A.S., Kuliamin V.V. *Compliance Theory for Lock and Crash Systems*. Moscow, Fizmatlit Publ., 2008, 412 p. (in Russian)
6. Burdonov I.B., Kossatchev A.S., Kulyamin V.V. Application of finite automatons for program testing. *Programming and Computer Software*, 2000, vol. 26, no. 2, pp. 61–73. doi: 10.1007/BF02759192
7. Rivkin B.S. Russia's first integrated navigation systems for commercial vessels. *Gyroscopy and Navigation*, 2019, vol. 10, no. 1, pp. 35–40. doi: 10.1134/S207510871901005X
8. Gruzlikov A.M., Kolesov N.V. Discrete-event diagnostic model for a distributed computational system. Independent chains. *Automation and Remote Control*, 2016, vol. 77, no. 10, pp. 1805–1817. doi: 10.1134/S0005117916100076
9. *Introduction to Discrete Event Systems*. Ed. by C.G. Cassandras, S. Lafortune. 2nd ed. New York, Springer, 2008, 770 p. doi: 10.1007/978-0-387-68612-7
10. Zaytoon J., Lafortune S. Overview of fault diagnosis methods for Discrete Event Systems. *Annual Reviews in Control*, 2013, vol. 37, no. 2, pp. 308–320. doi: 10.1016/j.arcontrol.2013.09.009
11. Cormen Th.H., Leiserson Ch.E., Rivest R.L., Stein C. *Introduction to Algorithms*. McGraw-Hill, 2003, 1056 p.

Авторы

Лукоянов Егор Васильевич — младший научный сотрудник, АО «Концерн «ЦНИИ «Электроприбор», Санкт-Петербург, 197046, Российская Федерация, Scopus ID: 57193712278, ORCID ID: 0000-0003-3882-4315, lukoyanov.egor@mail.ru

Грузликов Александр Михайлович — кандидат технических наук, начальник отдела, АО «Концерн «ЦНИИ «Электроприбор», Санкт-Петербург, 197046, Российская Федерация, Scopus ID: 56037536900, ORCID ID: 0000-0001-8814-0726, agruzlikov@yandex.ru

Authors

Egor V. Lukoyanov — Junior Researcher, JSC, CSRI "Elektropribor", Saint Petersburg, 197046, Russian Federation, Scopus ID: 57193712278, ORCID ID: 0000-0003-3882-4315, lukoyanov.egor@mail.ru

Alexander M. Gruzlikov — PhD, Department Head, JSC, CSRI "Elektropribor", Saint Petersburg, 197046, Russian Federation, Scopus ID: 56037536900, ORCID ID: 0000-0001-8814-0726, agruzlikov@yandex.ru