УДК 004.41(075.8) DOI: 10.17587/it.25.20-25

М. В. Сарамуд, инженер, e-mail: msaramud@gmail.com,
И. В. Ковалев, д-р техн. наук, проф., e-mail: kovalev.fsu@mail.ru,
В. В. Лосев, канд. техн. наук, доц., e-mail: basilos@mail.ru,
М. О. Петросян, аспирант, e-mail: mopetrosyan@gmail.com,
А. О. Калинин, аспирант, e-mail: andrey.kalinin@phkp.ru,
Сибирский государственный университет науки и технологий имени академика М. Ф. Решетнева

Сравнение отказоустойчивых моделей программного обеспечения в имитационной среде исполнения

Рассматривается методика, позволяющая сравнить основные отказоустойчивые модели программного обеспечения с программной избыточностью на основе методологии: N-версионного программирования (с четкими и нечеткими алгоритмами голосования), N-версионного программирования с самопроверкой, восстанавливающихся блоков, согласованных восстанавливающихся блоков и t/(n−1)-версионного программирования. Для реализации методики создана имитационная среда исполнения отказоустойчивых программных моделей. Проанализированы результаты моделирования в имитационной среде.

Ключевые слова: *мультиверсионное программирование, программная избыточность, надежность, блоки восстановления, среда исполнения*

Введение

В последние годы активно развиваются отрасли, требующие надежных, отказоустойчивых систем управления. К их числу относятся высокотехнологичные производства [1], использующие композитные и опасные материалы, автономные беспилотные объекты — от мультироторных систем до автомобилей с функцией автопилота и моторизованных кресел с голосовым управлением для людей с ограниченными возможностями. Все более актуальным становится вопрос научно-обоснованной разработки отказоустойчивых систем управления. Поскольку целевыми устройствами нередко становятся коммерческие устройства массовой доступности, актуальным также является вопрос минимизации временных и материальных затрат на создание программного обеспечения (ПО) для них. Одним из наиболее эффективных подходов к повышению надежности ПО является введение избыточности. Известны следующие отказоустойчивые модели ПО на основе методологии: N-версионного программирования (NVP); N-версионного программирования с самопроверкой (NSCP); восстанавливающихся блоков (RB); согласованных восстанавливающихся блоков (CRB); $t/(n-1)$-версионного программирования [2]. Каждая из упомянутых моделей обладает своими преимуществами и недостатками. Выбор одной из них не является очевидным. Необходимо также определить число программных версий и требования по надежности как к этим версиям, так и к блоку принятия решения, к аппаратному обеспечению [3]. Для решения задачи сравнения и выбора той или иной отказоустойчивой модели ПО предложена имитационная среда исполнения мультиверсионного ПО, представленная в настоящей статье.

Методика сравнения отказоустойчивых моделей ПО

Для большинства прикладных систем управления возможна реализация всех основных моделей повышения отказоустойчивости путем введения программной избыточности [4]. Возникает задача выбора соответствующей модели либо по соотношению характеристик и затрат на разработку, либо по целевой характеристике, если не все модели позволяют достичь требуемого значения целевой характеристики. В любом случае необходим инструментарий, позволяющий оценить характеристики надежности ПО или его компонента при реализации всех основных отказоустойчивых моделей при заданных характеристиках программных модулей и блока принятия решения.

Предлагаемая методика позволяет оценивать все модели на одинаковом наборе ответов версий программ на каждой итерации, что дает возможность сравнить результаты работы всех моделей в одинаковых условиях, соответствующих заданным характеристикам системы. Это обстоятельство позволяет определить, какая из моделей будет лучше работать имен-

но в представленных условиях, а именно: при определенной надежности версий на каждом из изменяющихся потоков данных; с учетом того, с какой вероятностью будут возникать межверсионные ошибки или ошибки округления и с какой вероятностью будут проявляться сбои в приемочных тестах и т. д.

Модель $t/(n-1)$-версионного программирования

Отдельно следует остановиться на модели $t/(n-1)$ версионного программирования, поскольку алгоритм ее работы не описан в русскоязычной литературе, а сама модель представляет интерес. Процедура принятия решения в данной модели мультиверсионного ПО, предложенная Цзе Сюй (Jie Xu) из университета Ньюкастла (University of Newcastle upon Tyne), основана на $t/(n-1)$ диагностируемости [5]. Для простоты будем называть его $t/(n-1)$-алгоритмом принятия решений. Суть алгоритма состоит в том, что сравниваются не все выходы версий, а лишь некоторые из них, достаточные для принятия решения.

Рассмотрим пример системы с числом версий $N=5$ и максимальным числом ошибок $t=2$, т.е. рассмотрим вариант $2/(5-1)$-алгоритма. При числе ошибок, не превышающем t, алгоритм гарантирует выбор правильного варианта из N выходов версий. Однако и при превышении числа неправильных выходов версий система не обязательно выберет неверный суммарный выход. Может произойти и выбор правильного выхода, однако такой результат не гарантируется [5].

Рассмотрим алгоритм подробнее на данном примере. Сравниваются попарно выходы четырех версий: 1 — 2; 2 — 3; 3 — 4. Получаем три результата сравнений ω12, ω23, ω34, равные 0, если выходы совпадают, и равные 1, если различаются. На основании только этих трех результатов сравнения алгоритм принимает решение о переключении выхода между выходами 1-й, 4-й и 5-й версий, т. е. версии 2 и 3 используются только для сравнения, значения их выходов никогда не используется в качестве выхода системы. Более наглядно схема работы представлена на рис. 1.

Как видно из рис. 1, схема принятия решения в $t/(n-1)$-алгоритме относительно несложна. В случае пяти мультиверсий для принятия решения (правильного управления переключателем выходов) необходимы только результаты трех парных сравнений выходов четырех версий, значение выхода пятой версии для принятия решения не используется. Логика управления переключателем выходов на основании результатов сравнений при $n=5$ представлена в табл. 1.

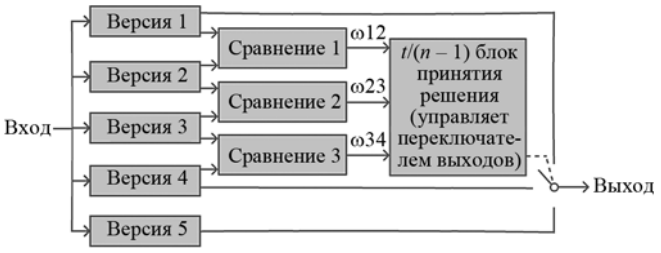

Рис. 1. Схема работы $t/(n-1)$ алгоритма при $n=5$ и $t=2$

Изучив табл. 1 и рис. 1, можно прийти к выводу, что при относительно надежных версиях в большей части случаев компараторы будут возвращать (0;0;0), и на выход будет подаваться значение выполнения первой версии. Можно также сделать вывод об отсутствии необходимости каждый раз исполнять пятую версию, а делать это только в случае соответствующих значений результатов сравнений, когда на выход необходимо подать именно результат пятой версии ((0;1;0), (1;0;1), (1;1;1)). Этот факт снижает среднюю нагрузку, требуемую среде исполнения мультиверсионного ПО для работы (в подавляющем большинстве случаев будут рассчитаны четыре из пяти версий).

Следует отметить, что и сам алгоритм принятия решения также является в значительной мере менее ресурсоемким по сравнению с голосованием, особенно со взвешенными его модификациями, где при каждом голосовании исполняются все версии, создаются классы и рассчитываются веса для каждого из них. Для $t/(n-1)$ при $n=5$ необходимо только три простые операции сравнения с бинарным выходом. Далее осуществляется однозначный, априорно заданный выбор выхода для одного из восьми возможных сочетаний значений выходов компараторов (табл. 1).

Таблица 1

Возможные варианты выбора на основе выходов компараторов для $n=5$

ω12	ω23	ω34	Предположительно правильные версии
0	0	0	1, 2, 3, 4
0	0	1	1, 2, 3
0	1	0	5
0	1	1	1, 2
1	0	0	2, 3, 4
1	0	1	5
1	1	0	3, 4
1	1	1	5

Программная реализация имитационной среды исполнения

Среда исполнения мультиверсионного ПО реализована в виде программного комплекса, который позволяет исполнять программные модули не только с помощью методологии *N*-версионного программирования (NVP), а также с использованием остальных распространенных мультиверсионных моделей, к числу которых относятся: модель восстанавливающихся блоков (RB), модель согласованных восстанавливающихся блоков (CRB); модель $t/(n-1)$-версионного программирования и модель мультиверсионного программирования с самопроверкой (NSCP).

В данной среде в качестве программных модулей выступают симуляции версий, которые работают в соответствии со следующими заданными в интерактивной форме характеристиками:

1) общая надежность версии в каждом потоке данных. В системе реализовано три последовательно изменяющих потока данных для исследования реакции системы на изменение работы программных модулей при различных входных данных;

2) вероятности возникновения межверсионной ошибки, "неточности". В частности, параметр допуска *E* для нечетких алгоритмов голосования используется симуляциями как значение разброса "неточностей".

В рассматриваемой среде исполнения симуляции версий дают следующие три типа ошибок: случайную ошибку, симулирующую сбой в модуле, межверсионную ошибку и неточность — ответ, близкий к правильному, т. е. удаленный от него на величину, не превышающую допуск, но не равный ему. Этот тип ошибки симулирует "неточность" — ошибки округления при нехватке разрядности, неточности оцифровки выходов аналоговых датчиков и т. д. Это ситуация, когда алгоритмически версия сработала верно, но дала неточный ответ из-за ошибок округления, оцифровки, нехватки разрядности, большой разницы в порядке величин при операциях с плавающей запятой [6].

Так как в среде исполнения ошибка проявляется с заданной для каждой версии и каждого потока данных вероятностью, то при возникновении ошибки выполняется ряд проверок. Если ошибка является не первой в текущем голосовании, то с заданной вероятностью генерируется межверсионная ошибка, т. е. возвращается значение, совпадающее со значением предыдущей ошибки. Этот тип ошибки симулирует связную ошибку — допущенный алгоритмический просчет, одинаковый в нескольких версиях, которые дадут одинаковую ошибку при одинаковом входе [7].

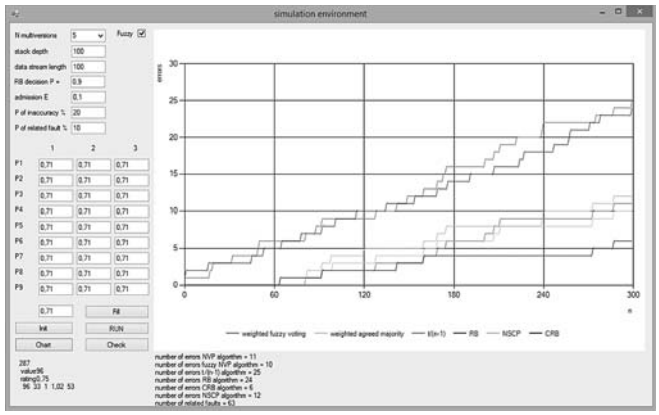

Рис. 2. Интерфейс программы с результатами моделирования: число ошибок в каждой модели и график их распределения по времени

"Неточность" также генерируется с заданной вероятностью. При этом возвращается выход, не равный правильному, но удаленный от него не более чем на заданное отклонение *E*. Если предыдущие вероятности не срабатывают, то возвращается случайная ошибка, симулирующая сбой в текущей версии модуля.

На рис. 2 представлен результат моделирования на основе 300 итераций мультиверсионного голосования с заданными в программе характеристиками модулей для варианта исполнения с симуляциями версий. Пользователем задается: число версий, глубина стека для взвешенных алгоритмов с забыванием, длина потока данных, вероятность ошибки приемочного теста алгоритмов с восстанавливающимися блоками, допуск вхождения для нечетких алгоритмов, вероятности возникновения неточностей и межверсионных ошибок, надежности каждой версии в каждом из трех потоков данных.

В предложенной программной реализации имеется возможность задавать значения надежности для всех версий, проводить инициализацию значений параметров системы, осуществлять моделирование с заданными параметрами, формирование графиков и проверку. Интерфейс программы имеет следующие области: графическая область, область вывода информации о последней возникшей ошибке и результатов моделирования, а именно общего числа накопленных ошибок за все итерации и числа межверсионных ошибок. Графики отображают распределение возникновения ошибок по каждой модели во времени.

Для практической проверки работоспособности предложенного инструментария в имитационной среде разработаны и используются в качестве версий пять различных алгоритмов

оптимизации: 1 — метод деления отрезка пополам; 2 — метод золотого сечения; 3 — метод дихотомии; 4 — метод квадратичной аппроксимации и 5 — метод Фибоначчи. В каждом алгоритме наложено ограничение на число итераций $K = 10$, поскольку в системах реального времени важно гарантированно получить ответ за заданный промежуток времени [8], т. е. нельзя допускать слишком длительного исполнения циклов. Это позволит дополнительно сравнить точность алгоритмов, которую они смогут достигнуть при введенном (одинаковом для всех версий) ограничении на число итераций.

В имитационной среде обеспечена поддержка принятия решения на основе любой из рассмотренных ранее моделей, однако в качестве примера будем рассматривать выходы модели N-версионного программирования с нечетким взвешенным голосованием согласованным большинством с забыванием. Отметим, что выбор нечеткого голосования обусловлен характером выходов версий — ожидаются ответы, близкие к реальным минимумам функций, но не обязательно равные им. Целесообразно использовать алгоритмы с элементом нечеткой логики [9], чтобы близкие ответы могли увеличивать веса мультиверсий, повышая устойчивость алгоритмов и модели в целом.

Включение в состав среды алгоритма взвешенного голосования позволяет нам численно сравнить алгоритмы оптимизации, так как по результатам прогонов мы будем накапливать их веса, рассчитывая оценку вероятности правильного ответа, основанную на статистике работы каждой версии. Чем чаще версия будет ошибаться, тем меньше будет ее вес. Оптимальной в нашем случае будет версия с наибольшей суммой весов, соответственно, оптимальным будет алгоритм, который реализуется данной версией.

В имитационной среде допускается возможность задавать глубину стека для взвешенного алгоритма голосования, значение допуска для элемента нечеткой логики и общее число прогонов. С учетом заданных значений указанных параметров имитационная среда осуществляет мультиверсионное исполнение всех алгоритмов и принятие решения, причем на каждой итерации изменяются значения коэффициентов функций, используемых в оптимизационных алгоритмах.

Более того, через каждые 100 итераций изменяется тип функции (в имитационной среде реализовано пять различных функций). Это позволяет исследовать и качество работы алгоритмов оптимизации, поскольку на различных типах функций используемые алгоритмы оптимизации показывают различные результаты. Итоговая оценка работы алгоритмов представляет собой сумму результатов работы по всем пяти функциям при 100 различных коэффициентах для каждой из них, что является достаточно объективным показателем и позволяет отсеять алгоритмы, работающие хорошо только в узком диапазоне вводимых параметров.

Таблица 2

Результат моделирования при различной надежности версий

Число ошибок	Надежность версий						
	0,7	0,75	0,8	0,85	0,9	0,95	0,99
Число ошибок в NVP с четким алгоритмом	10	6	3	1	0	0	0
Число ошибок в NVP с нечетким алгоритмом	12	9	4	1	0	1	0
Число ошибок в $t/(n-1)$-версионном программировании	28	14	7	3	1	0	0
Число ошибок в восстанавливающихся блоках (RB)	23	15	20	18	12	6	1
Число ошибок в согласованных RB	6	6	3	2	0	0	0
Число ошибок в NVP с самопроверкой	13	9	2	6	1	1	0
Число межверсионных ошибок	57	42	32	19	14	2	0

Анализ результатов моделирования в имитационной среде

При анализе результатов моделирования рассматривается зависимость числа ошибок для каждой отказоустойчивой модели ПО от надежности версий (программных модулей).

Установим следующие значения параметров системы: число мультиверсий = 5; глубина стека = 100; длина потока данных = 100; надежность приемочного теста = 0,9; допуск для нечетких алгоритмов = 0,1; вероятность возникновения неточности = 0,2; вероятность возникновения межверсионной ошибки = 0,1. Важно отметить, что в программе задаются не абсолютные вероятности конкретного типа ошибок, а условные, т. е. вероятность возникновения события ошибки. Например, общая вероятность возникновения неточности при работе программного модуля с надежностью $P = 0{,}95$ будет не 0,2, а $(1 - 0{,}95) \cdot 0{,}2 = 0{,}01$.

Итак, зафиксировав указанные параметры, мы можем менять значения надежности программных модулей (для простоты будем использовать равные значения надежности для всех версий и для всех трех потоков данных), используя кнопку "fill" (рис. 2). При ее нажатии все значения надежности версий приравниваются к одному значению, введенному в основном

окне программы, слева от кнопки. Проведем моделирование с различными значениями надежности версий от 0,7 до 0,95 с шагом 0,05 и 0,99.

Из результатов моделирования, приведенных в табл. 2, видно, как растет число межверсионных ошибок при снижении надежности версий. Виден рост числа допущенных ошибок, отличающийся для различных отказоустойчивых моделей ПО. Большое число ошибок зарегистрировано в модели восстанавливающихся блоков, отчасти это обусловлено относительно низкой заданной надежностью приемочного теста (0,9). Наилучшие результаты при невысокой надежности версий показывает модель согласованных восстанавливающихся блоков, поскольку она сочетает в себе механизмы как восстанавливающихся блоков, так и *N*-версионного программирования.

На рис. 3—5 (см. третью сторону обложки) и рис. 6 приведены результаты работы имитационной среды исполнения отказоустойчивого ПО с различными алгоритмами оптимизации при различных допусках в алгоритмах голосования.

Рассмотрим результаты работы имитационной среды с алгоритмами оптимизации на рис. 3 (см. третью сторону обложки). При допуске $E = 0{,}045$ есть однозначно лучший алгоритм, это метод 2 (метод золотого сечения). Он всегда голосует за выигравший класс, т. е. значения функции не являются равными, но отличаются не более чем на 0,045 в любую сторону. Таким образом, не обязательно иметь максимально близкий к правильному ответ, так как в соответствии с логикой работы взвешенного алгоритма голосования *1* в стек весов получают все версии с коэффициентом вхождения больше нуля, вне зависимости от его конкретного значения [10].

Очевидно, что значение допуска *E* будет существенно влиять на работу имитационной среды исполнения отказоустойчивых моделей ПО, поскольку определяет отклонение, которое может "допустить" версия и быть признанной верно ответившей. Рассмотрим случаи уменьшения допуска *E* до 0,001 на рис. 4 (см. третью сторону обложки) и его увеличения до значений 3 и 10 на рис. 5 (см. третью сторону обложки) и рис. 6 соответственно.

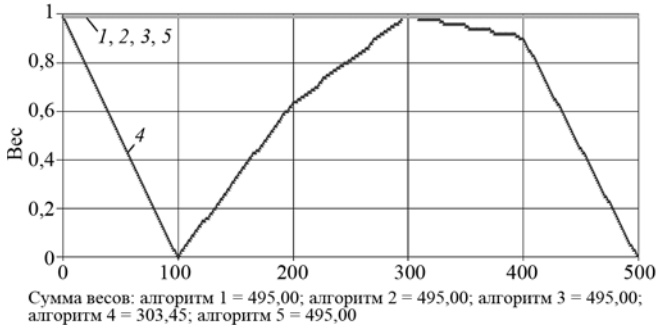

Рис. 6. Результат работы среды при E = 10:

1 — алгоритм 1; *2* — алгоритм 2; *3* — алгоритм 3; *4* — алгоритм 4; *5* — алгоритм 5

Из представленных результатов видно, что при уменьшении допуска суммы весов резко падают, поскольку алгоритм начинает все "строже" принимать решение о корректности выхода версии, помещая в стек весов "0" за отклонение, превышающее 0,001. При увеличении допуска наблюдается обратная картина, суммы весов растут, достигая максимального показателя в 495 уже для нескольких версий, поскольку даже алгоритмы, дающие ответы с существенным отклонением, признаются ответившими верно. В таком случае система уже не может однозначно выбрать наилучший алгоритм и выдает список субоптимальных алгоритмов, т. е. получивших максимальные суммы весов.

Заключение

Разработанная имитационная среда исполнения отказоустойчивых моделей ПО позволяет сравнить программные реализации алгоритмов, основанные на современных методологиях повышения отказоустойчивости с введением программной избыточности, в одинаковых условиях. Задавая характеристики имитационной среды и вероятности корректной работы всех программных компонентов, мы получаем результаты моделирования, которые упрощают выбор отказоустойчивой модели для реализации разрабатываемого ПО с избыточностью.

Исполнение в имитационной среде программных реализаций оптимизационных алгоритмов позволяет выбрать лучший алгоритм на заданном наборе функций с учетом допустимого отклонения, что показывает работоспособность предложенного инструментария на реальной прикладной задаче, когда заведомо правильный выход не известен, так как среда работает в реальным ПО, а не с симуляциями программных версий.

Список литературы

1. **Лосев В. В., Ковалев И. В.** Реинжиниринг информационного обеспечения интегрированных систем управления производством // Приборы. 2010. № 3 (117). С. 31—36.

2. **Jie Xu, Randell B.** Software fault tolerance: t/(n-1)-variant programming, Software fault tolerance: t/(n-1)-variant programming // IEEE Transactions on Reliability. 1997. Vol. 46, Iss. 1. P. 60—68.

3. **Ковалев И. В., Котенок А. В.** К проблеме выбора алгоритма принятия решения в мультиверсионных системах // Информационные технологии. 2006. № 9. С. 39—44.

4. **Ковалев И. В., Лосев В. В., Сарамуд М. В., Ковалев Д. И., Петросян М. О.** К вопросу реализации мультиверсионной среды исполнения бортового программного обеспечения автономных беспилотных объектов средствами операционной системы реального времени // Вестник СибГАУ. Красноярск, 2017. Т. 18, № 1. С. 58—61.

5. **Xu J.** The t(n-1)-diagnosability and its applications to fault tolerance // Digest of Papers. Fault-Tolerant Computing: The Twenty-First International Symposium. 1991. P. 496—503.

6. **Kovalev I., Voroshilova A., Losev V., Saramud M., Chuvashova M., Medvedev A.** Comparative Tests of Decision Making Algorithms for a Multiversion Execution Environment of the Fault Tolerance Software // Proc. of 2017 European Conference on Electrical Engineering and Computer Science (EECS 2017). 2017.

7. **Kovalev I., Losev V., Saramud M., Petrosyan M.** Model implementation of the simulation environment of voting algorithms, as a dynamic system for increasing the reliability of the control complex of autonomous unmanned objects // MATEC Web of Conferences 132, 04011 (2017).

8. **Стельмах В. О., Ковалев И. В.** Построение отказоустойчивых систем управления на основе мультиверсионного подхода // Информационно-телекоммуникационные системы и технологии (ИТСИТ-2012): Материалы Всерос. молодеж. конф. 2012. С. 172—173.

9. **Ковалев И. В.** и др. Оценка надежности АСУ с блокирующими модулями защиты // Приборы. 2013. № 6. С. 20—23.

10. **Kovalev I. V., Zelenkov P. V., Losev V. V., Kovalev D. I., Ivleva N. V., Saramud M. V.** Multiversion environment creation for control algorithm implementation by autonomous unpiloted objects] // IOP Conf. Series: Materials Science and Engineering 173 (2017) 012025. doi:10.1088/1757-899X/173/1/012025.

M. V. Saramud, Engineer, e-mail: msaramud@gmail.com, **I. V. Kovalev,** Professor, e-mail: kovalev.fsu@mail.ru,
V. V. Losev, Assistant Professor, e-mail: basilos@mail.ru,
M. O. Petrosyan, Postgraduate Student, e-mail: mopetrosyan@gmail.com,
A. O. Kalinin, Postgraduate Student, e-mail: andrey.kalinin@phkp.ru,
Reshetnev Siberian State University of Science and Technology, Krasnoyarsk, 660014, Russian Federation

Comparison of Methodologies for Increasing Software Fault Tolerance in a Simulation Execution Environment

In the article considered technique which makes it possible to compare the main methodologies for increasing fault tolerance with the introduction of software redundancy: N-version programming (with majority and fuzzy voting algorithms), N self-checking programming, recovery blocks, consensus recovery blocks, and t/(n − 1)-version programming. The software implementation of this technique allowed to compare them under the same conditions, with the specified system characteristics and the probabilities of correct operation of all components, to obtain system characteristics using these methodologies based on the simulation results. The results of modeling in the proposed environment are analyzed. A non-trivial methodology of t/(n − 1)-Variant Programming based on t/(n − 1)-diagnosability is considered in detail. The results of it's work are compared with classical methodologies. This simplifies the choice of methodology for implementing the software under development. A software implementation with real algorithms allows you to select the best optimization algorithm for a given set of functions and the amount of allowable deviation. Also it shows the working capacity of the proposed toolkit on a real applied task, where the system no longer knows the correct output, but works in real conditions, rather than with simulation versions.

Keywords: *multi-version programming, software redundancy, reliability, recovery blocks, execution environment, related software fault, t/(n − 1)-diagnosability, consensus recovery blocks, optimization, simulation*

DOI: 10.17587/it.25.20-25

References

1. **Losev V. V., Kovalev I. V.** Reengineering information support of integrated systems of production management, *The Devices*, 2010, no. 3 (117), pp. 31—36 (in Russian).

2. **Jie Xu, Randell B.** Software fault tolerance: t/(n-1)-variant programming, *IEEE Transactions on Reliability*, 1997, vol. 46, iss. 1, pp. 60—68.

3. **Kovalev I. V., Kotenok A. V.** Choice Problems the Decision Algorithm for Multi- Version Systems, *Informacionnye Tekhnologii*, 2006, no. 9, pp. 39—44 (in Russian).

4. **Saramud M. V., Losev V. V., Kovalev D. I., Petrosyan M. O.** To the question of implementation of multiversion execution environment software of onboard autonomous pilotless objects by means of real-time operating system, *Vestnik SibGAU*, 2017, vol. 18, no. 1, pp. 58—61(in Russian).

5. **Xu J.** The t(n-1)-diagnosability and its applications to fault tolerance, *Digest of Papers. Fault-Tolerant Computing: The Twenty-First International Symposium*, 1991, pp. 496—503.

6. **Kovalev I., Voroshilova A., Losev V., Saramud M., Chuvashova M., Medvedev A.** Comparative Tests of Decision Making Algorithms for a Multiversion Execution Environment of the Fault Tolerance Software, *Proceedings of 2017 European Conference on Electrical Engineering and Computer Science (EECS 2017),* 2017.

7. **Kovalev I., Losev V., Saramud M., Petrosyan M.** Model implementation of the simulation environment of voting algorithms, as a dynamic system for increasing the reliability of the control complex of autonomous unmanned objects, *MATEC Web of Conferences* 132, 04011 (2017).

8. **Stel'makh V. O., Kovalev I. V.** Building fault-tolerant control systems based on the multi-version approach, *Materialy vserossiiskoi molodezhnoi konferentsii "Informatsionno-telekommunikatsionnye sistemy i tekhnologii (ITSIT-2012)*", 2012, pp. 172—173(in Russian).

9. **Kovalev I.** Evaluation of the reliability of ACS with blocking protection modules, *The Devices,* 2013, no. 6, pp. 20—23 (in Russian).

10. **Kovalev I. V., Zelenkov P. V., Losev V. V., Kovalev D. I., Ivleva N. V., Saramud M. V.** Multiversion environment creation for control algorithm implementation by autonomous unpiloted objects, *IOP Conf. Series: Materials Science and Engineering* 173 (2017) 012025, doi:10.1088/1757-899X/173/1/012025.