N. N. Goryashin, A. S. Sidorov

ANALYSIS OF OPERATING MODES OF HIGH POWER LED LIGHT SOURCES SUPPLIED BY SWITCHING CONVERTER

The paper presents results of experimental investigation of operation of high power LEDs supplied by switching converter with output current regulation. Dynamic performance of two LEDs is investigated. Influence of LED dynamic characteristics on operating mode of converter output filter is analyzed.

Keywords: LED, switching power converter.

УДК 681.3

В. Г. Жуков, Д. А. Шеенок, В. А. Терсков

ПОВЫШЕНИЕ НАДЕЖНОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СЛОЖНЫХ СИСТЕМ

Описывается модель оценки надежности программного обеспечения со сложной архитектурой, а также возможные варианты построения мультиверсионных модулей. Рассматриваются изменяемые характеристики программной архитектуры, оценивается мощность пространства ее оптимизации.

Ключевые слова: оптимизация, программное обеспечение, надежность, информационная безопасность.

Основное внимание в теории и практике обеспечения информационной безопасности (ИБ) применения информационных технологий и систем сосредоточено на защите от деструктивных действий злоумышленника, искажения и хищения программных средств и информации баз данных. Однако при любом виде деятельности людям в процессе создания или применения изделий или систем свойственно непредумышленно ошибиться. В общем случае под ошибкой подразумевается дефект, погрешность или неумышленное искажение объекта или процесса. При этом предполагается, что известно правильное, эталонное состояние объекта, по отношению к которому может быть определено наличие отклонения – ошибки. В связи с этим дефекты функционирования информационных систем (ИС), которые не связаны с действиями злоумышленника и проявляются внешне как случайные, имеют разную природу и последствия, вплоть до катастрофических, соответствующих нарушению информационной безопасности использования ИС.

Наиболее полно ИБ ИС характеризует величина ущерба, который может возникнуть при проявлении дестабилизирующих факторов и реализации конкретных угроз ИБ, а также среднее время между проявлениями угроз. Однако описать и измерить в достаточно общем виде возможный ущерб при нарушении ИБ для критических ИС разных классов очень сложно, поэтому реализации угроз целесообразно характеризовать интервалами времени между их проявлениями, или наработкой на отказы. Это сближает понятия и характеристики степени защищенности с показателями надежности ИС.

Наиболее общим видом ресурсов, используемых при создании ИС, являются допустимые финансово-экономические затраты, или сметная стоимость разработки. При анализе защищенности этот показатель может применяться или как вид ресурсных ограничений, или как критерий оптимизации [1].

В настоящее время в качестве возможной альтернативы методам тестирования и верификации программ можно рассматривать подход мультиверсионного отказоустойчивого программирования, который обеспечивает высокий уровень надежности функционирования критичного программного обеспечения (ПО). Использование систем поддержки принятия решений при мультиверсионном программировании на этапе формирования надежного ПО позволяет уделять основное внимание формированию требований к его качеству. Однако улучшение характеристик надежности ПО с применением избыточности требует дополнительных временных и финансовых затрат. Поэтому основной вопрос на этом этапе заключается в том, каким образом, используя избыточность в архитектуре ПО, можно максимизировать надежность и снизить стоимость разработки. Этот вопрос может быть решен с помощью многочисленных методов многокритериальной поддержки принятия решений, ориентированных на задачи с дискретным пространством решений и учитывающих различные уровни информации о предпочтениях эксперта. Существующие методы определения глубины мультиверсионности и многокритериального принятия решений при выборе архитектуры позволяют спроектировать программную систему, отвечающую предъявляемым к ней требованиям [2].

Надежность архитектуры включает в себя как надежность центрального ядра системы, так и надежность индивидуальных компонентов, предоставляемых пользователю. Сбой отдельного компонента может привести к неработоспособности не только этого, но и, возможно, других компонентов ПО, однако это не приведет к неработоспособности всей системы в целом.

Тщательный анализ программной архитектуры позволяет выявить компоненты, ошибки в которых оказывают самое существенное влияние на надежность системы. Как правило, это компоненты, наиболее часто используемые или архитектурно связанные со множеством других компонентов, в связи с чем они более других подходят для разработки методом мультиверсионного программирования [3].

В зависимости от количества и величины компонентов условные и безусловные вероятности сбоя, доступа, анализа и времени восстановления, а также времени использования компонентов будут различными. Модель, приведенная в [4–6], может использоваться для оценки надежности ПО при возможных архитектурных изменениях и выбора надежной архитектуры из различных вариантов. В этой модели введены следующие обозначения:

1) M – число уровней в архитектуре ПО;

2) N_j – число компонентов на уровне j, $j \in \{1, ..., M\}$;

3) D_{ij} – множество индексов компонентов, зависящих от компонента i на уровне j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$;

4) F_{ij} – событие сбоя, произошедшего в компоненте i на уровне j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$;

5) PU_{ij} – вероятность использования компонента i на уровне j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$;

6) PF_{ij} – вероятность появления сбоя в компоненте i на уровне j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$;

7) $PL^{ij}{}_{nm}$ – условная вероятность появления сбоя в компоненте m на уровне n при появлении сбоя в компоненте i на уровне j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$, $n \in \{1, ..., N_m\}$, $m \in \{1, ..., M\}$;

8) TA_{ij} – относительное время доступа к компоненту i на уровне j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$, определяемое как отношение среднего времени доступа к компоненту i на уровне j к числу сбойных компонентов на малых уровнях архитектуры за одно и то же время;

9) TC_{ij} – относительное время анализа сбоя в компоненте i на уровне j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$, определяемое как отношение среднего времени анализа сбоя в компоненте i на уровне j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$, к числу сбойных компонентов на всех уровнях архитектуры, анализируемых в одно и то же время;

10) TE_{ij} – относительное время устранения сбоя в компоненте i на уровне j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$, определяемое как отношение среднего времени восстановления в компоненте i на уровне j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$, к числу сбойных компонентов на всех уровнях архитектуры, в которых устранение сбоев происходит в одно и то же время;

11) TU_{ij} – относительное время использования компонента i на уровне j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$, определяемое как отношение среднего времени использования компонента i на уровне j, $i \in \{1, ..., N_j\}$, $j \in \{1, ..., M\}$, к числу компонентов на всех уровнях архитектуры, применяемых в одно и то же время;

12) TR – среднее время простоя системы в большой архитектуре телекоммуникационного ПО реального времени, определяемое как время, в течение которого система не может выполнять свои функции;

13) $MTTF$ – среднее время появления сбоя в большой архитектуре телекоммуникационного ПО реального времени, определяемое как время, в течение которого сбоев в системе не происходит.

В архитектуре реального ПО среднее время простоя системы определяется по формуле

$$TR = \sum_{j=1}^{j=M} \sum_{i=1}^{i=Mj} \{PU_{ij} \times PF_{ij} \times [(TA_{ij} + TC_{ij} + TE_{ij}) + \\ + \sum_{(m=1)\&(m\neq j)}^{m=M} \sum_{n=1}^{n=Nn} [PL^{ij}_{nm} \times [(TA_{nm} + TC_{nm} + TE_{nm}) + \\ + \sum_{l \in Dnm} [PL^{nm}_{lm} \times (TA_{lm} + TC_{lm} + TE_{lm})]]] + \\ + \sum_{k \in Dij} [PL^{ij}_{kj} \times [(TA_{kj} + TC_{kj} + TE_{kj}) + \\ + \sum_{(m=1)\&(m\neq j)}^{m=M} \sum_{n=1}^{n=Nm} [PL^{kj}_{nm} [(TA_{nm} + TC_{nm} + TE_{nm}) + \\ + \sum_{l \in Dnm} [PL^{nm}_{lm} \times (TA_{lm} + TC_{lm} + TE_{lm})]]]]]]\},$$

а среднее время сбоя – по формуле

$$MTTF = \sum_{j=1}^{j=M} \sum_{i=1}^{i=Nj} \{PU_{ij} \times (1 - PF_{ij}) \times [TU_{ij} + \\ + \sum_{(m=1)\&(m\neq j)}^{m=M} \sum_{n=1}^{n=Nm} [(1 - PL^{ij}_{nm}) \times \\ \times [TU_{nm} + \sum_{l \in Dnm} [(1 - PL^{nm}_{lm}) \times TU_{lm}]]] + \\ + \sum_{k \in Dij} [(1 - PL^{ij}_{kj}) \times [TU_{kj} + \sum_{(m=1)\&(m\neq j)}^{m=M} \sum_{n=1}^{n=Nm} [(1 - PL^{kj}_{nm}) \times \\ \times [TU_{nm} + \sum_{l \in Dnm} [(1 - PL^{nm}_{lm}) \times TU_{lm}]]]]]\}.$$

Среднее время простоя системы и среднее время сбоя могут быть использованы для предсказания надежности системы, состоящей из разных уровней архитектуры [4].

Для случая непрерывной эксплуатации сложного ПО анализ его готовности можно производить по формуле коэффициента готовности S:

$$S = MTTF/(TR + MTTF).$$

Однако рассмотренная выше модель не предлагает оптимальную надежную архитектуру ПО. Возможный выбор такой архитектуры при разработке больших систем телекоммуникационного ПО в реальном мас-

штабе времени ограничен многими факторами, в частности аппаратной архитектурой, составляющими зависимостями, которые не могут быть изменены, архитектурными уровнями, которые не могут быть разделены или объединены, кодом компонента, который невозможно изменить, и, возможно, другими ограничениями [4].

В [4] описана усовершенствованная модель, в которой при оценке стоимости разработки C_s учитывается мультиверсионный подход и стоимость разработки всей системы рассчитывается как сумма стоимости разработки каждой версии компонента:

$$C_s = \sum_{j=1}^{M} \sum_{i=1}^{N} \sum_{k \in Z_{ij}}^{K} C_{ij}^k,$$

где Z_{ij} – множество версий компонента i, $i = 1, \ldots, N$, на уровне $j = 1, \ldots, M$, $k = 1, \ldots, K$; C^k_{ij} – стоимость разработки версии k компонента i на уровне j, $k \in Z_{ij}$, в денежных единицах.

Очевидно, что приведенные выше характеристики: коэффициент готовности системы S и стоимость разработки системы C_s – являются противоречивыми критериями оптимальности. Поэтому далее для критерия, отражающего затраты ресурсов на разработку, вместо стоимости будем использовать трудоемкость:

$$T_s = \sum_{j=1}^{M} \sum_{i=1}^{N} \sum_{k \in Z_{ij}}^{K} T_{ij}^k.$$

В [7] приводятся два метода построения архитектуры, основанных на мультиверсионном подходе: NVP и RB.

При создании мультиверсионного компонента из K версий методом N-версионного программирования (N-Version Programming, NVP) надежность для любого K находится по формуле

$$R_{ij} = p^v_{ij} \left(1 - \prod_{k \in Z_{ij}}^{K} (1 - p^k_{ij}) \right),$$

где p^v_{ij} – вероятность сбоя алгоритма голосования; p^k_{ij} – вероятность сбоя версии $k \in Z_{ij}$.

При построении мультиверсионного компонента из K версий методом блока восстановления (Recovery Block, RB) надежность определяется следующим образом:

$$R_{ij} = \sum_{k \in Z_{ij}}^{K} p^k_{ij} p^{AT}_{ij} \prod_{l=1}^{k-1} \left((1 - p^l_{ij}) p^{AT}_{ij} + p^l_{ij} (1 - p^{AT}_{ij}) \right),$$

где p^k_{ij} – вероятность сбоя версии $k \in Z_{ij}$; p^{AT}_{ij} – вероятность безотказной работы приемочного теста для компонента i, $i = 1, \ldots, N$, на уровне j, $j = 1, \ldots, M$. Вероятность сбоя таких компонентов рассчитывается как $PF_{ij} = 1 - R_{ij}$.

Каждый программный компонент архитектуры или каждая его версия также могут быть доведены до определенного уровня надежности с помощью тестирования. С помощью статистических моделей, например модели роста надежности (Software Growth Reliability Model, SGRM) [5] или экспертной оценки, можно спрогнозировать трудозатраты и достигнутый уровень надежности программного компонента. Таким образом, у аналитика или проектировщика ПО возникает выбор из нескольких вариантов уровней надежности проектируемого компонента и соответствующих этим уровням трудозатрат.

Обобщая сказанное выше, приведем следующую усовершенствованную модель:

1) M – число архитектурных уровней в архитектуре ПО;

2) N_j – число компонентов на уровне j, $j \in \{1, \ldots, M\}$;

3) D_{ij} – множество индексов компонентов, зависящих от компонента i на уровне j, $i \in \{1, \ldots, N_j\}$, $j \in \{1, \ldots, M\}$;

4) F_{ij} – событие сбоя, произошедшего в компоненте i на уровне j, $i \in \{1, \ldots, N_j\}$, $j \in \{1, \ldots, M\}$;

5) PU_{ij} – вероятность использования компонента i на уровне j, $i \in \{1, \ldots, N_j\}$, $j \in \{1, \ldots, M\}$;

6) PF_{ij} – вероятность появления сбоя в компоненте i на уровне j, $i \in \{1, \ldots, N_j\}$, $j \in \{1, \ldots, M\}$;

7) PL^{ij}_{nm} – условная вероятность появления сбоя в компоненте m на уровне n при появлении сбоя в компоненте i на уровне j, $i \in \{1, \ldots, N_j\}$, $j \in \{1, \ldots, M\}$, $n \in \{1, \ldots, N_m\}$, $m \in \{1, \ldots, M\}$;

8) TA_{ij} – относительное время доступа к компоненту i на уровне j, $i \in \{1, \ldots, N_j\}$, $j \in \{1, \ldots, M\}$, определяемое как отношение среднего времени доступа к компоненту i на уровне j к числу сбойных компонентов на малых уровнях архитектуры за одно и то же время;

9) TC_{ij} – относительное время анализа сбоя в компоненте i на уровне j, $i \in \{1, \ldots, N_j\}$, $j \in \{1, \ldots, M\}$, определяемое как отношение среднего времени анализа сбоя в компоненте i на уровне j, $i \in \{1, \ldots, N_j\}$, $j \in \{1, \ldots, M\}$, к числу сбойных компонентов на всех уровнях архитектуры, анализируемых в одно и то же время;

10) TE_{ij} – относительное время устранения сбоя в компоненте i на уровне j, $i \in \{1, \ldots, N_j\}$, $j \in \{1, \ldots, M\}$, определяемое как отношение среднего времени восстановления в компоненте i на уровне j, $i \in \{1, \ldots, N_j\}$, $j \in \{1, \ldots, M\}$, к числу сбойных компонентов на всех уровнях архитектуры, в которых происходит устранение сбоев в одно и то же время;

11) TU_{ij} – относительное время использования компонента i на уровне j, $i \in \{1, \ldots, N_j\}$, $j \in \{1, \ldots, M\}$, определяемое как отношение среднего времени использования компонента i на уровне j, $i \in \{1, \ldots, N_j\}$, $j \in \{1, \ldots, M\}$, к числу компонентов на всех уровнях архитектуры, применяемых в одно и то же время;

12) Z_{ij} – множество версий компонента i на уровне j, $k = 1, \ldots, K$;

13) T^k_{ij} – трудоемкость разработки версии k компонента i на уровне j, $k \in Z_{ij}$, человеко-часов;

14) T_{ij} – трудоемкость разработки приемочного теста (для метода RB) или алгоритма голосования (для метода NVP);

15) T_s – общая трудоемкость системы;

16) B_{ij} – дихотомическая переменная, принимающая значение 1 (тогда $NVP_{ij} = 0$, $RB_{ij} = 0$), если в про-

граммном компоненте не используется программная избыточность, в противном случае она равна 0;

17) NVP_{ij} – дихотомическая переменная, принимающая значение 1 (тогда $B_{ij} = 0$, $RB_{ij} = 0$), если в программном компоненте используется программная избыточность по методу *N*-версионного программирования, в противном случае она равна 0;

18) RB_{ij} – дихотомическая переменная, принимающая значение 1 (тогда $B_{ij} = 0$, $NVP_{ij} = 0$), если в программном компоненте используется программная избыточность по методу блока восстановления, в противном случае она равна 0;

19) *TR* – среднее время простоя системы в большой архитектуре телекоммуникационного ПО реального времени, определяемое как время, в течение которого система не может выполнять свои функции;

20) *MTTF* – среднее время появления сбоя в большой архитектуре телекоммуникационного ПО реального времени, определяемое как время, в течение которого сбоев в системе не происходит;

21) *S* – коэффициент готовности.

Рассмотрим параметры усовершенствованной модели, поддающиеся изменению, для поиска оптимальной архитектуры.

Надежность программного компонента можно повысить, используя программную избыточность, с помощью метода NVP или RB. Поэтому введем переменную $ARC_{ij} \in \{0$ – при NVP, 1 – при RB, 2 – для обычных компонентов$\}$.

При $ARC_{ij} = 0$; 1 возможна разработка Z_{ij} версий мультиверсионного компонента *i* на уровне *j*. В [6] рассматривается программная избыточность мультиверсионного компонента от двух до пяти версий.

Каждую версию компонента *i* на уровне *j* можно довести до заданного уровня надежности p^k_{ij}, затратив на ее разработку и тестирование некоторое количество ресурсов. Трудоемкость можно рассчитать с помощью статистических моделей надежности (например, модели SGRM). Множество различных вариантов надежности Var_{ij} компонента *i* на уровне *j* (или его версии, если была применена программная избыточность) и соответствующие этим вариантам трудозатраты определяются аналитиком или менеджером проекта. Например, для достижения 99%-го уровня надежности версии компонента необходимо затратить 4 человеко-часов, а для достижения 95%-го уровня надежности достаточно 2 человеко-часа. Тогда $VarR^1_{ij} = 0{,}99$, $VarT^1_{ij} = 4$, $VarR^2_{ij} = 0{,}95$, $VarT^2_{ij} = 2$. Для каждого компонента или его версии варианты надежности и соответствующей трудоемкости будут одинаковыми, так как эти версии разрабатываются по одной спецификации, но разными командами разработчиков или с помощью разных средств разработки.

Если принимается решение о введении программной избыточности, то в зависимости от выбранного метода разработки необходимо создание своего метакласса мультиверсионности – приемочного теста (для метода RB) или алгоритма голосования (для метода NVP). Множество вариантов надежности для p^{AT}_{ij} или p^v_{ij} и соответствующей им трудоемкости вводиться не будут, так как на практике приемочные тесты и алгоритмы голосования должны быть не слишком сложными, но обладающими абсолютной надежностью.

Задача оптимизации сводится к поиску максимума функции коэффициента готовности при ограничении на ресурсы, затрачиваемые на разработку или модификацию программной системы:

$$S \to \max,\ S \ge S^0,$$
$$T \to \min,\ T_s \le T_s^0,$$

где *S* – критерий оценки коэффициента готовности системы; T_s – критерий оценки трудоемкости разработки системы; S^0, T_s^0 – предельные допустимые уровни критериев.

Оценим мощность пространства оптимизации архитектуры программной системы, взяв для примера упрощенную ситуацию. Пусть при модификации программной системы необходимо разработать 10 новых программных компонентов. Перед системным аналитиком возникает задача выбора оптимальной архитектуры новых компонентов, чтобы надежность всей системы удовлетворяла требованиям заказчика, а трудоемкость разработки и тестирования соответствовала возможностям компании-разработчика.

В ходе работы системный аналитик определил, что в половину новых компонентов можно ввести программную избыточность, т. е. для них $ARC_{ij} \in \{0,1,2\}$, а при $ARC_{ij} \in \{0,1\}$ может быть введено не более трех версий. Для каждой версии определены по два варианта надежности с соответствующей трудоемкостью. Для половины компонентов, в которые не вводится программная избыточность, можно изменить только варианты надежности с соответствующей трудоемкостью. Тогда общее количество комбинаций вычисляется следующим образом:

$$N = \left(\mathrm{Var} + 2 \sum_{\mathrm{Ver}=2}^{L} \mathrm{Var}^{\mathrm{Ver}} \right)^D + \mathrm{Var}^F,$$

где *N* – количество комбинаций; Var – количество вариантов надежности с соответствующей трудоемкостью; Ver – количество версий при введении избыточности, изменяется от двух и более; *L* – предельно допустимое количество версий; *D* – количество программных компонентов, в которых возможно введение избыточности; *F* – количество программных компонентов, в которых введение избыточности невозможно. Для нашего примера Var = 2, *L* = 3, *D* = 5, откуда $N = 1{,}2 \cdot 10^7$.

Зависимости *N*(*D*), *N*(*F*) и *N*(*L*) в логарифмической шкале по оси *N* при тех же начальных условиях для параметров Var, Ver, *F*, *D* представлены ниже (рис. 1–3).

Анализ графиков на рис. 1 и 2 показывает, что увеличение количества программных компонентов, в которых возможно введение избыточности, влияет на рост мощности пространства оптимизации значительно больше, чем увеличение количества компонентов, в которых введение избыточности не предусматривается.

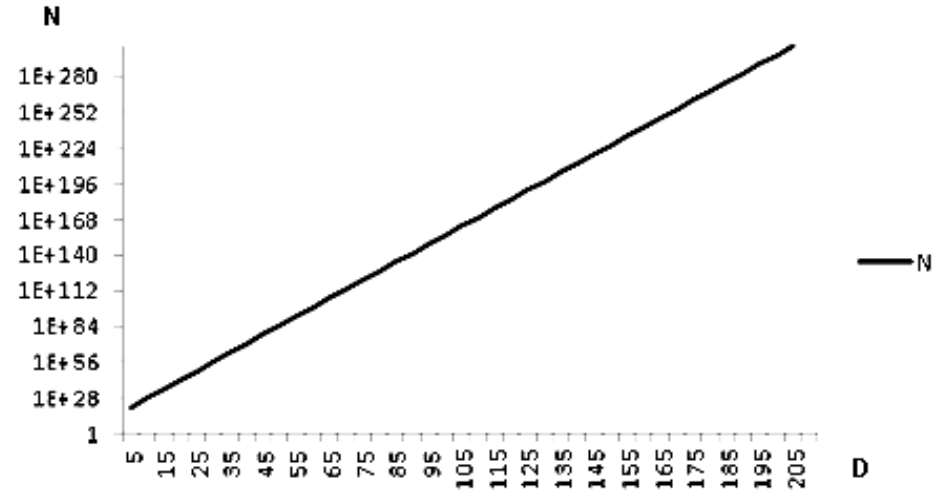

Рис. 1. Зависимость количества комбинаций от количества компонентов, в которых возможна избыточность

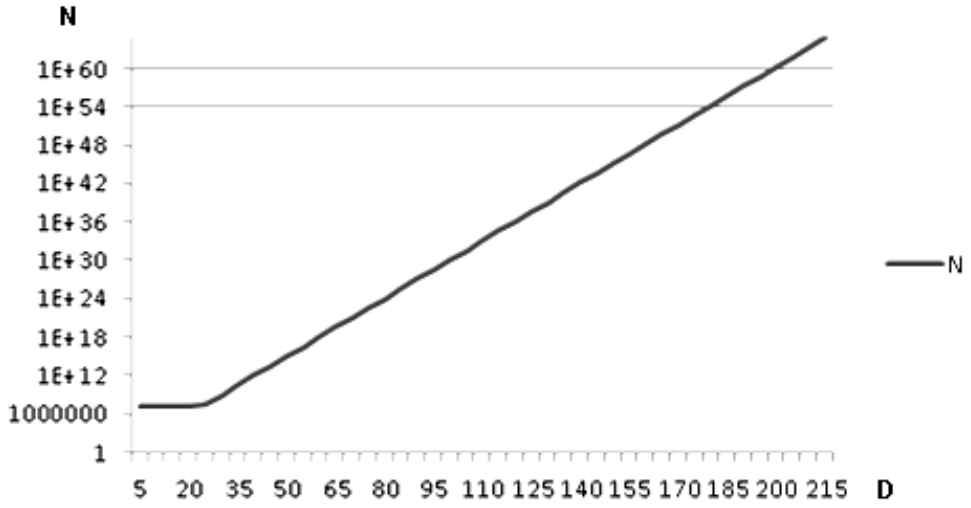

Рис. 2. Зависимость количества комбинаций от количества компонентов,
в которых невозможно введение избыточности

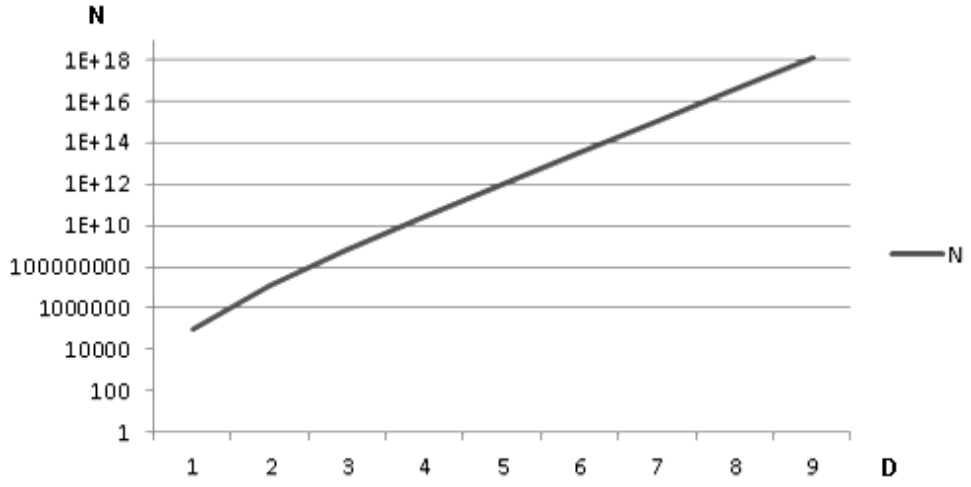

Рис. 3. Зависимость количества комбинаций от предельного количества версий

В рассмотренном нами примере выбрано достаточно малое количество компонентов, поэтому полученная размерность пространства оптимизации не слишком велика. Но если учесть, что функции коэффициента готовности и трудоемкости рассчитываются на персональном компьютере со средними характеристиками как минимум за 0,5 с, то перебор всех комбинаций даже для такого простого примера займет несколько месяцев.

Увеличение аппаратных характеристик персонального компьютера может незначительно снизить время вычислений, как и существенное ограничение критериев параметрами S^0 и T_s^0, которое также не позволит значительно сократить количество поисковых точек для сложных программных систем, имеющих большое количество компонентов в архитектуре. Это означает, что для решения поставленной задачи могут быть использованы только эвристические алгоритмы усеченного перебора, например генетические алгоритмы. Но даже тогда поиск решения такой задачи потребует значительных временных затрат. Эта проблема может быть разрешена за счет применения модификаций генетического алгоритма, такие как гибридные алгоритмы с локальным и глобальным поиском или коэволюционные алгоритмы [8–10]. Для проверки разработанного алгоритма должна быть создана тестовая задача с известным решением, которое алгоритм должен находить за приемлемое время.

Таким образом, нами был выполнен анализ параметров, поддающихся изменению при поиске оптимальной архитектуры программных систем, произведена оценка пространства оптимизации и определено влияние количества параметров архитектуры на рост мощности пространства оптимизации.

Библиографические ссылки

1. Липаев В. В. Функциональная безопасность программных средств [Электронный ресурс] // URL: http://sec.bl.by/articles/178811.php (дата обращения: 20.11.2012).

2. Царев Р. Ю. Система многоатрибутивного формирования мультиверсионных программных средств отказоустойчивых систем управления : дис. … канд. техн. наук. Красноярск, 2003.

3. Дегтерев А. С., Русаков М. А. Архитектурная надежность программного обеспечения информационно-телекоммуникационных технологий // Приоритетные направления развития науки, технологий и техники : сб. материалов Всерос. заоч. электрон. конф. / Рос. акад. естеств. наук. М., 2005. С. 161–162.

4. Русаков М. А. Многоэтапный анализ архитектурной надежности в сложных информационно-управляющих системах : дис. … канд. техн. наук. Красноярск, 2005.

5. Юнусов Р. В. Система модельно-алгоритмической поддержки многоэтапного анализа надежности программных средств : автореф. дис. … канд. техн. наук. Красноярск, 2004.

6. Привалов А. С. Система формирования гарантоспособной программной архитектуры для АСУ ТП : дис. … канд. техн. наук. Красноярск, 2000.

7. Новой А. В. Система анализа архитектурной надежности программного обеспечения : дис. … канд. техн. наук. Красноярск, 2011.

8. Жуков В. Г., Жукова М. Н. Исследование коэволюционного алгоритма решения нестационарных задач оптимизации // Вестник СибГАУ. 2006. Вып. 1 (8). С. 27–30.

9. Сергиенко Р. Б., Семенкин Е. С. Коэволюционный алгоритм для задач условной и многокритериальной оптимизации // Прогр. продукты и системы. 2010. № 4. С. 24–28.

10. Жуков В. Г., Паротькин Н. Ю. Дифференцированный адаптивный генетический алгоритм // Вестн. Новосиб. гос. ун-та. Серия: Информационные технологии. Новосибирск, 2011. Т. 9, вып. 1. С. 5–11.

V. G. Zhukov, D. A. Sheenok, V. A. Terskov

RELIABILITY IMPROVEMENT OF COMPLEX NETWORKS SOFTWARE

The authors describe a model of estimation of reliability of software of complex architecture and options for the construction of multi-version modules and consider variable characteristics of software architecture and estimate cardinality of its optimization.

Keywords: optimization, software, reliability, information security.