V.A. Terskov, A.S. Timohovich, D.A. Sheenok

Secutity operation of the software systems on design stage

Keywords: security information technologies, genetic algorithm, software architecture

Design of software architecture – an important and one of first stages in the life cycle of failover software. At this stage is determined the depth of software redundancy and planned costs of achieving the required level of reliability of software components. Number of alternatives to the construction of architecture is quite large and depends on the number of developed components. Offer a solution to this problem using a specialized genetic algorithm.

В.А. Терсков, А.С. Тимохович, Д.А. Шеенок

ОБЕСПЕЧЕНИЕ БЕЗОПАСНОСТИ ФУНКЦИОНИРОВАНИЯ ПРОГРАММНЫХ СИСТЕМ ПРИ ПРОЕКТИРОВАНИИ

Введение

Информационных технологии широко внедряются в жизнь современного общества. Это приводит к появлению ряда проблем безопасности их функционирования.

Для безопасного функционирования информационной системы необходимо обеспечить следующие ее свойства [1]:

1. Целостность – решения должны приниматься на основе информации, которая является достоверной, точной и защищенной от возможных намеренных и непреднамеренных искажений;
2. Доступность (готовность) – информация и модули, ее предоставляющие должны быть доступны, т.е. готовы к работе в момент обращения к ним;
3. Конфиденциальность – возможность получения информации должна быть только у тех лицам, для которых эта информация предназначена.

На реальные программные системы воздействуют непреднамеренные дестабилизирующие факторы, которые способны вызвать аномалии функционирования и даже катастрофические последствия, порой более тяжелые, чем последствия преднамеренных действий. Технологические аварии и катастрофы могут быть результатом принципиальных алгоритмических ошибок проектирования. Также требуются адекватные методы и средства защиты от их влияния. В теории и практике сложилось особое направление по обеспечению безопасности информационных систем при случайных дестабилизирующих воздействиях в отсутствии злоумышленного влияния.

Поэтому необходимо применение методов выявления и предотвращения непредумышленных угроз безопасности функционирования программных средств, снижения соответствующих рисков до допустимого уровня и определения реального достигнутой степени безопасности использования информационных систем. Другими словами необходимо обеспечение алгоритмической и программно-технологической безопасности еще на стадии проектирования информационной системы [1].

Одним из самых первых этапов проектирования информационной системы является проектирование ее архитектуры. На данном этапе определяется состав модулей, из взаимосвязи, глубина программной избыточности. Уже на этой стадии учитываются ограничения на трудовые ресурсы. Количество альтернатив построения архитектуры достаточно велико и зависит от количества разрабатываемых компонентов и их возможных связей. Поэтому для проектировщика возникает задача оптимизации характеристик архитектуры таким образом, чтобы была реализована достаточно безопасная программная система с наименьшими затратами ресурсов.

Задачи оптимизации постоянно возникают в деятельности человека, в частности при проектировании сложных технических систем. Если существует возможность выбора параметров такой системы, то их следует выбрать оптимальным образом. Многие возникающие задачи оптимизации характеризуются такими свойствами, как алгоритмическое задание целевой функции, наличие большого количества локальных экстремумов, большая размерность и различный характер параметров.

Важным классом методов, способных решать такие задачи оптимизации являются эволюционные алгоритмы и, в частности, генетические алгоритмы (ГА).

Постановка задачи. Для сложных информационных систем, используемых в критических областях (технологические процессы, транспорт, финансовые операции) возникает задача построения такой архитектуры программной системы, чтобы система была технологически безопасной и ее реализация требовала минимальных трудозатрат. Таким образом, задача сводится к поиску максимума функции коэффициента готовности проектируемой системы и минимума функции ресурсов, затрачиваемых на разработку или модификацию программной системы:

$$S \to \max,\ T_s \to \min,$$

где S – критерий оценки коэффициента готовности системы, T_s – критерий оценки трудоемкости разработки системы.

Критерии оптимизации заданы алгоритмически, согласно модели надежности программного обеспечения, основные обозначения которой следующие [2]:

1) M – число архитектурных 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) PU_{ij} – вероятность использования компонента i на уровне j;
5) PF_{ij} – вероятность появления сбоя в компоненте i на уровне j;
6) PL^{ij}_{nm} – условная вероятность появления сбоя в компоненте m на уровне n при появлении сбоя в компоненте i на уровне j, $i \in \{1,..,N_j\}$, $j \in \{1,..,M\}$, $n \in \{1,..,N_m\}$, $m \in \{1,..,M\}$;
7) TA_{ij} – относительное время доступа к компоненту i на уровне j;
8) TC_{ij} – относительное время анализа сбоя в компоненте i на уровне j;
9) TE_{ij} – относительное время устранения сбоя компонента i на уровне j;
10) TU_{ij} – относительное время использования компонента i на уровне j;
11) Z_{ij} – множество версий компонента i, на уровне j, k=1,..,K;
12) T_{ij} – трудоемкость разработки компонента i на уровне j;
13) T^k_{ij} – трудоемкость разработки версии k компонента i на уровне j, $k \in Z_{ij}$ в чел-часах;
14) NVX_{ij} – трудоемкость разработки среды исполнения версий (приемочного теста для *RB* или алгоритма голосования для *NVP*);
15) B_{ij} – дихотомическая переменная, принимающая значение 1 (тогда NVP_{ij}=0, RB_{ij}=0), если в программном компоненте не используется программная избыточность, иначе равна 0.
16) 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, иначе равна 0.
17) 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, иначе равна 0.
18) TR – среднее время простоя системы, определяемое как время, в

течение которого система не может выполнять свои функции;

19) *MTTF* – среднее время появления сбоя, определяемое как время, в течение которого сбоев в системе не происходит;

20) S – коэффициент готовности.

21) T_s – общая трудоемкость системы;

Компоненты архитектуры программного обеспечения могут быть реализованы с использованием программной избыточности. Если вводится программная избыточность, то ее можно реализовать методом мультиверсионного программирования (*NVP – N-version programming*) или блока восстановления (*RB – recovery block*) [3].

Программный компонент или каждая его версия, в случае применения программной избыточности, могут быть доведены до определенного уровня надежности (вероятности сбоя). С помощью статистических моделей надежности или экспертных оценок, могут быть определены варианты возможного уровня надежности компонента (или его версии) и соответствующих трудозатрат на достижение этого уровня надежности.

Трудоемкость разработки системы рассчитывается следующим образом:

$$T_S = \sum_{j=1}^{j=M} \sum_{i=1}^{i=N_j} \left(B_{ij} T_{ij} + \left(NVP_{ij} + RB_{ij} \right) \left(NVX_{ij} + \sum_{k \in Z_{ij}}^{K} T_{ij}^{k} \right) \right)$$

Среднее время сбоя равно[2]:

$$MTTF = \sum_{j=1}^{j=M} \sum_{i=1}^{i=N_j} \{ PU_{ij} \times (1 - PF_{ij}) \times [TU_{ij} +$$

$$\sum_{(m=1)\&(m \neq j)}^{m=M} \sum_{n=1}^{n=N_m} \left((1 - PL_{nm}^{ij}) \times \left(TU_{nm} + \sum_{l \in D_{nm}} \left((1 - PL_{lm}^{nm}) \times TU_{lm} \right) \right) \right) +$$

$$\sum_{k \in D_{ij}} \left((1 - PL_{kj}^{ij}) \times \left(TU_{kj} + \sum_{(m=1)\&(m \neq j)}^{m=M} \sum_{n=1}^{n=N_m} \left((1 - PL_{nm}^{kj}) \times \left(TU_{nm} + \sum_{l \in D_{nm}} \left((1 - PL_{lm}^{nm}) \times TU_{lm} \right) \right) \right) \right) \right)]\}$$

Среднее время простоя системы равно[2]:

$$TR = \sum_{j=1}^{j=M} \sum_{i=1}^{i=Mj} \{ PU_{ij} \times PF_{ij} \times [(TA_{ij} + TC_{ij} + TE_{ij}) +$$

$$\sum_{(m=1)\&(m \neq j)}^{m=M} \sum_{n=1}^{n=N_m} \left(PL_{nm}^{ij} \times \left((TA_{nm} + TC_{nm} + TE_{nm}) + \sum_{l \in D_{nm}} \left(PL_{lm}^{nm} \times (TA_{lm} + TC_{lm} + TE_{lm}) \right) \right) \right)$$

$$\sum_{k \in Dij} [PL_{kj}^{ij} \times [(TA_{kj} + TC_{kj} + TE_{kj}) +$$

$$\sum_{(m=1)\&(m \neq j)}^{m=M} \sum_{n=1}^{n=N_m} \left(PL_{nm}^{kj} \left((TA_{nm} + TC_{nm} + TE_{nm}) + \sum_{l \in D_{nm}} \left(PL_{lm}^{nm} \times (TA_{lm} + TC_{lm} + TE_{lm}) \right) \right) \right)]]]\}$$

Коэффициент готовности равен:

$$S = \frac{MTTF}{MTTF + TR}$$

Анализ мощности пространства оптимизации позволяет сделать вывод о том, что для решения задачи необходимо применение генетического алгоритма (ГА), т.к. задача определения оптимальной архитектуры ПО является *NP*-полной и перебор всех вариантов решения не представляется возможным [4].

Построение структуры генотипа для поставленной задачи оптимизации. Для части компонентов архитектуры, участвующих в ГА и для которых возможно введение программной избыточности, могут быть изменены следующие характеристики:

1. Метод реализации программной избыточности: мультиверсионное программирование (NVP_{ij}=1, RB_{ij}=0) или блок восстановления (NVP_{ij}=0, RB_{ij}=1). Если выбран метод *NVP*, то устанавливается значение 0, если *RB*, то 1.
2. Var_{v1} – вариант вероятности сбоя компонента и соответствующей трудоемкости для достижения этой вероятности сбоя. Возможные варианты задаются аналитиком (1<=Var_{v1}<=*Кол-во вариантов для данного компонента*).
3. Var_{v2}..Var_{v10} – номер варианта вероятности сбоя для каждой версии компонента, аналогично пункту 2 (0<=$Var_{v2..10}$<=*Кол-во вариантов для данного компонента*, 0 – версии нет). Предельное количество версий программного компонента, если будет применена избыточность, заранее задается проектировщиком.

Если аллели Var_{v2}..Var_{v10} принимают значение 0, то считается, что программная избыточность не вводится в данном компоненте (B_{ij}=1).

Для другой части компонентов, участвующих в ГА, недопустимо введение программной избыточности. Поэтому для них возможно только изменение варианта *Var* вероятности сбоя компонента и соответствующей трудоемкости для достижения этой вероятности сбоя (1<=Var_{v1}<=*Кол-во вариантов для данного компонента*).

Таким образом, фенотип особи формируется из компонентов, участвующих в ГА, где каждая приведенная характеристика программного компонента представляет собой ген. В таблице 1 представлен общий вид фенотипа и примером аллелей и локусов.

Таблица 1. Генотип и фенотип особи

Группа компонентов, с возможностью программной избыточности									Группа компонентов, без возможности программной избыточности		
Компонент 1				..	Компонент N				Компонент 1	..	Компонент N
NVP/RB	Var_{v1}	..	Var_{v4}	..	NVP/RB	Var_{v1}	..	Var_{v5}	Var	..	Var
1	3	..	0	..	0	1	..	0	2	..	3

Этапы генетического алгоритма. Каждая особь с рассчитанными критериями записывается в массив. До расчета значений критериев для следующей особи происходит поиск уже рассчитанной ранее особи с идентичными характеристиками. Это позволяет сэкономить время, т.к. процедура расчета критериев значительно дольше поиска идентичной особи в массиве решений.

Скрещивание выбранных особей происходит с заранее заданной вероятностью. Если родители не скрещиваются, то происходит их клонирование.

В природе существует огромное количество признаков и свойств живых организмов, которые определяются двумя и более парами генов, и наоборот, один ген часто контролирует многие признаки. Кроме того, действие гена может быть изменено соседством других генов и условиями внешней среды. Еще в одной из работ 1928 г. Ю.А. Филипченко, было доказано, что наряду с «основным» геном, определяющим признак, существует ряд генов-модификаторов этого признака. Таким образом, фенотип, как правило, представляет собой результат сложного взаимодействия генов [5, с.50].

Надежность и трудозатраты для компонентов, в которых возможна программная избыточность, определяются взаимодействующими генами версий и метода избыточности. Разрывы хромосомы могут происходить в месте жирной черты (таблица 1). Причем вероятность выбора точки разрыва хромосомы между генами одного программного компонента (связанными генами) заранее задана. Равномерное скрещивание возможно с разрывом во всех точках или только несвязанных генов.

Мутация особей популяции происходит с заданной вероятностью. Кроме вероятности применения мутации к каждой особи, используется вероятность применения

мутации к каждому ее гену, величину которой обычно задают от 1 до 10%. При мутации бинарных генов метода избыточности происходит инвертирование значения.

Аллели генов вариантов надежности/трудоемкости характеризуются шкалой порядка, таким образом, что каждый следующий вариант, лучше по надежности, но хуже по трудозатратам. Т.е. заранее заданные варианты для конкретного программного компонента оптимальны по Парето. Мутация таких генов может происходить двумя способами:

1. Выбор любого варианта надежности/трудоемкости равновероятен.
2. Выбор любого варианта надежности/трудоемкости происходит вероятностью распределенной по закону нормального распределения вероятностей для дискретной случайной величины [9].

Генетический алгоритм основан на методе с независимой селекцией Шаффера при многокритериальной оптимизации *VEGA* (*Vector Evaluated Genetic Algorithm*) [6, с. 33]. *Селекция* происходит с вероятностью, пропорциональной значению критерия [7, с.36].

Входные параметры для ГА следующие:

- размер популяции (*N*);
- вероятность скрещивания (*prob_cross*);
- вид скрещивания (1,2,3-точечное, равномерное);
- вероятность разрыва связанных генов (*prob_cross_inter*);
- вероятность мутации особи (*prob_mutate*) и гена (*prob_mutate_gen*);
- закон распределения вероятности выбора альтернативы при мутации гена (равновероятный, нормальный);
- критерии останова (максимальное время работы, количество популяций).

Описание алгоритма следующее:

1. Генерация родительской популяции *P* размером *N* случайных особей.
2. Расчет критериев для всех особей популяции *P* (рисунок 1).
3. Пропорциональная селекция *N*/2 особей из *P* по критерию *S* в промежуточную популяцию *P'*.
4. Пропорциональная селекция *N*/2 особей по критерию T_s в промежуточную популяцию *P'*.
5. Скрещивание (с вероятностью *prob_cross*) *N*/2 случайно выбранных пар особей из промежуточной популяции *P'*. Добавление *N* полученных потомков в основную популяцию *P*.
6. Проведение оператора мутации с вероятностью *prob_mutate* на каждой особи основной популяции *P*.
7. Расчет критериев для всех особей популяции *P*.
8. Если не сработал хотя бы один критерий останова, то переход на шаг 3.

По полученным в результате работы ГА решениям формируется множество решений, недоминируемых по Парето. С точки зрения математики решения множества Парето не могут быть предпочтены друг другу, поэтому после формирования множества Парето задача может считаться математически решенной [6, с.58].

Экспериментальная часть. Алгоритм был протестирован на 20 тестовых задачах, с заранее просчитанными вариантами решения и отобранными оптимальными решениями. На каждой задаче был был запущен алгоритм в 56 различных конфигурациях. На каждый ГА было дано 50 поколений по 50 популяций, вероятность скрещивания – 0.9. Алгоритмами было просчитано не более 7% поискового пространства. Для сравнения алгоритмов использовались показатели рассеяния решений в критериальном пространстве и процент решений, принадлежащих множеству Парето решаемой задачи.

Анализ результатов тестирования позволил сделать следующие выводы:

1. Закон распределения вероятности выбора аллели, при мутации, влияет на эффективность ГА не значительно.
2. На эффективность ГА больше влияет вероятность мутации. При вероятности мутации особи 0.15 и гена 0.1 эффективность алгоритмов более высокая, чем при вероятности мутации особи 0.05 и гена 0.01.
3. Поиск решений наиболее эффективен при использовании модифицированного оператора 3-точечного кроссинговера.
4. Наиболее эффективно отработали алгоритмы с вероятностью разрыва связанных генов 0,5 для выбранной задачи.
5. Применение модифицированного равномерного кроссинговера с разрывом только несвязанных генов менее эффективно.
6. В среднем 14.63% найденных алгоритмом решений входят в множество Парето. Остальные решения достаточно близки к множеству Парето.

Заключение

Разработан и протестирован ГА для многокритериальной безусловной оптимизации программной архитектуры, основанный на методе *VEGA*. Алгоритм находит оптимальные решения и близкие к оптимальным за приемлемое время, поэтому он может быть использован при решении задач проектирования реальных программных систем.

СПИСОК ЛИТЕРАТУРЫ:

1. Липаев В.В. Программно-технологическая безопасность информационных систем. – Jet Info, №6/7 (37/38), 1997.
2. Русаков М.А. Многоэтапный анализ архитектурной надежности в сложных информационно-управляющих системах: Дис. канд. техн. наук: Красноярск, 2005 – 168 с.
3. Новой А.В. Система анализа архитектурной надежности программного обеспечения.: Дис. канд. техн. наук: Красноярск, 2011 – 131 с.
4. Шеенок Д.А, Жуков В.Г., Терсков В.А. Повышение надежности программного обеспечения сложных систем. // Вестник СибГАУ. Вып. 5(45). – Красноярск, 2012. – С. 28-33.
5. Инге-Вечтомов С.Г. Генетика с основами селекции: Учеб. для биолог. спец. ун-тов. – М.: Высш. шк., 1989. – 591 с.
6. Сергиенко Р.Б. Автоматизированное формирование нечетких классификаторов самонастраивающимися коэволюционными алгоритмами: Дис. канд. техн. наук: Красноярск, 2010 – 192 с.
7. Гуменникова А.В. Адаптивные поисковые алгоритмы для решения сложных задач многокритериальной оптимизации: Дис. канд. техн. наук: Красноярск, 2006 – 129 с.

REFERENCES:

1. Lipaev V.V. Programmno-tehnologicheskaya bezopasnost' informaczionnyh sistem. – Jet Info, №6/7 (37/38), 1997.
2. Rusakov M.A. Mnogoetapnyj analiz arhitekturnoj nadezhnosti v slozhnyh informaczionno-upravlyayushih sistemah: Dis. kand. tehn. nauk: Krasnoyarsk, 2005 – 168 p.
3. Novoj A.V. Sistema analiza arhitekturnoj nadezhnosti programmnogo obespecheniya.: Dis. kand. tehn. nauk: Krasnoyarsk, 2011 – 131 p.
4. Sheenok D.A. Zhukov V.G., Terskov V.A. Povyshenie nadezhnosti programmnogo obespecheniya slozhnyh sistem. // Vestnik SibGAU. Vyp. 5(45). – Krasnoyarsk, 2012. – P. 28-33.

5. Inge-Vechtomov S.G. Genetika s osnovami selekczii: Ucheb. dlya biolog. specz. un-tov. – M.: Vyssh. shk., 1989. – 591 p.

6. Sergienko, R.B., Avtomatizirovannoe formirovanie nechetkih klassifikatorov samonastraivayushhimisya koevolyuczionnymi algoritmami: Dis. kand. tehn. nauk: Krasnoyarsk, 2010 – 192 p.

7. Gumennikova A.V. Adaptivnye poiskovye algoritmy dlya resheniya slozhnyh zadach mnogokriterial'noj optimizaczii: Dis. kand. tehn. nauk: Krasnoyarsk, 2006 – 129 p.