МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
им. М.В. ЛОМОНОСОВА

Факультет вычислительной математики и кибернетики

ПРОГРАММНЫЕ СИСТЕМЫ И ИНСТРУМЕНТЫ

Тематический сборник

№ 7

Под общей редакцией
чл.-корр. РАН Л.Н. Королева

МАКС ПРЕСС

МОСКВА – 2006

УДК 519.6+517.958
ББК 22.19
П75

Печатается по решению Редакционно-издательского совета факультета вычислительной математики и кибернетики МГУ им. М.В. Ломоносова

П75 **Программные системы и инструменты:** Тематический сборник факультета ВМиК МГУ им. М. В. Ломоносова: № 7/ Под общ. ред. Л.Н. Королева. – М: Издательский отдел факультета ВМиК МГУ (лицензия ИД №05899 от 24.09.2001г.); МАКС Пресс, 2006. – 184 с.
ISBN 5-89407-287-5
ISBN-13: 978-5-317-01842-9
ISBN-10: 5-317-01842-0

В данный сборник включены научные работы и сообщения, связанные с общими вопросами программирования и информатики, касающиеся, в том числе проблем образования, исследований и разработок, связанных с интеллектуальным анализом данных, имитационным моделированием систем реального времени, сетевой обработкой, а также с описанием некоторых инструментальных систем, которые могут оказаться полезными.

В этих публикациях нашли отражение исследования и разработки в области создания программных систем, выполненные учеными, аспирантами и студентами факультета. Большая часть результатов доложена на Ломоносовских Чтениях 2006 года.

Статьи сборника будут интересны студентам, аспирантам и специалистам в области разработки прикладных программных систем с использованием новых информационных технологий.

УДК 519.6+517.958
ББК 22.19

Издательский отдел
Факультета вычислительной математики и кибернетики
МГУ им. М.В. Ломоносова
Лицензия ИД N 05899 от 24.09.01 г.

119992, ГСП-2, Москва, Ленинские горы, МГУ им. М.В. Ломоносова,
2-й учебный корпус

Напечатано с готового оригинал-макета
в издательстве ООО "МАКС Пресс"
Лицензия ИД N 00510 от 01.12.99 г.
Подписано к печати 13.12.2006 г.
Формат 60x90 1/16. Усл.печ.л. 11,5. Тираж 100 экз. Заказ 893.

119992, ГСП-2, Москва, Ленинские горы, МГУ им. М.В. Ломоносова, 2-й учебный корпус, 627 к
Тел. 939-3890, 939-3891. Тел./Факс 939-3891.

ISBN 5-89407-287-5
ISBN-13: 978-5-317-01842-9
ISBN-10: 5-317-01842-0

СОДЕРЖАНИЕ

В предлагаемом читателю тематическом сборнике публикуются статьи, посвященные в основном описаниям инструментальных программных систем, разработанных авторами публикаций, относящихся к методам интеллектуального анализа данных, имитационному моделированию, проблемам сетевой обработки, некоторым вопросам теоретического программирования.

В нем также публикуются статьи и заметки общего характера, касающиеся разделов информатики, тесно связанных с программированием.

Редколлегия

Раздел I
Общие вопросы программирования и информатики

Махнычев В. С., Ильичев А. Б.

Распараллеливание задачи о составлении вузовского расписания с использованием распределенного обхода дерева перебора

Введение

Задача о составлении вузовского расписания – это задача о составлении расписания комбинаторного типа с дополнительными условиями оптимальности решения и ограничениями сложной формы. Особенностью задачи является большая многовариантность, не позволяющая перебрать все возможные комбинации для получения точного решения.

Для решения подобных задач применяются комбинации переборных и эвристических алгоритмов. При этом количество рассмотренных в переборе вариантов и – как следствие – точность полученного решения напрямую зависят от мощности используемой вычислительной системы. Таким образом, для получения решения, наиболее близкого к оптимальному, становится актуальным эффективное распараллеливание алгоритма решения задачи.

Для распараллеливания была разработана специальная комбинация переборного и эвристического алгоритма, позволившая применить селективную схему параллельного обхода дерева перебора, ранее проверенную на алгоритме игры в шахматы и подробно описанную в [1]. Особое внимание уделялось оптимизации переборного алгоритма, так как именно он подлежит распараллеливанию и определяет эффективность параллельной схемы. Для испытания реализованной системы использовалось несколько различных компьютеров, объединенных в локальную сеть.

Постановка задачи

В общем случае задачу о составлении вузовского расписания можно сформулировать следующим образом.

Дано множество групп G, множество преподавателей C,

множество академических часов (например, за неделю) T и множество аудиторий P. Определим занятие a как четверку (g,c,t,p), $g \in G$, $c \in C$, $t \in T$, $p \in P$, означающую, что преподаватель c проводит занятие в группе g в академический час t в аудитории p. Требуется составить график занятий A (множество занятий) таким образом, чтобы количество часов, проведенных в каждой группе определенным преподавателем, соответствовало заданному для этой группы (исходя из учебного плана). При этом один преподаватель не может вести занятия в нескольких группах одновременно, то есть

$$\forall a_0 = (g_0, c_0, t_0, p_0) \in A \Rightarrow \bigcup_{\substack{a=(g,c,t,p)\in A,\\ t=t_0, p=p_0, g\neq g_0}} a = \phi .$$

Кроме этого, специфика высшего образования вносит дополнительные элементы расписания – такие, как лекции, читаемые одним преподавателем для нескольких групп одновременно. Некоторые виды занятий требуют также аудиторий определенного типа (например, поточные лекции могут проходить только в больших аудиториях, а семинары по программированию должны включать в себя занятия в машинном зале). На аудитории также накладывается ограничение: ни одна аудитория не может быть одновременно занята более чем одним преподавателем.

Также задаются дополнительные условия оптимальности решения, как-то: нежелательность «окон» (пустых пар) между занятиями групп, нежелательность занятий в субботу, индивидуальные пожелания преподавателей и так далее.

Оценка оптимальности решения

Для оценки эффективности каждого полученного решения использовалась система неотрицательных штрафов: для каждого занятия и каждого места в расписании задан штраф за назначение данного занятия на данное место. Легко видеть, что при этом не играет существенной роли, в какой аудитории будет проходить занятие, значение имеет только тип необходимой аудитории. Таблица штрафов строится эвристическим алгоритмом. Отличительной особенностью этой системы является возможность вычислить точный штраф для неполного решения, когда известна лишь часть расписания – например, расписание только для части групп; и, что немаловажно, вычисленный штраф не зависит от изменения остальной части решения (расписания остальных групп), то есть входит в суммарный (полный) штраф в качестве постоянного слагаемого.

Кроме того, для каждой отдельно взятой группы можно получить некоторое число m_i – минимальный штраф для этой группы

при условии, что все используемые группой ресурсы (преподаватели и аудитории) не заняты другими группами. В рамках этого предположения задача составления оптимального расписания для одной группы есть не что иное как задача о назначениях. Для ее решения используется венгерский метод.

Таким образом, вычислив суммарный штраф для сформированной части расписания и прибавив к ней минимальные штрафы m_i для тех групп, расписание которых еще не было составлено, мы получим нижнюю границу полного штрафа (очевидно, что при составлении общего расписания штраф каждой конкретной группы может только увеличиться по сравнению с минимальным штрафом). А так как алгоритм старается составить оставшуюся часть расписания «идеально», то штраф в оставшейся части – в случае успешного решения задачи – близок к сумме минимальных штрафов m_i, и полный штраф близок к полученной нижней границе. Эта особенность – близость предполагаемого штрафа оптимального решения к оценке, полученной быстрыми приближенными вычислениями, – позволила применить метод ветвей и границ в качестве алгоритма перебора.

Переборный блок

При такой системе оценок удобно было выбрать в качестве одного шага ветвления назначение одного занятия. При этом, в чистом виде метод ветвей и границ не применим из-за огромного количества ветвлений (при шести учебных днях и пяти возможных занятиях в день число вариантов расписания для одной группы приближается к факториалу 30, а для нескольких десятков групп – факультета целиком – количество различных вариантов необозримо даже при самых эффективных отсечениях ветвей). Поэтому число вариантов на каждом уровне перебора ограничивалось некоторой функцией, которая зависит от общего числа вариантом в данной вершине, от глубины этой вершины в дереве перебора и от числового параметра, который медленно увеличивался по мере увеличения времени перебора. При таком ограничении числа вариантов имеет смысл рассматривать варианты в порядке уменьшения их перспективности. Поэтому в каждой вершине дерева перебора варианты сортируются в соответствии с их штрафом.

Для удобства применения селективной схемы в переборном алгоритме используется метод итеративного углубления, заключающийся в том, что на первой итерации перебор затрагивает один уровень дерева перебора, на второй итерации – два уровня, и так далее. Это было реализовано путем введения дополнительного параметра в функцию, ограничивающую число вариантов на каждом уровне перебора. Этот параметр определяет глубину в дереве перебора,

начиная с которой в каждой вершине рассматривается ровно один вариант. При рассмотрении на каждом уровне ровно одного варианта, переборный алгоритм фактически вырождается в жадный алгоритм. При этом часть дерева выше этого уровня соответствует собственно перебору, а ветвь, расположенная ниже, соответствует решению, которое строит жадный алгоритм.

Эвристический алгоритм

Для построения таблицы штрафов использовался «жадный» алгоритм с некоторыми эвристиками. Общая схема работы алгоритма такова:

1) все занятия сортируются согласно некоторому порядку;
2) для каждого занятия изначально устанавливается высокий штраф для всех мест в расписании;
3) последовательно просматриваются все занятия и для каждого занятия, с учетом оценки на количество свободных ресурсов и общую занятость группы, определяется день, в который это занятие желательно назначить. После чего штрафы для этого дня понижаются;
4) пожелания преподавателей вносятся в таблицу в явном виде.

В алгоритме учитывается также нежелательность «окон» в расписании отдельных групп. Чтобы избежать «окон», предусмотрены следующие меры. Во-первых, порядок рассмотрения занятий таков, что сначала рассматриваются все лекционные занятия. Это существенно, поскольку лекция проходит одновременно у многих групп, и ее назначение затруднительно, когда часть расписания уже построена. Кроме того, при выборе «желательного» дня для каждого занятия наименьший штраф устанавливается для 3-й пары, чуть больше – для 2-й и 4-й, и наибольший – для 1-й и 5-й. Это, с одной стороны, делает окна в расписании невыгодными с точки зрения переборного алгоритма. С другой стороны, поскольку занятия можно назначать как до уже назначенных, так и после, это увеличивает число вариантов для назначения занятий «без окон». В противоположность этому: если мы назначим занятие 1-й парой, то у нас будет только один вариант для назначения следующего занятия в этот день так, чтобы не получилось «окон».

Стоит, кроме того, заметить, что тот же самый порядок занятий используется в переборе: занятия назначаются именно в том же порядке. Это делает возможным не осуществлять перебор из корня дерева, а предварительно назначить несколько занятий эвристически. Это фактически достигается тем, что функция, ограничивающая число вариантов на каждом уровне перебора, задает очень маленькие

ограничения на верхних уровнях. Фактически самые верхние уровни перебора, на которых рассматривается только один вариант, соответствуют эвристическому назначению первых занятий, причем в нашем случае это будут лекции.

Применение селективной схемы

Идея селективной схемы параллельного обхода дерева перебора [1] состоит в рассмотрении лишь нескольких, наиболее перспективных ветвей дерева перебора и наличии в вершинах дерева, соответствующих узлам распределенной системы, проверяющих узлов, с помощью которых производится поиск новых перспективных вариантов. Метод итеративного углубления, применяемый для решения задачи о составлении расписания, хорошо согласуется с селективной схемой, так как позволяет получать уточняющиеся оценки для выбранных ветвей с течением времени.

Применение селективной схемы в чистом виде, начиная с корня дерева перебора, оказалось нецелесообразным. Причина этого в том, что часть расписания может быть легко составлена эвристическим алгоритмом без какого-либо перебора, и для этой части расписания существует еще достаточное количество вариантов составления расписания, при каждом из которых может быть получено оптимальное решение [2]. Таким образом, более целесообразным оказалось зафиксировать положение некоторого (подбираемого эмпирически) числа занятий в расписании и начать применять селективную схему с некоторой внутренней вершины дерева перебора.

В выбранной вершине применяется селективная схема перебора: для выбранной вершины на одном из узлов распределенной системы запускается переборный блок, по промежуточным результатам его работы определяются несколько наиболее перспективных ветвей, и для этих ветвей в свою очередь запускаются переборные блоки на других узлах распределенной системы. Система входит в режим перебора вариантов, при этом оператор системы уведомляется о текущем наилучшем решении. Решение о прекращении перебора принимается оператором системы.

Для ускорения работы алгоритма, все узлы системы информируются о текущем минимальном достигнутом значении полного штрафа. Это позволяет увеличить количество отсечений при переборе.

Результаты испытаний и направления дальнейших исследований

Описанная схема была реализована и испытана на распределенной системе из семи счетных узлов. При таком выборе, для корневой вершины перебора рассматривались две наиболее перспективные ветви; для каждой из этих двух ветвей рассматривалось еще по две наиболее перспективные ветви. Таким образом, селективная схема перебора позволила увеличить глубину просмотра алгоритма на два уровня, что заметно улучшило результаты на различных тестовых наборах, требовавших переборных действий для получения оптимального расписания: по сравнению с однопроцессорной версией того же алгоритма ускорение на тестах составило от 2 до 3,3 раз.

В текущей версии алгоритм не адаптирован к специфике составления расписания для нескольких потоков и курсов одновременно. Специфика состоит в том, что расписания для различных потоков и тем более курсов имеют намного меньше пересечений по общим ресурсам, чем расписания групп одного потока. С учетом этого, в дальнейшем предполагается адаптировать алгоритм, изменив переборный блок таким образом, чтобы минимизировать количество ненужных перестановок (например, при формировании расписания для каждого относительно независимого подмножества групп можно применять ту же стратегию определения точки переборного ветвления, которая применялась в реализованной схеме для всего множества групп).

Литература

1. Махнычев В.С. Распараллеливание сложных интеллектуальных задач, основанных на обходе дерева перебора // Программные системы и инструменты. Тематический сборник №5. – М.: Факультет ВМиК МГУ, 2004.
2. Панкратьев Е.В., Чеповский А.М., Черепанов Е.А., Чернышёв С.В. Алгоритмы и методы решения задач составления расписаний и других экстремальных задач на графах больших размерностей // Фундаментальная и прикладная математика 2003, т.9, выпуск 1, с.235-251.

Истомин Т.Е., Пыптев С.А.

Технология конфигурирования программных систем

В данной статье описывается технология конфигурирования программных систем перед компиляцией исходных кодов. Технология создавалась при разработке операционной системы для узлов беспроводных сетей на основе микроконтроллеров, что определило ее основные характеристики.

Одним из основных требований к программным системам для миниатюрных устройств является компактность программного кода. С другой стороны, нередко требуется определенная гибкость исходного кода для поддержки нескольких аппаратных платформ, что приводит к включению в него множества параметров, а так же к использованию платформенно-зависимых или специально оптимизированных вариантов модулей системы.

Исходный код параметризуется, например, используемыми типами данных, различными аппаратными константами (номера прерываний, адреса устройств ввода-вывода), количественными величинами (размеры таблиц, буферов, стека вызовов).

Специально оптимизированными вариантами модулей являются, например, статический менеджер ресурсов против динамического, ограниченная реализация стека протоколов против полной, и т.п. При настройке программной системы на конкретное применение в окончательный код должен быть включен один из вариантов реализации.

Технология конфигурирования призвана разрешить противоречие между требованиями гибкости и компактности, позволяя принимать как можно больше решений статически, до компиляции.

Возможности описываемой в данной статье технологии показаны на примерах из опыта разработки реальной программной системы.

Основные характеристики

Предлагаемый подход основывается на языке описания правил и возможностей конфигурирования исходного кода, и инструментального средства для построения корректного и компактного кода на основе исходного, с учетом требований пользователя.

Приведем основной сценарий применения данной технологии конфигурирования.

1. Программный код разрабатываемой системы дополняется специальными директивами, которые могут помещаться как в файлах с исходным кодом, в специальных комментариях, так и в

отдельных файлах конфигурации.

2. Специальное программное средство (конфигуратор) сканирует файлы проекта, считывая конфигурационные директивы, строит представление о системе в плане возможностей конфигурирования, и предлагает пользователю возможности по изменению этого представления. Под пользователем понимается человек, настраивающий систему на конкретные условия применения.
3. Пользователь осуществляет необходимые действия по настройке системы с помощью конфигуратора. Конфигуратор контролирует корректность этих действий.
4. По запросу конфигуратор генерирует дополнительные файлы исходного кода, необходимые для сборки системы и отражающие выбор пользователя.
5. Далее осуществляется компиляция системы стандартными средствами.

К директивам конфигурации относятся объявление объектов конфигурации, и установление связей между ними. Объектами конфигурации являются конфигурационные переменные, группы и массивы таких переменных. Примерами связи являются установка значения одной конфигурационной переменной в зависимости от значения другой, или ограничение диапазона допустимых значений какой-либо конфигурационной переменной.

С объектами конфигурации может быть связано несколько атрибутов, например тип данных переменной, диапазон допустимых значений, тип генерации кода для данного объекта, значение по умолчанию, и другие.

Конечной целью конфигурирования является задание значений всем конфигурационным переменным с соблюдением всех зависимостей.

Определение констант и типов данных

Простейшей задачей конфигурации является определение констант и типов данных. Покажем, как это делается в рамках нашего подхода.

В каком-либо заголовочном файле определяется конфигурационные переменные в области комментария, игнорируемой компилятором.

```
/*
<var name="ResourceStaticEnabled" type="bool"
codegen="define" default="True"> Defines whether
static resource registration support is included
</var>

<var name="ResourceId" type="name"
```

```
codegen="typedef" default="uint16_t">
  Type of resource ID
</var>
*/
```

В примере выше определены две конфигурационные переменные. Первая из них имеет булевский тип данных, значение по умолчанию “True”, тип генерации кода “define”. Внутри XML-элемента находится текст описания переменной, который отображается в пользовательском интерфейсе конфигуратора. Вторая переменная задает тип данных, что отражает тип генерации кода “typedef”.

Предположим, что пользователь при конфигурировании не изменил значения этих переменных, оставив в силе значения по умолчанию. В таком случае для них сгенерируется следующий код:

```
#define ResourceStaticEnabled True
typedef uint16_t ResourceId;
```

Теперь рассмотрим более сложную задачу, с зависимостями между конфигурационными объектами. Пусть приведенные выше переменные заданы в интерфейсном файле некоторого модуля, для которого существует два варианта реализации. Пусть в исходном коде одного из них встречается директива:

```
<assign
name="/resource/interface/resource.h/ResourceId"
value="uint8_t"/>
```

Тогда, в случае включения этого варианта реализации модуля (о том, как это делается — ниже), переменной присвоится значение, подходящее именно для этого варианта. Таким образом, получаем связь между выбором модуля и переменной конфигурации. В данном случае пользователь, выбрав модуль, уже не вправе изменять жестко установленное модулем значение.

Возможна ситуация, когда модуль не задает значение жестко, а лишь сужает возможный диапазон значений переменной.

```
<assign
name="/resource/interface/resource.h/ResourceId"
range="uint8_t,uint16_t"/>
```

Для строковых переменных используются диапазоны-перечисления возможных значений, указываемых через запятую. Диапазоны числовых переменных задаются набором интервалов.

Конфигуратор следит за тем, чтобы устанавливаемые разными модулями значения и диапазоны переменной не конфликтовали.

Конфликты значений или диапазонов свидетельствуют о несовместимости выбранных модулей друг с другом.

В примерах выше видно как адресуются объекты. В адресе участвует путь до файла, где объект определен, и имя объекта, отделенное косой чертой. Путь указывается относительно корня иерархии каталогов проекта. Для адресации множества объектов можно использовать шаблоны обобщения пути и имени, подробнее об этом — ниже.

Выбор модулей системы

Второй основной возможностью данной технологии конфигурирования является включение или исключение частей исходного кода на уровне файлов и каталогов. Файл или каталог может быть включен в процесс конфигурирования и компиляции только если включен его родительский каталог. Таким образом, исключение каталога означает автоматическое исключение всех содержащихся в нем файлов и подкаталогов.

Каждый файл и каталог исходных кодов проекта считается объектом конфигурации наряду с конфигурационными переменными и другими объектами. Файлам и каталогам может быть присвоено одно из значений True или False, говорящее о том, включен ли данный файл или каталог. Как и переменным, значение активности каталога может быть присвоено как пользователем, так и из исходных кодов.

При включении файла активизируются все конфигурационные директивы, содержащиеся в нем, при условии отсутствия конфликтов. После исключения файла, все эффекты действия его директив на внешние конфигурационные объекты снимаются, и код для него не генерируется.

Рассмотрим примеры. Пусть в некотором глобальном для проекта файле конфигурации определены следующие директивы:

```
<assign name="/~/dir.conf" value="true"/>
<assign name="//test" default="false"/>
```

Сначала обратим внимание на адресацию объектов. Здесь используются шаблоны обобщения. Строка “/~/” означает любое количество произвольных имен в пути, “//” — одно произвольное имя. Таким образом, в первой строчке включаются все файлы с именами “dir.conf” независимо от их расположения. Во второй строке — все файлы или каталоги с именем “test”, находящиеся во всех каталогах второго уровня.

Интересен эффект первой строчки. Ее применение, по существу означает, что при включении произвольного каталога проекта автоматически включается содержащийся в нем файл “dir.conf” (если он есть), что позволяет ставить в соответствие каталогу порцию

конфигурационных директив, например, для логического связывания содержащихся в нем файлов и подкаталогов между собой и с внешними конфигурационными объектами.

Поддерживаются и некоторые другие обобщающие шаблоны, например, диапазоны имен.

```
<assign name="//implementation//!{test}"
default="true"/>
```

В этом примере по умолчанию включаются все файлы и каталоги, находящиеся в каталогах второго уровня относительно "implementation", за исключением тестов. Подобные конструкции позволяют описывать логическую структуру исходных кодов проекта.

Группировка булевых переменных

Конфигурационные переменные логического типа и, в частности, переменные, соответствующие файлам и каталогам проекта могут быть объединены в группы. Не более чем одна переменная группы может иметь значение True в каждый момент времени.

Такая возможность нужна, в основном, для организации выбора одного из вариантов реализации модуля, или одного из вариантов конфигурации системы или части системы. Например, задание такой группы позволяет выбрать конфигурацию для одной из поддерживаемых программных платформ:

```
<group>
  /arch/
</group>
```

В примере выше использован обобщающий шаблон, соответствующий всем элементам каталога "/arch".

Группы могут быть именованными, что позволяет наполнять группу распределенно, из разных файлов, помещая переменную в группу непосредственно при объявлении этой переменной.

Настройка интерфейсов модулей

Как уже было сказано выше, интерфейсы модулей (в Си — заголовочные файлы) могут быть настроены на выбранный вариант реализации, например, заданием используемых типов данных. Кроме этого, с помощью конфигурационных переменных можно обеспечить дополнительную возможность объявления функции интерфейса, независимо от природы ее реализации. Реализация может быть как функцией времени выполнения, так и функцией времени компиляции, то есть макросом.

```
<var name="EventCl_register(clName, priority)"
type="name" codegen="define">
```

С каждым объектом конфигурации может быть связан дополнительный код, вставляемый конфигуратором до или после объявления соответствующего объекта языка программирования при генерации кода. Дополнительный код может добавляться к объекту из произвольных файлов проекта.

Массивы

Наряду со скалярными данными, конфигуратор поддерживает одномерные массивы. Длина массива может зависеть от значения какой-либо целой конфигурационной переменной.

```
<array name="STEPS" codegen="define" type="int"
       dim="NUMBER_OF_STEPS"
       default="1 2 4 8 65535 131070"/>
```

Для массива, объявленного выше сгенерируется следующий код:

```
#define STEPS 1,2,4,8,65535,131070
```

Этот код может использоваться при создании массива на языке Си:

```
int steps[] = {STEPS};
```

Интерфейс пользователя

Инструмент конфигурирования имеет интерфейсы командной строки и графический, и используется для управления конфигурацией проекта. На рисунке представлено главное окно графического интерфейса пользователя. В левой панели отображено дерево проекта, в правой — перечень конфигурационных переменных выбранного файла и описание выбранной переменной.

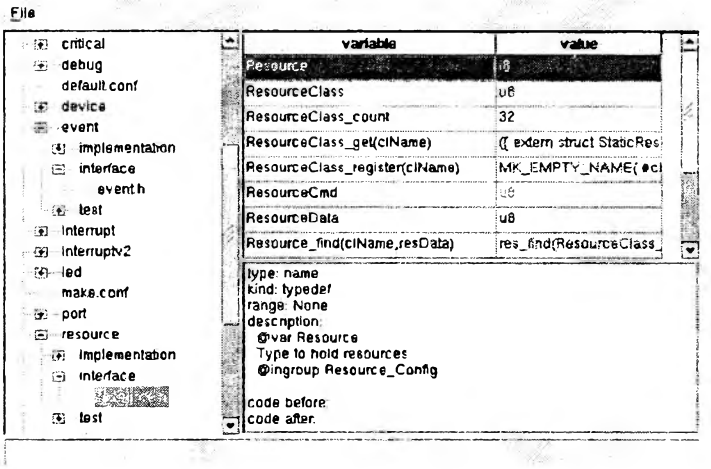


Рисунок 1. Инструмент конфигурирования: графический интерфейс пользователя.

Заключение

Технология, описанная в данной работе, является простой и в то же время довольно общей и гибкой по сравнению с подходами, импользуемыми, например, для конфигурации ядра Linux, систем eCos [1] и TinyOS [2]. Она разрабатывалась не как расширение какого-то конкретного языка программирования, а как дополнение, не зависящее от языка. Технология была опробована при создании операционной системы для узлов сенсорной сети, но может быть использована и в других проектах, предъявляющих особые требования к конфигурированию.

Ссылки

1. The eCos Configuration Tool. http://ecos.sourceware.org/docs-2.0/user-guide/the-ecos-configuration-tool.html
2. TinyOS Tutorial. http://www.tinyos.net/tinyos-1.x/doc/tutorial/

Смирнов А.А.

Методы автоматической расстановки консистентных контрольных точек [1]

Данная статья посвящена вопросам анализа параллельных программ с целью расстановки консистентных контрольных точек. Рассматриваются основные подходы к реализации систем КТ, приводится описание одной из реализаций таких систем, библиотеки libcheckpoint. Описываются два метода автоматического поиска мест создания консистентных контрольных точек: статический анализ трассы параллельной программы и динамический алгоритм поиска мест создания КТ во время работы параллельной программы, приводится сравнение данных методов.

Введение

Контрольная точка – информация о состоянии задачи, которая позволяет продолжить её выполнение с заданного момента времени. Механизм контрольных точек применяется в параллельных вычислительных системах для обеспечения отказоустойчивости (контрольные точки создаются через заданные интервалы времени, в случае сбоя в вычислительной системе происходит откат к последней сохраненной контрольной точке) и для повышения гибкости работы планировщика задач (при помощи механизма контрольных точек планировщик может временно снять с выполнения длительную вычислительную задачу, освободив ресурсы для более коротких по времени выполнения задач). Механизм контрольных точек может применяться и для последовательных программ, но для таких программ в случае отсутствия взаимодействия через коммуникационную среду создание контрольных точек является достаточно простой задачей.

В данной статье в качестве параллельных систем будут рассматриваться только системы обмена сообщениями, использующие

---

[1] Работа выполнена при поддержке Российского фонда фундаментальных исследований, проект № 05-01-00719-а

интерфейс MPI [5], а под параллельной вычислительной системой мы будем понимать кластерную вычислительную систему (или просто кластер). При применении механизма контрольных точек для параллельных задач в системе обмена сообщениями возникают особые проблемы, не свойственные системам контрольных точек для последовательных задач. Дело в том, что возможны ситуации, при которых даже корректное восстановление состояния всех ветвей (процессов) задачи приведет к некорректному продолжению вычислений, что может быть связано с потерей или дублированием сообщений в коммуникационной среде. Кроме проблемы обеспечения корректности восстановления для параллельных задач актуальной является задача повышения эффективности самого механизма контрольных точек. Под повышением эффективности системы контрольных точек понимается уменьшение издержек по времени на создание контрольной точки и восстановление из неё. Очевидно, что контрольные точки должны быть реализованы таким образом, чтобы их применение не оказывало существенного влияния на время выполнения задачи на параллельном вычислителе. Основная часть времени при создании контрольной точки тратится на запись контрольной точки на внешний носитель (контрольная точке не может быть сохранена в оперативной памяти, т.к. её содержимое не сохранится при сбое в вычислителе). Необходимо отметить, что большие издержки на сохранение контрольной точки связаны с большим объемом самой КТ (контрольная точка может включать содержимое всей памяти параллельной задачи, которое складывается из содержимого памяти всех её ветвей, что может составлять объемы в несколько десятков гигабайт).

Все механизмы контрольных точек по способу реализации можно разделить на два класса: *пользовательские* контрольные точки и *автоматические* контрольные точки. В случае применения первого подхода программист для каждой конкретной задачи реализует механизм контрольных точек «с нуля», полностью решая задачу сохранения и корректного восстановления состояния задачи. Автоматические контрольные точки реализуются на уровне операционной системы или на уровне библиотеки программирования, такая реализация предлагает услуги по созданию контрольных точек для широкого класса задач, требуя от программиста внесения минимальных изменений в исходный код задачи. Реализация пользовательских контрольных точек требует от программиста вычислительной задачи достаточно глубоких знаний в области системного программирования для обеспечения корректного продолжения выполнения задачи при восстановлении из контрольной точки, которое требует как восстановления областей данных, так и продолжения выполнения кода с некоторой точки. Автоматический

(или автоматизированный) подход, напротив, требует от программиста минимальных усилий для добавления функциональности контрольных точек в вычислительную задачу. Он может потребовать внесения небольших изменений в исходный код задачи или не потребовать их вовсе. Однако, в силу того, что такой подход может не учитывать все особенности конкретной вычислительной задачи, он может работать не столь эффективно, как ручной (пользовательский) подход. Целью данной работы является создание именно автоматической системы контрольных точек.

Как уже упоминалось ранее, параллельные задачи в кластерных вычислительных системах взаимодействуют при помощи обмена сообщениями через общую коммуникационную среду, которой в кластерах является сеть. Это означает, что в состояние параллельной задачи (а значит и в контрольную точку этой задачи) входит кроме состояния каждой её ветви как отдельного процесса еще и состояние коммуникационной среды. Обычно состояние коммуникационной среды (сообщения, которые могли в ней находиться в момент создания контрольной точки) не сохраняется, так как его сохранение и восстановление потребует значительных усилий, вместо этого вводятся ограничения на это состояние.

При восстановлении из контрольной точки возможны два типа ошибок – потеря и дублирование сообщений, обе эти ошибки для вычислительных задач приведут к некорректному продолжению вычислений (в других применениях, например, в случае взаимодействующих при помощи сообщений приложений, цена потери и дублирования может быть другой). Поэтому будем называть в дальнейшем контрольную точку вычислительной задачи *консистентной*, если в момент её создания не было обменов сообщениями, то есть все отправленные сообщения были получены, и в каналах связи не было сообщений. Во всех других случаях контрольная точка будет называться *неконсистентной*.

Существует два классических алгоритма обеспечения консистентности: *синхронная* и *асинхронная* фиксация контрольных точки. В первом случае происходит инициализация процедуры создания контрольной точки в одной из ветвей, которая отправляет служебное сообщение всем остальным ветвям: «Готовьтесь к созданию контрольной точки». После получения такого сообщения ветви прекращают отправлять сообщения, за исключением служебных, но продолжают принимать сообщения. Затем, когда все ветви убеждаются в том, что коммуникационные каналы пусты (например, при помощи тестовых сообщений), они сообщают о своей готовности ветви-инициатору. Как только все ветви готовы к созданию контрольной точки, они одновременно начинают создавать свои контрольные точки.

После того, как создание контрольных точек завершено, выполнение параллельной задачи продолжается.

Асинхронные контрольные точки создаются в ветвях задачи в произвольные моменты времени, причем в каждой ветви независимо. При этом пропадает всякая необходимость в синхронизации процессов, однако получение консистентной контрольной точки из набора контрольных точек ветвей не всегда является тривиальной задачей. При таком подходе требуется хранить несколько (а иногда и все) контрольные точки для каждой ветви. Поиск консистентного состояния на наборах контрольных точек является задачей отдельного алгоритма, но при его использовании возможны ситуации, когда ни одна комбинация контрольных точек отдельных ветвей не образует консистентного состояния (эффект домино).

Рассмотрим еще один подход: *предварительная синхронизация*, гарантирующая консистентность контрольной точки. В этом случае в исходном коде задачи заранее размещаются подсказки для системы контрольных точек, которые говорят о том, в какой именно момент ветви параллельной задачи находятся в таком состоянии, что их контрольные точки будут образовывать консистентное состояние. Выбор места синхронизации основывается на следующих двух **критериях консистентности**:

1. коммуникационные каналы и буферы MPI пусты (все сообщения отправлены, все операции по приёму сообщений завершены, т.е. содержимое сообщения скопировано в буфер программы);
2. ни один процесс не заблокирован в состоянии получения сообщения.

Эти условия, очевидно, гарантируют консистентность контрольной точки, так как ни одно сообщение не будет потеряно или продублировано (первое условие), и все процессы готовы к созданию контрольной точки (второе условие). Такие точки синхронизации обычно располагаются в конце итерации вычислений, после выполнения групповых операций MPI, которые затрагивают все ветви задачи. Данный подход за счет предварительного выбора предполагает минимальные временные затраты на синхронизацию ветвей задачи перед созданием контрольной точки.

Библиотека libcheckpoint

Была выполнена экспериментальная реализация библиотеки автоматизированных контрольных точек для параллельных задач, данная библиотека позволяет создавать контрольные точки параллельных задач, выполняющихся на кластерах под управлением ОС семейства UNIX и использующих библиотеку MPICH. Консистентность

контрольных точек обеспечивается при помощи предварительной синхронизации – данный метод позволяет обеспечить низкие задержки на синхронизацию ветвей, требует хранения лишь одного набора контрольных точек ветвей, и его реализация не приводит к изменению структуры обмена сообщений задачи (не требует применения служебных сообщений). В библиотеке были реализованы три метода повышения эффективности: сжатие данных, асинхронные контрольные точки и исключение ненужных областей данных из контрольной точки.

Основными характеристикам библиотеки являются:

- реализация автоматических контрольных точек на уровне процесса пользователя; сама библиотека libcheckpoint написана на языке C;
- поддержка различных архитектур: FreeBSD/x86, Linux/x86, Linux/Alpha;
- поддержка языков программирования C/C++, Fortran;
- сохранение состояния процесса (сегмент данных, стек, куча, контекст процесса);
- сохранение состояния операционной системы (отображения в память, открытые файлы, обработчики сигналов);
- миграция процессов между машинами с идентичной архитектурой (восстановление задачи на другом наборе узлов по сравнению с тем, на котором она работала до создания контрольной точки);
- поддержка библиотеки обмена сообщениями MPICH в трех коммуникационных средах:
  - разделяемая память (SMP-узлы);
  - сеть Myrinet;
  - сеть Ethernet.

Для повышения эффективности механизма контрольных точек в библиотеке применяется два метода: *сжатие данных* и *исключение ненужных областей памяти из КТ*. Использование сжатия данных контрольной точки основывается на предположении, что содержимое контрольной точки (т.е. данные программы) обладают избыточностью, и применение алгоритмов сжатия позволит существенно сократить объем КТ. Исключать из контрольной точки можно те области памяти, содержимое которых не понадобится при восстановлении задачи. Такие области можно охарактеризовать тем, что первое обращение после момента создания контрольной точки к данной области памяти будет на запись, а не на чтение. В текущей версии библиотека libcheckpoint позволяет лишь исключать указанные области памяти из контрольной точки или включать их обратно, она не может автоматически определять те области памяти, которые можно исключить, сохранив корректность выполнения задачи после восстановления из КТ.

Как уже отмечалось выше, при предварительной синхронизации требуется указать библиотеке контрольных точек места, в которых будут созданы контрольные точки. Для библиотеки libcheckpoint это означает, что в местах создания контрольной точки в каждой ветви программы должен быть произведен вызов функции создания КТ. Каким образом могут быть обнаружены такие места создания КТ? Можно предложить несколько методов:

Ручной выбор места синхронизации. Программист вручную размещает в исходном коде своей задачи обращения к функции создания контрольной точки. Вызовы должны быть размещены таким образом, что в момент создания контрольной точки все ветви параллельной задачи должны выполнить функцию создания контрольной точки. Также выбор места создания контрольной точки должен подчиняться критериям консистентности, приведенные в разделе «Введение». Такой выбор может оказаться нетривиальным, если схема обмена сообщениями между ветвями задачи является достаточно сложным. Однако, необходимо еще раз отметить, что существует достаточно большое количество ситуаций, когда можно легко указать место, удовлетворяющее критериям консистентности: после групповой операции MPI, затрагивающей все ветви задачи (например, MPI_Barrier), достаточно часто в итеративных вычислительных задачах консистентную контрольную точку можно создать в конце каждой итерации.

Статический анализ параллельной программы. Заранее, до запуска параллельной программы, собирается трасса параллельной программы. Путём анализа данной трассы можно получить рекомендации по расстановке консистентных контрольных точек, которые используются уже при запуске программы на счёт. Данный подход основан на анализе отдельных событий в параллельной программе, а также на анализе связей между ними.

Динамический анализ параллельной программы. В данной ситуации, во время работы параллельной программы, в случае необходимости создания контрольной точки (например, по истечении заданного интервала времени между создаваемыми контрольными точками), программа будет приостановлена в такой момент времени, когда возможно создание консистентной контрольной точки. Этот и предыдущий подход будут рассмотрены в рамках данной статьи.

Тестирование эффективности. Для тестирования эффективности библиотеки была выбрана вычислительная задача расчета свободномолекулярного обтекания цилиндра, написанная на языке Fortran-77. Тестирование производилось на супер-ЭВМ МВС-1000М.

Из полученных результатов следует, что в среднем издержки на создание контрольной точки такой задачи составляют 9,35 секунды, что

при интервале создания контрольных точек в 30 минут составляет 0,5% потери по времени. Коэффициент сжатия данных контрольной точки в данном эксперименте достигал 43%. Среднее время восстановления задачи составляет около 15 секунд, что является вполне приемлемым.

Данная реализация библиотеки контрольных точек не является единственной, существуют также другие [6,7] реализации систем контрольных точек, однако они обладают рядом существенных недостатков таких, как отсутствие поддержки параллельных задач, малая переносимость и отсутствие поддержки MPICH.

Статический анализ трассы параллельной программы

Общую схему данного подхода можно представить следующим образом: производится тестовый запуск параллельной задачи со сбором трассы параллельной программы, полученная трасса попадает на вход анализатору, который выдает рекомендации по расстановке контрольных точек в параллельной программе.

Для рассмотрения данного подхода нам потребуются дополнительные термины и понятия. *Момент отправки сообщения* – момент вызова одной из функций семейства MPI_*Send (нам не важен фактический момент отправки сообщения, т.к. отправка сообщения произойдет заведомо раньше, чем это сообщение будет получено, а в промежутке между отправкой и получением сообщения контрольная точка не может быть создана). *Момент приёма сообщения* – момент возврата из функции MPI_Recv или завершения асинхронной операции приёма MPI_Irecv. *Действие параллельной программы* – это операция по приему или отправке сообщения (мы не рассматриваем внутренние действия ветвей задачи, такие как вычисления, так как они не влияют на выбор места создания контрольной точки).

Детерминированность MPI-программы. Т.к. наш анализ основывается на некоторой статической характеристике задачи, такой как её трасса, мы должны делать предположения о том, что её трасса не изменяется от запуска к запуску или изменяется контролируемым образом. Источником недетерминированности для последовательной программы является её взаимодействие с окружением. Так как каждая ветвь параллельной программы представляет собой последовательную программу, её недетерминированность (т.е. различие трассы программы от запуска к запуску) может быть обусловлена:

1. обращением к недетерминированным вызовам MPI;
2. зависимостью от входных данных;
3. другими причинами (например, использованием датчика псевдослучайных чисел, взаимодействие с другими процессами, минуя средства MPI и т.п.).

Мы не будем рассматривать программы, в которых недетерминированность обусловлена третьим пунктом. Данный подход

всё же может быть применен к задачам, трасса которых изменяется в зависимости от входных данных, а недетерминированность, связанная с вызовом MPI-функций, может быть проанализирована.

Отметим, что в стандарте MPI отдельно отмечены источники потенциальной недетерминированности: ими являются неблокирующие операции и получение сообщения без указания отправителя или тега (MPI_ANY_TAG и MPI_ANY_SOURCE). Необходимо заметить, что такие вызовы могут оказаться детерминированными, например, вызов функции получения сообщение без указания отправителя при условии, что только одно сообщение может быть получено в данный момент. Для сообщений, отправленных от одного процесса другому с одним тегом порядок сохраняется (FIFO).

Под отсутствием зависимости трассы MPI-программы от входных данных понимается не отсутствие изменений в результатах работы программы и не отсутствие изменений в содержимом передаваемых через MPI сообщений, а неизменность последовательности событий по отправке и приёму сообщений.

Причинно-следственные зависимости и векторные таймеры. Введем понятие *действия параллельной программы*. Под действием параллельной программы мы будем понимать операцию по приему или отправке сообщения (мы не рассматриваем внутренние действия, такие как вычисления, так как они не влияют на выбор места создания контрольной точки). Определим отношение *причинно-следственного предшествования* между двумя действиями:

1. Два действия одного процесса причинно-следственно предшествуют, если одно следует за другим (во времени).
2. Действие по отправке сообщения одного процесса причинно-следственно предшествует соответствующему ему действию по приему этого сообщения в другом процессе.
3. Два любых действия причинно-следственно предшествуют, если существует цепочка действий, начинающаяся с первого действия и заканчивающаяся последним, такая что два любых соседних действия в этой цепочка причинно-следственно предшествуют друг другу в смысле пунктов 1 и 2.

Причинно-следственное предшествование на множестве действий параллельной программы является отношением частичного порядка. Действительно, оно антирефлексивно (очевидно) и транзитивно (третий пункт определения). *Трассой параллельной программы* будем называть последовательность её действий.

Отношение причинно-следственного предшествования очень удобно при анализе параллельных программ, оно позволяет

сформулировать простое необходимое условие консистентности, обнаружить соответствующие друг другу операции по приему и отправке сообщений, а также проводить анализ потенциально недетерминированных мест в параллельных программах. Однако выявление данного отношения является сложной задачей, поэтому при анализе используется еще одно понятие – векторный таймер.

В силу того, что упорядочение событий в параллельной программе при помощи часов невозможно, вводятся некоторые «виртуальные» часы, не позволяющие измерять промежутки времени, но позволяющие упорядочить действия параллельной программы (например, часы Лампорта [1]). Немного более сложным механизмом, также позволяющим упорядочить события в параллельной программе, являются векторные таймеры (подробное их описание можно найти, например, в [2,3]): каждая ветвь параллельной программы поддерживает целочисленный вектор длины n (где n – количество ветвей параллельной программы). При запуске программы этот вектор в каждой ветви обнуляется. Перед выполнением действия по отправке сообщения ветвь программы увеличивает «свою» компоненту таймера (т.е. компоненту с номером, соответствующим номеру ветви). После этого к передаваемому сообщению ветвь прикрепляет текущие показания своего таймера. При получении сообщения ветвь-получатель вычисляет новый таймер как покомпонентный максимум из показаний своего таймера и таймера, полученного вместе с сообщением, после чего увеличивает свою компоненту таймера на единицу.

Будем называть показания часов (векторного таймера), соответствующими действию параллельной программы, показания часов ветви, выполнившей действие, *после* выполнения данного действия. Тогда легко доказать утверждение, что если a и b – два любых действия параллельной программы, а T_a и T_b - соответствующими этим действиям показания часов, то a причинно-следственно предшествует b тогда и только тогда, когда $T_a < T_b$. Таким образом, векторные таймеры эквивалентны отношению причинно-следственного предшествования, однако при сборе трассы параллельной программы гораздо проще получить векторные таймеры, чем вычислить отношение причинно следственной зависимости.

Конечно, MPI-программа состоит не только из операций по приёму или отправке сообщений, в ней могут быть групповые операции, которые затрагивают сразу несколько ветвей параллельной задачи. Такие операции мы можем представить (для анализа) как такую суперпозицию операций по приему-отправке сообщений, которая накладывает такие же причинно-следственные зависимости, как и групповая операция. Например, операцию MPI_Bcast можно представить как отправку сообщений от одной ветви всем ветвям,

участвующим в операции, а MPI_Barrier – как операцию по отправке сообщений от всех процессов какому-то выделенному, а затем от него – обратно всем процессам.

Получение трасс MPI-программ. Средство получения трасс MPI-программ представляет собой библиотеку, которая подменяет большую часть вызовов MPI-функций. После записи сообщения в трассу о произошедшем действии параллельной программы оно вызывает подмененную функцию MPI уже из конкретной реализации MPI. Для каждого действия в трассе должен быть записан тип действия, номер ветви, в которой оно произошло, значение таймера, связанного с данным действием, а также дополнительная информация, зависящая от типа действия, так для действия по отправке сообщения это может быть адресат сообщения и т.п. Кроме того, средство получения трасс поддерживает в каждой ветви векторные таймеры, осуществляя их изменение при возникновении соответствующих событий, а также передавая значения таймера вместе с сообщениями параллельной программы, таким образом, что семантика обмена сообщениями параллельной программы не изменяется.

Алгоритм поиска консистентных контрольных точек. Для определения места создания контрольной точки параллельной задачи необходимо выбрать место создания КТ в каждой из её ветвей. Контрольная точка может быть размещена между любыми двумя соседними действиями в ветви. На практике создание контрольной точки может осуществляться непосредственно до или после действия (напомним, что под действием понимается приём или отправка сообщения). Такая комбинация из n выбранных мест может быть оценена с точки зрения удовлетворения критерия консистентности, а также с точки зрения других критериев, таких как:

- удобство для программиста исходной задачи: контрольные точки во всех ветвях создаются на одной и той же строчки исходного кода программы, при каждом входе на строчку может быть создана контрольная точка;
- эффективность: ожидаемое время синхронизации (время, которое потребуется, чтобы все ветви программы достигли места создания контрольной точки) является малым;
- интервал между контрольными точками: временной интервал между выбранными контрольными точками удовлетворяет требованиям, которые пользователь предъявил перед анализом, и т.п.

Сформулируем необходимое условие консистентности выбранного места создания контрольной точки, выразив его в терминах трассы параллельной программы. Пусть контрольная точка в ветви i создается после действия a_i и перед действием b_i. При вызове функции

создания контрольной точки будет выполнена синхронизация, т.е. программа будет приостановлена до тех пор, пока все её ветви не сделают вызов функции создания контрольной точки. Такой вызов они смогут сделать только тогда, когда все действия a_i завершатся; отметим, что в этот момент действия b_i еще заведомо не выполнятся. Тогда необходимое условие создания может выглядеть следующим образом: $\forall k,l = \overline{1,n} : a_l$ причинно-следственно *не* следует за b_k (легко показать, что в случае нарушения данного условия будет получена тупиковая ситуация). Достаточным условием возможности создания контрольной точки является завершенность всех операций по приёму и передаче сообщений.

Обозначим $\overline{e_i}$ - действие, непосредственно следующее за действием e_i в ветви *i* параллельной программы.

Рассмотрим самый простой вариант алгоритма поиска мест создания консистентных контрольных точек:

1. Рассмотрим все возможные наборы действий длины n, в которых все действия произошли в разных ветвях: $\{e_1, e_2, ..., e_n\}$. При этом, пусть действие e_i произошло в ветви *i*.
2. Если для $\forall i = \overline{1,n}$ в паре $\left(e_i, \overline{e_i}\right)$ оба действия являются детерминированными, а для набора действий $\{e_1, e_2, ..., e_n\}$ выполнено достаточное условие консистентности контрольной точки (т.е. оно выполнено после завершения этих действий), то контрольную точку можно создать в данном месте выполнения параллельной программы (в ветви *i* контрольную точку можно создать между действиями e_i и $\overline{e_i}$).

Данный переборный алгоритм может быть улучшен путём упорядочения перебора и отсечения бесперспективных вариантов на ранних этапах перебора. Так, если необходимое условие консистентности не выполнено для выбора места создания в *k* ветвях параллельной программы, то оно не будет выполнено и при любом выборе места создания в оставшихся *n-k* ветвях. Также используется и тот факт, что если место создания контрольной точки зафиксировано в одной из ветвей, а в другой будет ожидаться приём сообщения, которое должна была отправить первая ветвь, то такое место создания контрольных точек также недопустимо. Эти оптимизации позволяют существенно сократить перебор, однако на данный момент не удалось разработать полиномиального варианта данного алгоритма или доказать NP-полноту данной задачи.

Тестовый запуск. Как уже отмечалось ранее, трасса параллельной программы для анализа собирается при тестовом запуске параллельной задачи. «Идеальным» тестовым запуском является запуск исходной задачи, с теми же сами входными данными. Но с практический точки зрения такой вариант вряд ли приемлем, т.к. задача уже отработала, были получены результаты, и расставлять в ней контрольные точки бессмысленно. Поэтому для тестового запуска задача может быть изменена таким образом, чтобы сократить время работы параллельной программы, сохранив при этом обмены сообщениями. Так, для итеративной задачи, при условии, что все итерации одинаковые, может быть сокращено количество итераций, могут быть исключены участки вычислений и т.п. Если трасса программы зависит от исходных данных, может быть произведено несколько запусков в попытке обнаружить те места создания консистентных контрольных точек, которые сохраняются при каждом запуске задачи независимо от входных данных.

Динамический анализ трассы параллельной программы

Данный подход к автоматическому обнаружению мест создания консистентных контрольных точек основан на предположении, что при выполнении параллельной программы существуют моменты времени, когда удовлетворены критерии консистентности, приведенные в разделе «Введение». Во время работы программы, в случае возникновения события о необходимости создания контрольной точки, происходит анализ текущего состояния выполнения параллельной программы: будет ли консистентной контрольная точка, созданная в данный момент времени? Если текущее состояние является консистентным, контрольная точка может быть создана, иначе осуществляется «поиск» (т.е. выборочное продолжение выполнения параллельной программы) до тех пор, пока не будет обнаружено такое место, которое удовлетворяет критерию консистентности. Как только программа остановлена в таком состоянии, происходит создание контрольной точки.

Событие о необходимости создания контрольной точки может возникнуть в одной или нескольких ветвях параллельной программы. Оно может быть вызвано, например: истечением интервала между последовательными контрольными точками, сигналом от планировщика задач параллельной вычислительной системы, который сообщает задаче о том, что она будет снята со счета в ближайшее время, и предлагает создать контрольную точку. В любом случае, после возникновения такого рода события, все ветви параллельной задачи должны быть извещены о том, что должна быть создана контрольная точка, они должны перейти в особый режим выполнения, когда происходит анализ

консистентности текущего состояния задачи. Алгоритм такого анализа должен быть распределенным, децентрализованным.

Рассмотрим более подробно работу алгоритма. Во время работы параллельной программы (еще до появления события о необходимости создания контрольной точки) каждая ветвь программы собирает информацию о том, сколько сообщений данная ветвь отправила каждой ветви, а также о том, сколько сообщений она получила от каждой из ветви. Таким образом, ветвь i параллельной задачи за время своей работы построит два вектора: $a_i = [a_{i1}, a_{i2}, ..., a_{in}]$ и $b_i = [b_{i1}, b_{i2}, ..., b_{in}]$, где a_{ij} — количество сообщений, которое ветвь i отправила ветви j, а b_{ij} — количество сообщений, которое ветвь i получила от ветви j, а n – количество ветвей параллельной задачи.

В момент создания контрольной точки вся параллельная задача обладает матрицами $A = \{a_{ij}\}$ и $B = \{b_{ij}\}$ для $i, j = \overline{1,n}$, т.е. она обладает полной информацией о том, сколько сообщений каждая ветвь отправила каждой и сколько сообщений каждая ветвь получила от каждой (полная информация может быть получена путём обмена векторами между ветвями задачи). Теперь условие консистентности текущего состояния задачи может быть записано следующим образом: $\forall i = \overline{1,n}\ \forall k = \overline{1,n}\ a_{ik} = b_{ki}$. Очевидно, что условие $a_{ik} > b_{ki}$ невозможно, так как ветвь k не могла получить от ветви i больше сообщений, чем ветвь i ей отправила. Соответственно, единственным вариантом нарушения условия консистентности является: $\exists k_0, i_0 : a_{i_0 k_0} < b_{k_0 i_0}$.

Рассмотрим общую схему работы алгоритма:

1. Всё время работы параллельной программы (даже до появления события о необходимости создания контрольной точки) каждая ветвь задачи поддерживает информацию в векторах a_i и b_i.
2. После возникновения события о необходимости создания контрольной точки все ветви извещаются о том, что будет создаваться КТ и алгоритм переходит в «активную фазу». В этот момент ветви обмениваются собранными векторами a_i и b_i и получают матрицы A и B.
3. Если условие консистентности выполнено, то все ветви создают свои контрольные точки, которые в совокупности образуют консистентное множество контрольных точек для всей задачи. После создания КТ выполнение задачи продолжается обычным образом (пункт 1).

4. Если условие консистентности не выполнено, т.е. $\exists k_0, i_0 : a_{i_0 k_0} < b_{k_0 i_0}$, то выполнение ветви k_0 должно быть продолжено до того момента, пока не станет истинным $a_{i_0 k_0} = b_{k_0 i_0}$. Конечно, в результате выполнения ветви k_0 могут измениться другие элементы матриц A и B, что может привести к тому, что выполненные ранее равенства вида $a_{ik} = b_{ki}$ окажутся неверными. Тогда данный процесс будет продолжаться и далее, пока не будет выполнено условие консистентности, после чего алгоритм переходит к пункту 3.

Отметим, что алгоритм может зациклиться в пункте 4, если задача не содержит мест потенциального создания консистентных контрольных точек. К таким задачам он неприменим, и мы такие задачи рассматривать не будем, так как в них нет мест, удовлетворяющих критерию консистентности.

Такой алгоритм был реализован на уровне библиотеки пользователя, без изменения библиотек обмена сообщениями MPI. Для реализации используется интерфейс профилировщика, описанный в стандарте MPI [5]. Таким образом, реализация алгоритма будет получать управление при вызове любой функции MPI (возможна реализация дополнительных точек вызова, например, по сигналу от таймера). В «спящей» фазе своей работы при вызове функций MPI, связанных с отправкой и приёмом сообщений будет лишь обновляться информация в векторах a_i и b_i. Для реализации обмена служебными сообщениями, в частности, для извещения ветви задачи о том, что необходимо создать контрольную точку, все блокирующиеся методы MPI превращаются в неблокирующиеся, в которых будет участвовать, например, как событие получения сообщения для основной программы, так и событие получения служебного сообщения алгоритма. Если первым будет получено сообщение основной программы, результат будет передан программе, если служебное сообщение – его обработает алгоритм.

Заключение

Приведенные выше подходы позволяют обеспечить автоматическое создание консистентных контрольных точек в параллельной программе. К недостаткам статического анализа можно отнести потенциальную зависимость трассы программы от входных данных, недетерминированность, а также проблему подготовки корректного тестового запуска. Статический анализ всю работу производит на подготовительном этапе, не внося никаких издержек уже во время работы параллельной программы. Динамический анализ менее

требователен к исходной задаче, не зависит от изменения в обмене сообщениями от запуска к запуску задачи, но этот подход может приводить к задержкам в выполнении при попытке синхронизации перед созданием контрольной точки.

Наиболее перспективным выглядит второй подход – динамический анализ поведения программы во время выполнения с целью выявления мест создания консистентных КТ. Любой из этих подходов вместе с библиотекой контрольных точек libcheckpoint позволяет получить автоматическое средство создание контрольных точек, позволяющее с минимальными усилиями добавить функциональность контрольных точек к широкому классу вычислительных параллельных программ.

Литература

1. L. Lamport. Times, clocks, and the ordering of events in a distributed system. Communications of the ACM, 21(7): 558-565, July 1978.
2. C.J. Fidge. Partial orders for parallel debugging. Proceeding of the ACM SIGPLAN/SIGOPS Workshop on Parallel and Distributed Debugging, 24(1):183 194, January 1989. Published in ACM SIGPLAN Notices.
3. R.H.B. Netzer, B.P. Miller. Optimal tracing and replay for debugging message-passing parallel programs. In Proceedings of Supercomputing 92, 502-511, Minneapolis, MN, November 1992.
4. Машечкин И.В., Попов И.С., Смирнов А.А. Автоматизированная библиотека контрольных точек для кластерных вычислительных систем // Программные инструменты и системы: Тематический сборник фак-та ВМиК МГУ им. Ломоносова: №5 / под ред. Л.Н. Королева. – М.: Издательский отдел ф-та ВМиК МГУ, 2005. – с. 40-50.
5. Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. Version 1.1, June 1995.
6. James S. Plank, Micah Beck, Gerry Kingsley, Kai Li. Libckpt: Transparent Checkpointing under Unix. In Proceedings of the 1995 Winter USENIX Technical Conference. – UT-CS-94-242.
7. Sriram Sankaran, Jeffrey M. Squyres, Brian Barrett, Andrew Lumsdaine, Jason Duell, Paul Hargrove, Eric Roman. Parallel Checkpoint/Restart for MPI Applications. — http://crd.lbl.gov/~jcduell/papers/cr.pdf.

Горицкая В. Ю.

Алгоритмы планирования в многопроцессорных системах

Введение

Очень часто программистам и прикладным специалистам для оптимизации большого объема вычислений приходится использовать технологии параллельного программирования и многопроцессорные сиситемы. Эти системы часто состоят из большого количества процессоров – их число может измеряться тысячами. Безусловно, это затрудняет планирование и анализ поведения параллельных программ. Также появляются новые аппаратные платформы (такие, например, как мультиядерные архитектуры), при планировании в которых необходимо учитывать аппаратные особенности (напрмер, использование общего кэша).

В связи этим остро стоит вопрос об эффективности использования ресурсов мультипроцессора. Для решения этой проблемы существуют системы управления заданиями[1, 2, 3, 4], в чьи функции входит распределение ресурсов вычислительной системы по задачам, планирование и построение расписания выполнения задач пользователей.

Построение расписания выполнения заданий на вычислителе должно заключаться, прежде всего, в решении двух основных подзадач: определения очередности выполнения заданий и распределения процессоров между ними. Для решения первой подзадачи (определения очередности) существует множество алгоритмов планирования, которые могут использоваться, вообще говоря, и на уровне планирования очереди задач в многопроцессорной системе, и при планировании процессов на более низком уровне. Среди них можно выделить статические алгоритмы планирования и динамические. К статическим алгоритмам относятся две основных группы алгоритмов – пакетное планирование (**batch**) и планирование **Gang** (планирование групп пакетов). Динамические алгоритмы планирования в основном применяются для приложений, где в днамике работы программы возможно изменение числа паралельных нитей (например, OpenMP-приложения [5]).

В данной работе приведен обзор алгоритмов планирования вычислений. Основное внимание уделено планированию процессов и

задач в реальных многопроцессорных системах, приведены примеры организации систем управления заданияит конкретных вычислителей.

Простейшие алгоритмы

Приведем сначала несколько примеров самых простых алгоритмов планирования процессов[6, 7].

Данные алгоритмы составляют базу для планирования работы процессов, вообще говоря, на любом уровне вычислительной системы, – будь то приложения пользователей или же процессы операционной системы. Чаще всего они используются для планирования процессов в системах реального времени. Для построения расписания выполнения задач пользователей используются и более сложные механизмы, позволяющие в зависимости от текущей загрузки и типов задач переключаться между различными алгоритмами или использовать подход, построенный на основе самых простых способах разделения вычислительных ресурсов.

FCFS (First Come First Served, First In First Out - FIFO) – самый простой механизм, реализующий обычную очередь. Если текущая задача по каким-то причинам прерывается, то начинает выполняться следующая; когда "старая" задача снова готова к выполнению, она ставится в конец очереди. Очевидно, что этот алгоритм будет давать приемлемые результаты по эффективности работы системы в случае, если все задачи в ней требуют одинаковых вычислительных ресурсов – это касается как количества процессоров (иначе неизбежна ситуация простоя некоторых процессоров), так и времени работы задачи (иначе время ожидания "коротких" задач может во много раз превышать время их выполнения).

RR (Round Robin) позволяет уменьшить время ожидания "коротких" задач в очереди с помощью таймера. Данный алгоритм используется в операционной системе Linux для планирования процессов. При обнулении таймера выполняющаяся задача прерывает свое выполнение и отправляется в конец очереди. Следующая готовая к выполнению задача выбирается с помощью FCFS.

SPN (Shortest Process Next). В этом алгоритме невозможно прервать или приостановить задачу, она должна завершиться сама. Следующим на выполнение выбирается процесс с наименьшим предполагаемым временем выполнения (при запуске указывается максимальное время работы задачи). Таким образом также можно уменьшить время ожидания в очереди "коротких" задач.

SRT (Shortest Rest Time) работает так же, как и SPN, но здесь возможна приостановка выполнения задач. По истечении определенного кванта времени выполняющаяся задача приостанавливается и отправляется обратно в очередь. Из очереди

планировщик выбирает процесс с наименьшим ожидаемым временем до конца его работы. В момент времени, когда новый процесс попадает в очередь, оставшимся временем является предполагаемое время выполнения (см. SPN).

Планирование с использованием приоритетов (Priority Scheduling) предполагает связывание с каждой задачей определенного численного значения. Возможны три ситуации: приоритет приписывается задаче один раз при ее постановке в очередь на выполнение (статический приоритет); приоритет может меняться в процессе работы приложения, например, в зависимости от параметров выполнения задачи (динамический приоритет); может использоваться также смешанный подход, комбинирующий статический и динамический механизмы.

В каждой конкретной системе планирования приоритеты вычисляются по-разному, при этом могут учитываться различные свойства задачи, например,принадлежность ее владельца определенной группе пользователей, количество запрошенных процессоров или максимальное время счета. Пример расчета значения приоритета P(x) для задачи x:

$$P(x) = ClassSysprio * Weight_1 + UserSysprio * Weight_2 + GroupSysprio * Weight_3 + QDate*Weight_4$$

При таком планировании задача выполняется, пока она не прервется (по каким-либо внутренним причинам, независимо от планировщика) или пока в очереди не появится задача с более высоким приоритетом. Заметим, что в данном случае очередь задач не является очередью в обычном понимании (это т.н. приоритетная очередь), - она может быть реализована как однонаправленный список или как система из нескольких очередей FIFO.

Gang-планирование

Алгоритм **GANG** [8, 9] представляет собой комбинацию двух основных подходов к планированию вычислений: разделение времени (каждой задаче выделяется временной слот; задача выполняется в течение этого временного слота; по истечении выделенного времени будет выполняться другая задача и т. д.) и разделение процессоров (на вычислителе выполняется одновременно несколько приложений). Алгоритмы с разделением процессоров значительно сокращают накладные расходы по переключению контекста и потери эффективности выполнения при синхронизации.

Согласно Gang, все процессы объединяются в группы (gangs), внутри этих групп процессы выполняются одновременно (разделение процессоров), а между группами реализуется разделение времени.

Существуют способы повысить эффективность Gang-планирования, рассмотрим один из них.

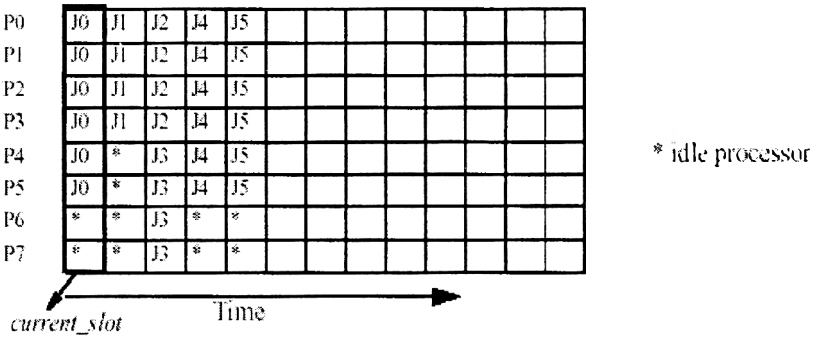


Рис. 1. Матрица расписания выполнения задач в системе из 8 процессоров

P0	J0	J2	J4			
P1	J0	J2	J4			
P2	J0	J2	J4			
P3	J0	J2	J4			
P4	J0	J3	J5			
P5	J1	J3	J5			
P6	J1	J3	J5			
P7	J1	J3	J5			

Рис. 2. Матрица, полученная из матрицы рис. 1 применением алгоритма Compress&Join

Compress&Join [9]. Целью данного алгоритма является минимизация расходов на переключение контекста процессов между временными промежутками, в течении которых выполняются процессы одной группы (т.н. временные слоты), путем уменьшения числа этих временных промежутков. При приходе новой задачи в систему не выделяется новой группы и, соответственно, нового временного слота для этой группы, а сначала сокращается число выделенных процессоров для уже выполняющихся задач (Compress) и затем новая задача "занимает" освободившееся "место" внутри уже существующих групп (Join) (см. рис. 1 и 2, показаны матрицы выполнения задач на 8ми

процессорной системе, временные слоты – по горизонтали, по вертикали – распределение по процессорам; в результате применения алгоритма Compress&Join число временных слотов уменьшилось с 5 до 3, таким образом, удалось сократить потери от переключения контекстов и синхронизации).

Динамические алгоритмы планирования

Динамические алгоритмы планирования задач [8] предполагают изменения в расписании выполнении задач в процессе их счета. Эти изменения касаются распределения процессоров между задачами. Чтобы достичь эффективного использования системы, наибольшее число процессоров выделяется задаче, достигшей в процессе своего выполнения большего уровня эффективности. Этот механизм может быть реализован далеко не во все системах: необходима поддержка масштабируемости приложений. Эту возможность предоставляет, например, OpenMP, - по ходу выполнения программы можно изменять число параллельных нитей.

Кроме того, работа этих алгоритмов осложняется необходимостью постоянной работы специальных агентов, которые должны осуществлять мониторинг выполнения каждой задачи и производить соответствующие вычисления эффективности работы того или иного приложения. Из-за сложности практической реализации этих алгоритмов они не используются в реально существующих системах планирования вычислений, а необходимые исследования проводятся на имитаторах реальных вычислительных систем, и в этих условиях такие алгоритмы показывают очень хорошие результаты (см. рис. 3 - показана величина средней потери производительности в 60, 80 и 100% случаев для алгоритмов Batch, Equipartition, Equip++, PDPA и GANG [8]).

Equipartition. Изначально всем задачам выделяется равное количество процессоров, если это возможно. Затем планировщик циклически добавляет процессоры задачам до тех пор, пока все задачи не получат требуемое количество процессоров или не будет свободных процессоров для распределения. При таком алгоритме повышается эффективность использования многопроцессорной системы, поскольку сводятся к минимуму ситуации, когда система будет простаивать.

Equip++. Данный алгоритм представляет собой модификацию Equipartition. В процессе распределения ресурсов между задачами измеряется эффективность выполнения каждого приложения, если она не ниже некоторого уровня, то задаче выделяется еще один процессор. Если эффективность приложения ниже этого уровня, то дополнительный процессор не выделяется (способ вычисления эффективности может производиться в разных системах по-разному). Алгоритм не очень хорошо подходит для планирования задач, которые

рассчитаны на строго определенное количество процессоров и достигают высокой эффективности, когда выполняются именно на таком количестве процессоров.

PDPA (Performance-Driven Processor Allocation). Согласно этому алгоритму, каждой задаче, которая достигла нужного уровня эффективности, вычисленного во время выполнения, выделяется столько процессоров, сколько потребовала задача во время запуска. Остальные задачи с низкой эффективностью "разбирают" оставшиеся незанятыми процессоры. Аналогично Equip++, алгоритм «невыгоден» для задач, достинающих своей эффективности на определенном количестве процессоров.

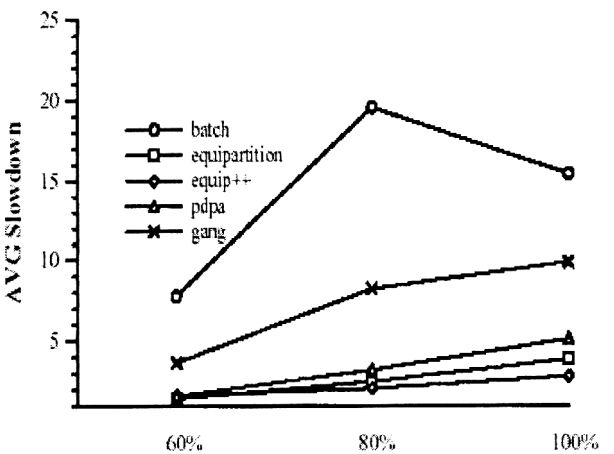


Рис. 3. Снижение эффективности счета задач при использовании разных стратегий планирования вычислений (сравнение алгоритмов, использующих механизм разделения времени, и алгоритма GANG)

«Оптимизационные» алгоритмы планирования вычислений

Существует множество других алгоритмов планирования процессов, однако из-за простоты реализации на практике чаще применяются описанные выше простые "списочные" алгоритмы или же планирование Gang.

Как правило, более сложные механизмы планирования основываются на различных решениях задачи распределения N заданий на M процессоров с целью минимизировать, например, общее время

работы всего набора заданий. Известно, что эта проблема является NP-трудной[10, 11].Если вычислитель состоит из очень небольшого числа процессоров, то приемлемым и самым простым решением является полный перебор всевозможных вариантов расписания и выбора среди них наилучшего. К сожалению, для большого количества процессоров такой метод не применим. Существующие алгоритмы решения этой задачи предоставляют возможности уменьшения сложности перебора, но, как правило, эти алгоритмы имеют свои ограничения и требуют достаточно большого времени работы.

Алгоритм имитации отжига основывается на имитации физического процесса, который происходит при кристаллизации вещества из жидкого состояния в твёрдое. Предполагается, что атомы уже выстроились в кристаллическую решётку, но ещё допустимы переходы отдельных атомов из одной ячейки в другую. Предполагается, что процесс протекает при постепенно понижающейся температуре. Переход атома из одной ячейки в другую происходит с некоторой вероятностью, причём вероятность уменьшается с понижением температуры. Устойчивая кристаллическая решётка соответствует минимуму энергии атомов, поэтому атом либо переходит в состояние с меньшим уровнем энергии, либо остаётся на месте. Примерение алгоритма имитации отжига для построения расписания выполнения задач в многопроцессорной системе описано в [12].

Одним из популярных способов составления расписаний для многопроцессорных систем является способ составления при помощи **генетического алгоритма** [13, 14]. Генетический алгоритм – это алгоритм поиска максимума или минимума некоторой функции с помощью направленного перебора области определения функции. Перебор проводится методами схожими с законами естественного отбора. Концепция построения генетических алгоритмов была предложена Холландом [13]. Схематично работа ГА состоит в следующем:

1. Сгенерировать случайным образом популяцию размера P;
2. Вычислить целевую функцию для каждой строки популяции;
3. Выполнить операцию селекции;
4. Выполнить операцию скрещивания;
5. Выполнить операцию мутации;
6. Если критерий останова не достигнут, перейти к шагу 2, иначе завершить работу.

Популяция - это множество битовых строк. Каждая строка представляет в закодированном виде одно из возможных решений задачи. По строке может быть вычислена целевая функция, которая характеризует качество решения. В качестве начальной популяции может быть использован произвольный набор строк. Основные операции алгоритма: селекция, скрещивание и мутация выполняются

над элементами популяции. Результатом их выполнения является очередная популяция. Данный процесс продолжается итерационно до тех пор, пока не будет достигнут критерий останова.

Системы управления потоком задач

Основная функция систем управления заданиями в многопроцессорных системах – планирование вычислений. Это планирование обычно включает

- постановку задач в очередь,
- построение расписания выполнения задач,
- мониторинг состояния очереди,
- управление ресурсами вычислителя,
- учет и документирование использования ресурсов.

Рассмотрим кратко принципы работы некоторых из них, обращая внимание на программно-аппаратные платформы, на которые ориентирована система, и используемые алгоритмы планирования вычислений (примеры алгоритмов см. выше).

IBM LoadLeveler [1]– система управления заданиями, позволяющая эффективно распределять выполнение заданий в многопроцессорной системе. Это достигается путем анализа и сопоставления ресурсов, требующихся для счета задачи, и доступных ресурсов системы. LoadLeveler строит расписание выполнения задач, обеспечивает быстрое и эффективное выполнение компоновки модулей, постановки задачи в очередь и счета задачи в динамике работы вычислительной системы.

LoadLeveler может работать на гетерогенных кластерах, выделенных узлах, а также RISC System/6000(R) и Scalable POWERparallel(R) System (SP) (главное требование – наличие операционной системы AIX).

LoadLeveler оперирует таким понятием, как LoadLeveler-кластер, - это комбинация всех типов машин, которые используют LoadLeveler (*машина-планировщик (Scheduling Machine), управляющая машина (Central Manager Machine), машина-исполнитель (Executing Machine), поставщик заданий (Submitting Machine)*). Каждая машина в LoadLeveler-кластере исполняет одну или несколько функций построения расписания выполнения заданий. Работу LoadLeveler обеспечивает набор демонов, контролирующих процессы, которые перемещают задания внутри LoadLeveler-кластера.

OpenPBS (Portable Batch System) [3] - довольно гибкая система планирования заданий (была разработана для NASA в начале 1990х годов). Система может работать на различных аппаратных платформах, поддерживающих Unix-подобные операционные системы

(кластеры, рабочие станции, системы с массовым параллелизмом (MPP)). OpenPBS включает в свой состав несколько алгоритмов планирования вычислений (таких, как циклический Round Robin, прямой алгоритм FIFO, очередь, поддерживающая приоритеты задач), возможна классификация задач (разбиение задач на приоритетные группы). При установке системы администратор может выбрать и настроить должным образом тот алгоритм, который, по его мнению, наиболее подходит данному вычислителю. OpenPBS включает четыре основные компоненты:

- *команды* (поддерживается как командная строка, так и графический интерфейс);
- *сервер задач* (Job Server) – система управления задачами, в его функции входит получение (создание) задачи, изменение задачи, защита задачи от системных сбоев и отправка задачи на вычислитель;
- *исполнитель задач* (Job Executor) – демон, который отправляет задачу на вычисление;
- *планировщик задач* – реализует выбранный алгоритм планирования вычислений.

SGE (Sun Grid Engine) [2] - система планирования очередей в GRID системах (также может работать на отдельных кластерах и других многопроцессорных системах и суперкомпьютерах, поддерживающих Unix-подобные операционные системы). В силу специфики GRID-систем SGE может поддерживать систему из нескольких очередей. В данном контексте очередь следует понимать как хранилище для задач, обладающих сходными свойствами; эти задачи выполняться одновременно на определенном вычислителе. Свойства, или требования, которым удовлетворяет задача, определяются при отправке пользователем задачи на вычислитель – в качестве параметров запуска указываются ограничения на необходимые ресурсы для выполнения задания. После анализа этих свойств "вновь прибывшая" задача отправляется в подходящую очередь на наименее загруженный вычислитель. В рамках конкретного вычислителя допускается одновременное выполнение задач. В первую очередь на счет отправляются задачи, имеющие наибольший приоритет или дольше всех находившиеся в очереди.

Основными компонентами SGE являются:

- *хосты* (hosts) – основной хост, хост-администратор, хост-исполнитель и хост для отправки заданий на счет (по умолчанию основной хост сочетает в себе все функции);
- *демоны* (основной демон – управляет работой остальных демонов, демон-планировщик – ответственный за распределение задач по разным очередям. демон-исполнитель –

следит за состоянием очередей на определенном вычислителе, за выполнением задач на этом вычислителе, сообщает о статусе задач и о загрузке машины основному демону, демон-коммуникатор предоставляет единый сервис для всех взаимодействующих компонент системы (основан на протоколах TCP));

- *очереди* в SGE представляют собой структуру-контейнер, способную содержать разнородные по природе очереди с различных вычислителей; таким образом, реализуется единое пространство обработки заданий для вычислителей (произвольно) сложной архитектуры;
- *команды* – стандартные команды систем управления ресурсами, реализующие постановку и удаление задач из очередей, просмотр очереди и состояния задачи и т.д.

Condor [4] – система планирования вычислений, базирующаяся на алгоритме планирования DAGman (Directed Acyclic Graph Manager). Этот алгоритм отправляет задачи на счет согласно направленному ациклическому графу (DAG). Пример такого графа представлен на рис.4.

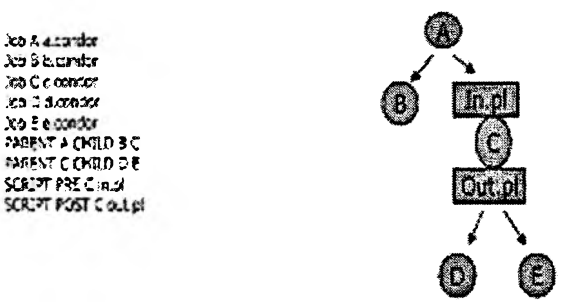


Рис. 4. Condor Directed Acyclic Graph.

Для формирования графа DAG нужно специфицировать связи и зависимости между задачами следующим образом:

- объявление "Job" связывает с именем задачи файл .condor, который содержит полное описание задачи;
- оператор "Parent-child" задает связь между двумя или более заданиями;
- специальные задания "PRE" и "POST" выполняются на хосте, отвечающем за отправку заданий на счет, и проводят пред- и постобработку заданий пользователей.

Очереди задач, формируемые Condor, не являются очередями FIFO. Всем задачам присваиваются приоритеты для определения порядка начала их выполнения. Могут использоваться прерывания выполняемой задачи для выполнения задания с большим значением приоритета, причем обычно считаются не абсолютные значения приоритетов, а относительные (по отношению к другим задачам). Кроме того, системой Condor не допускается "зависание" задач.

Рассмотренные системы управления заданиями, используемые обычно на мощных вычислителях [1, 2, 3, 4], обладают очень широкими возможностями для обеспечения эффективности функционирования многопроцессорной системы: можно применять различные настройки параметров алгоритмов планирования под реальный вычислитель и поток задач с определенными свойствами, алгоритм планирования может меняться в реальном времени в зависимости от изменений характеристик потока. Однако эти правила изменения алгоритма и различные настройки четко определены - они являются параметрами планировщика. Поэтому существует необходимость системы, способной самостоятельно создавать правила на основе данных о текущей загрузке и конфигурации вычислителя, о свойствах задач, выполняющихся на нем.

Управление заданиями в реальных системах

Рассмотрим кратко, как организовано планирование вычислений в реальных многопроцессорных системах.

МВС-15000 [15]– мультипроцессор Межведомственного Суперкомпьютерного Центра, состоит из 924 процессоров, работает под управлением ОС Linux.

Все задачи пользователей делятся на три категории – *отладочные, пакетные и фоновые.*

Отладочные задачи – это короткие по времени задачи, которые запускаются исключительно в целях отладки.

Пакетные задачи – это средние по времени задачи, которые производят реальные расчеты и выполняются, не прерываясь.

Фоновые задачи – задачи с большим временем счета, которые могут прерываться системой. Для фоновой задачи пользователь должен явно указать *квант* – минимальное время счета фоновой задачи, в течение которого задачу прерывать нельзя.

Планирование очередей в каждый момент времени производится в соответствии с параметрами текущего *режима планирования.* В системе может быть несколько *расписаний* режимов планирования, задающих некоторые общие параметры и определяющих все множество доступных на данный момент режимов. Переключение

расписаний осуществляется по директиве оператора, а переключение режимов внутри расписания — автоматически.

Режим планирования определяется следующими параметрами:

- дата и время включения режима - определяют время, начиная с которого параметры режима вступают в силу; параметры режима действуют вплоть до включения следующего режима;
- максимальное время, отведенное для отладочных задач;
- максимальное время, отведенное для пакетных задач;
- число процессоров, которые «резервируются» для отладочных задач (процессоры числом, указанным в данном параметре, в текущем режиме будут использоваться только для счета отладочных задач);
- шкала приоритетов пользователей (задача пользователя с высоким приоритетом может посчитаться раньше, чем задача пользователя с низким приоритетом; приоритет пользователя определяется по указанной шкале (с нулем в качестве признака конца) и напрямую зависит от суммарного времени счета пользователя за *учетный период*, который задается одинаковым для всех режимов текущего расписания).

Планировщик пытается выделить процессоры из числа свободных сначала для счета задач из очереди самых приоритетных задач, потом — из очереди задач чуть меньшего приоритета и т.д. Внутри одной очереди ресурсы выделяются в порядке (от меньшего к большему) номеров в *списке задач*, в котором находятся все считающиеся и ожидающие задачи. Если свободных процессоров для текущей задачи нет, определяется момент, когда нужное число процессоров будет доступно, и устанавливается время старта данной задачи. Никакая менее приоритетная задача не может занять процессоры так, что это бы отодвинуло старт более приоритетной. Однако, при отсутствии конфликта по ресурсам менее приоритетная задача может стартовать раньше более приоритетной. В случае конфликта по ресурсам опережающий старт менее приоритетной задачи возможен лишь, если ее время счета таково, что она освободит ресурсы раньше запланированного старта более приоритетной.

Как видно, механизм планирования задач в МВС-15000 довольно сложный, на практике большие вычислительные задачи могут сутками стоять в очереди на выполнение (ситуация усугубляется еще и тем, что приоритет владельца задачи уменьшается вместе с увеличением общего времени счета задач данного пользователя).

Следующие две многопроцессорные системы находятна на факультете ВМиК МГУ им. Ломоносова.

IBM eServer pSeries 690 (Regatta) [16]– вычислительная система, имеющая 16 Power4 процессора (64GB памяти). Запуск заданий происходит через стандартный планировщик IBM LoadLeveler

[1]. При постановке задания в очередь, помимо других параметров, пользователь может указать число процессоров и максимальное время счета (по умолчанию 1 процессор и 10 минут). При наличии достаточного числа свободных процессоров задание начнет выполняться. После постановки на счет задание будет выполняться не больше указанного времени, если оно не успеет завершиться само - его удалит планировщик. Задания согласно указанному максимальгному времени работы делятся на классы задач с разным приоритетом: чем меньше время – тем больший приоритет приписывается задаче. На данный момент поддерживается четыре класса задач: задачи продолжительностью не более 5 минут, от 5 до 30 минут, от 31 минуты до 12 часов и задачи продолжительностью более 12 часов.

IBM eServer pSeries 360 (Hill) - вычислительная система, представляющая собой шестипроцессорный кластер, работающий под управлением операционной системы LINUX. Планирование и запуск заданий осуществляется планировщиком Sun Grid Engine (SGE) [2].

Как уже было сказано, SGE позволяет поддерживать систему из нескольких очередей. На данной машине существует две очереди – «приоритетная» и «продолжительная», соответствующие двум режимам выполнения программ. Первый режим предназначен для учебных задач и задач оценки производительности. Второй режим - для задач, выполняющихся в течении длительного времени. При расчёте задач первого типа задачи второго типа приостанавливаются на запрошенных задачами первого типа узлах. Время работы задач первого типа ограничено 10 минутами согласно календарному времени (по истечении - снятие), второго типа – не ограничено.

Следует отметить, что SGE настроен таким образом, что задача получает в своё полное распоряжение выделенное (и запрошенное) ей количество процессоров. При этом поддерживается очередь задач для распределения на процессоры, если в данный момент нужного количества не имеется.

Иерархические многоуровневые системы управления заданиями в GRID

Для полноты вопроса необходимо заметить, что в последнее время активно развивается новый вид многопроцессорной архитектуры – GRID-системы. Поскольку локальные GRID-узлы территориально удалены друг от друга, то здесь требуются немного другие методы распределения ресурсов для выполняющихся задач. В таких системах планирование осуществляется метапланировщиками (планировщиками планировщиков), а на локальных узлах работают уже известные нам системы управления заданиями. Проблемы планирования в grid-

системах заключаются в том, что метапланировщики, во-первых, не контролируют использование ресурсов – предполагаеся, что этим озабочен пользователь, и ресурсы контролируются локальными планировщиками; во-вторых, в таких системах поддерживается резервирование времени (то есть заранее заказывается время, когда задача должна начать выполняться), что не совсем соответствует концепции пакетного планирования. Кроме того, часто возникают проблемы в случае, если параллельно выполняющиеся части рапределенного приложения должны обменитваться данными.

В качестве примера такого Grid-вычислителя можно привести систему **DEISA (Distributed European Infrastructure for Supercomputing Applications)** [17]. Вычислительные узлы этой системы находятся в Германии, Франции, Италии, Финляндии, Испании и Нидерландах. Архитектура системы поддерживает два уровня вычислений: «внутренний» (для жестко связанных по данным, однородных заданий), состоящий из схожих по характеристикам многопроцессорных систем, и «внешний», представляющий собой объединение разнородных вычислителей. На локальных вычислителях установлены стандартные пакетные планировщики – SGE, LoadLeveler, PBSPro, LSF и т.д. Распределение распределенных задач по удаленным узлам осуществляется сетевым планировщиком, который отслеживает загруженность вычислителей, анализирует информацию о задаче (главным образом, интересна интенсивность обмена данными) и принимает решение о назначении задачи на тот или иной узел Grid-системы.

На основе проведенного обзора можно сделать вывод, что задача планирования в многопроцессорных системах сложна и эффективное ее решение требует тщательного изучения особенностей архитектуры вычислителя и потока задач, выполняющихся на нем. Простые алгоритмы организации вычислений могут показаться недостаточно эффективными; алгоритмы, оптимизирующие использование ресурсов часто трудно реализуемы и эффективно работают только применимо к однороджному потоку задач. Системы управления заданиями, использующихся на большинсве многопроцессорных систем, способны поддерживать разные режимы работы в зависимости от характеристик выполняющихся программ, но не имеют гибкого механизма настройки и изменения параметров этих режимов. Как следствие, работа планировщиков не всегда удовлетворяет пользователей, чьи программы ожидают начала своего выполнения часами и даже сутками. Поэтому существует необходимость механизма плинирования, который при небольших накладных расходах был бы способен самостоятельно создавать расписание выполнения задач на основе информации о свойствах вычислителя и потока задач.

Литература

1. International Business Machines Corporation, IBM LoadLeveler for AIX 5L: Using and Administering Version 3 Release 2. [PDF](http://www.ibm.com/Redbooks/am2ug300.pdf)

2. Sun Microsystems, Inc. Sun ONE Grid Engine: Administration and Users Guide. Sun ONE Grid Engine, Enterprise Edition Administration and Users Guide. Sun ONE Grid Engine and Sun ONE Grid Engine, Enterprise Edition Reference Manual. [PDF](http://www.sun.com/GridWare.pdf)

3. Veridian Information Solutions, Inc. Portable Batch System. OpenPBS Release 2.3 Administration Guide. [PDF](http://www.openpbs.org/docs/v2.3 admin.pgf)

4. Douglas Thain, Todd Tannenbaum, Miron Livny. "Condor and the Grid", in Fran Berman, Anthony J.G. Hey, Geoffrey Fox, editors, Grid Computing: Making The Global Infrastructure a Reality, John Wiley, 2003. ISBN: 0--470-85319—0

5. OpenMP: Simple, Portable, Scalable SMP Programming (http://www.openmp.org/)

6. M. Saleh, Z. Othman, and S. Shamala «FCFS Priority-based: An Adaptive Approach in Scheduling Real-Time Network Traffic» Networks and Communication Systems proceeding 527, 2006, ISBN 0-88986-590-6.

7. RIOT: Remote Interactive Optimization Testbed (http://riot.ieor.berkeley.edu/index.html)

8. Julita Corbalan, Xavier Martorell and Jesus Labarta. Improving Gang scheduling through job performance analysis and malleability.

9. Shoukat Ali, Howard Jay Siegel, Muthucumaru Maheswaran, Debra Hensgen and Sahra Ali. Representing task and machine heterogeneities for heterogeneous computing systems. Tamkang Journal of Science and Engineering. 2000. Vol. 3, №3, P. 195-207.

10.Д.Р. Гончар. Мультиоценочный эвристический алгоритм распределения M заданий на N процессоров. //Труды научной конференции "Методы и средства обработки информации", изд. отдел ф--та ВМиК МГУ им. М.В. Ломоносова, 2005, ISBN 5--89407--230--1, стр. 537 – 540

11.Д.С. Гуз, Д.Р. Красовский, М.Г. Фуругян. Эффективные алгоритмы планирования вычислений в многопроцессорных системах реального времени. //Труды научной конференции "Методы и средства обработки информации", изд. отдел ф--та ВМиК МГУ им. М.В. Ломоносова, 2005, ISBN 5--89407-230--1, стр. 540 – 545.

12.Костенко В.А., Калашников А.В. Исследование различных модификаций алгоритмов имитации отжига для решения задачи построения многопроцессорных расписаний// Дискретные модели в теории управляющих систем. //Труды VII Международной конференции. М.: МАКС Пресс, 2006. - С.179-184.

13.Holland J.N. Adaptation in Natural and Artificial Systems. Ann Arbor, Michigan: Univ. of Michigan Press, 1975.

14.В.А. Костенко. Алгоритмы оптимизации, опирающиеся на метод проб и ошибок, в совместном проектировании аппаратных и программных средств ВС. //Труды Всероссийской научной конференции "Высокопроизводительные вычисления и их приложения" -М.: Изд-во МГУ, 2000, С.123-127.

15.Межведомственный Суперкомпьютерный Центр Российской Академии Наук (http://www.jscc.ru/)

16.Moscow State University Regatta Community, (http://www.regatta.cs.msu.su/)

17.DEISA - Distributed European Infrastructure for Supercomputing Applications (http://www.deisa.org/)

Герасимов С.В., Качуровский М.А., Киреев А.А.

Технология создания проблемно-ориентированных пользовательских интерфейсов[1]

1. Введение

Создание удобных пользовательских интерфейсов, позволяющих пользователям программных систем эффективно решать прикладные задачи, относится к разряду не решенных на текущий момент проблем. Об этом свидетельствует значительное число публикаций, например [1], [2], содержащих критические отзывы об удобстве использования широко известных и популярных сегодня программ. Вот некоторые из причин появления продуктов с неудовлетворительным качеством пользовательских интерфейсов.

- Разработка пользовательских интерфейсов производится программистами, а не специалистами-проектировщиками *взаимодействия пользователя с системой*, обладающими знаниями и опытом в таких областях, как эргономика и психология труда.
- Нежелание или непонимание руководством проекта необходимости в создании *прототипов пользовательского интерфейса* и их обсуждения с заказчиком до начала реализации.
- Недостаточная поддержка процесса разработки пользовательских интерфейсов CASE-технологиями, шаблонами проектирования, средствами автоматизированной генерации кода, позволяющих минимизировать усилия, затрачиваемые на рутинные фрагменты разработки.

При создании программных систем, предназначенных для решения специализированных прикладных задач, ситуация усложняется необходимостью разработки нестандартных предметно-ориентированных пользовательских интерфейсов и сопутствующего изучения малознакомой предметной области.

В данной статье предлагается технология разработки предметно-ориентированных пользовательских интерфейсов информационных систем. Технология была использована при построении системы «Интеллектуальная история болезни клиники неотложной кардиологии».

[1] Работа поддерживается грантом РФФИ 06-01-00691.

2. Особенности предметной области

Работа по созданию системы проводится в содружестве с клиникой неотложной кардиологии НИИ СП им. Н.В.Склифосовского. Научно-практическая деятельность клиники направлена на борьбу с последствиями инфаркта миокарда и приступов нестабильной стенокардии. Оба состояния характеризуются стенозом коронарных артерий (сужением, образованием т.н. «бляшек»), приводящих к тяжелым последствиям: ишемии сердца, а при инфаркте миокарда в 70% случаев к летальному исходу. Для борьбы с последствиями инфаркта и нестабильной стенокардии сотрудники клиники разработали и применяют метод *отложенной ангиопластики*. Суть метода заключается в проведении *через 24 и более часов* после инфаркта нетравматичного для организма воздействия, выполняемого под местной анастезией, заключающегося в расширении «бляшки» и установке на ее место специального каркаса-стента[1], препятствующего сужению артерии в будущем. В мировой практике для достижения тех же целей применяются методы *неотложной ангиопластики* (в сроки от 12 до 24 часов после инфаркта) и *тромболитической терапии (ТЛТ,* медикаментозное воздействие на стеноз).

3. Задачи системы

Разрабатываемая система предназначена для решения следующих задач.

- Атоматизация научных исследований в области апробации и обоснования методов борьбы с последствиями стеноза.
- Поддержка в принятии решений в рамках практической деятельности клиники. Система должна предоставлять функции ввода, хранения и доступа к данным электронных историй болезни (ЭИБ), а также решать задачи анализа накопленных данных:
    - прогнозирование максимального времени ожидания пациента;
    - выбор наиболее приоритетных пациентов из списка ожидания.

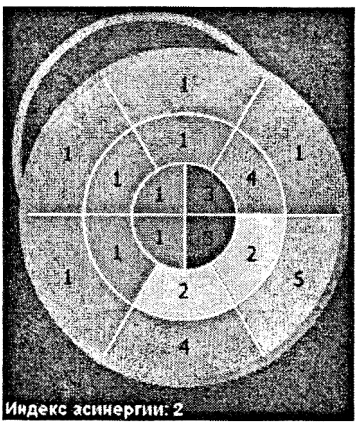
4. Анализ бизнес-процессов

При первоначальном анализе бизнес-процессов клиники была получена следующая информация. На каждого пациента, попадающего в клинику, заводится бумажная история болезни. Изначально заполняются ее разделы «Паспортная информация» и «Анамнез».

---

[1] Более подробная информация доступна на сайте о стентировании http://www.stent.ru

Пациент проходит через стандартный набор исследований, анализов и мониторингов и, в случае удовлетворительных результатов, подвергается одному из воздействий, призванному избавить его от стеноза. В первые часы после воздействия и в течение последующих нескольких месяцев проводятся повторные исследования, позволяющие отследить ближайшие и отдаленные результаты лечения.



Типичный набор исследований выглядит следующим образом:

- эхокардиография (ЭХО);
- стресс-ЭХО;
- ЭКГ;
- ферменты;
- рентгенография;
- велоэргометрия (ВЭМ);
- коронароангиография;
- холтер;
- биохимия крови.

Разновидности воздействий:

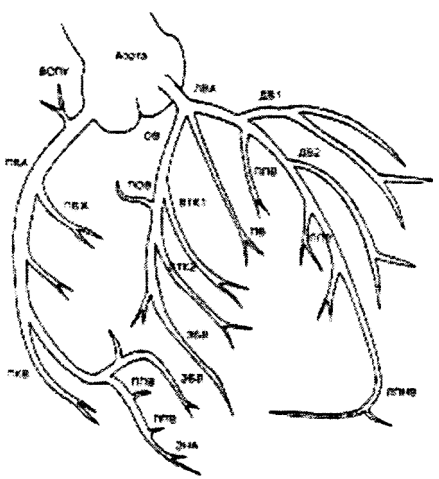


Рисунок 2. Топология коронарных артерий

- чрезкожное коронарное вмешательство (ЧКВ);
- тромболитическая терапия (ТЛТ).

Результаты большинства исследований фиксируются в бумажной истории болезни от руки на бланках стандартизованной формы. Некоторые из бланков содержат более 30 полей для заполнения.

Результаты отдельных исследований и воздействий попадают в историю болезни в виде распечаток, получаемых с медицинского оборудования. Часть

аппаратуры клиники снабжена интерфейсом, позволяющим получать данные в формате DiCOM. На рис.1 приведен пример отчета о посегментных нарушениях локальной скоратимости сердца, формируемый аппаратом ультразвуковой диагностики.

Результаты таких воздействий как коронарография (диагностика точного места сужения с помощью рентгеновских лучей с предварительным введением контрастного вещества в коронарную артерию через катетер) и ЧКВ фиксируются врачом в виде текстового описания либо с помощью отметок от руки на специально подготовленных бумажных шаблонах (рис.2), задающих топологию коронарных сосудов.

4. Предложенная технология

Результаты анализа бизнес-процессов, тесное взаимодействие с сотрудниками клиники привели к решению о разделении пользовательских интерфейсов разрабатываемой системы на две группы:

1. Экранные формы, соответствующие исследованиям, результаты которых фиксируются путем заполнения бумажных бланков.
2. Нестандартные пользовательские интерфейсы отдельных исследований и воздействий, плохо представимые в виде диалоговых окон и требующих отдельной проработки.

4.1. Библиотека FastUI

Для исследований 1-й группы, ввиду взаимно однозначного соответствия свойств классов модели предметной области (выбранный шаблон проектирования промежуточного слоя) и полей визуализирующих их форм, было принято решений о создании на базе .NET Windows Forms 2.0 библиотеки, получившей название *FastUI*, и обладающей следующими возможностями:

- простой декларативный язык описания экранных форм (включая переходы между формами);
- визуализация форм;
- автоматизированная генерация форм по классам промежуточного слоя, наделенных дополнительной информацией;
- автоматизированная синхронизация форм с классами промежуточного слоя в случае внесения изменений в последние.

Язык, построенный на базе XML, (получивший название *FastUI Language, FUIL*) позволяет наглядно описать порядок следования элементов управления на форме, их связь с данными, а также задать

```
<?xml version="1.0" encoding="utf-8"?>
<Form name="addPatient" title="Новый пациент" readOnly="false" class="Patient"
      titleProp="Passport.LastName">
  <TabControl>
    <Tabs>
      <Tab caption="Паспортная часть">
        <TextBox prop="Passport.LastName" caption="Фамилия" required="true" />
        <TextBox prop="Passport.FirstName" caption="Имя" />
        <TextBox prop="Passport.Patronymic" caption="Отчество" />
      </Tab>
      <Tab caption="Госпитализации">
        <DataGrid prop="Hospitalizations" readOnly="false">
          <ChildForms>
            <ChildForm form="hospitalization" class="Hospitalization"/>
          </ChildForms>
          <Columns>
            <Column header="Время госпитализации" prop="HospitalizationTime"  />
            <Column header="Номер ИБ" prop="Number" />
          </Columns>
        </DataGrid>
      </Tab>
    </Tabs>
  </TabControl>
</Form>
```

Рисунок 3. Описание формы на языке FUIL

навигацию по формам. Пример формы, описанной на FUIL, приведен на рис.3.

Визуализатор форм обеспечивает интерпретацию FUIL-файлов, реакцию на события навигации по формам. Для отображения форм была использована концепция многодокументного интерфейса (Multiple Document Interface, MDI), с дополнительными гибкими возможностями изменения местоположения окон (т.н. docking) (рис.4), которые были реализованы при помощи shareware библиотеки DockPanel Suite. Визуализатор форм обеспечивает интерпретацию элементов языка FUIL и позволяет включить в приложение через API произвольные фрагменты пользовательского интерфейса, заданные в виде Windows Forms 2.0 User Control. Кроме этого визуализатор берет на себя функции отслеживания изменения данных формы (т.н. проблемы dirty данных), необходимости проведения каскадных операций (сохранения и удаления) и способен взаимодействовать практически с любым поставщиком объектов промежуточного слоя, не накладывая каких-либо ограничений на последний.

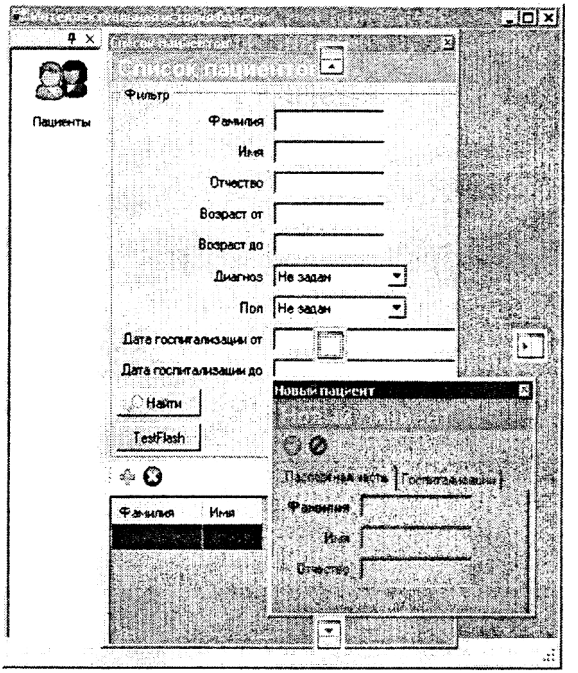


Рисунок 4. Внешний вид MDI-форм с docking

В силу взаимно однозначного соответствия экранных форм и классов предметной области разработка пользовательского интерфейса и классов промежуточного слоя может производиться параллельно. FastUI обладает возможностью автоматизированного создания *прототипа пользовательского интерфейса*. Для этого достаточно лишь наделить свойства классов предметной области мета-информацией об отображаемых названиях полей формы, единицах измерения, элементах управления, наилучшим образом подходящих для поля и другой информацией. Свойства классов, являющиеся ссылками и коллекциями могут быть «помечены» дополнительной информацией, указывающей о возможности и режимах навигации и об уместных классах-наследниках в случае полиморфных ссылок или коллекций.

Возможность добавления мета-информации к классам реализована с помощью атрибутов (attributes) и отражения (reflection) типов .NET. Следует отметить, что такой способ задания мета-информации обеспечивает ее наглядность, доступность и «близость» к описываемым ею данным. Пример свойства класса, «размеченного» атрибутами библиотеки FastUI приведен на рис.5.

```
/// <summary>
/// Фермент тропонин I
/// </summary>
[PropInfo(Name="Тропонин I", Unit="пр./моль", Required = true )]
public Double Troponin
{
    get { return troponin; }
    set { troponin = value; }
}
```

Рисунок 5. Разметка свойства класса атрибутами FastUI

Возможность *синхронизации* обеспечивает обновление файлов пользовательского интерфейса при изменениях классов промежуточного слоя. При этом изменения, внесенные дизайнером в описание пользовательского интерфейса, не будут перезатерты.

Следует отметить, что приложение, использующее FastUI, по сути соответствует т.н. *архитектуре, управляемой моделью (Model-Driven Architecture, MDA)*[3]. MDA предполагает разработку системы в 2 этапа:

- всеобъемлющее детализированное проектирование (моделирование) логики работы;
- реализация функциональности на базе построенных моделей.

При этом модели (все или значительная часть), возникающие на этапе проектирования являются платформонезависимыми. А реализация производится для заданной целевой программной платформы. MDA обладает следующими преимуществами.

- Простота переноса системы на новую платформу (как FUIL, так и специфицированная в FastUI мета-информация о промежуточном слое платформонезависимы).
- Экономия ресурсов при реализации приложения для нескольких программных платформ одновременно.
- Возможность автоматизации программирования. Наличие подробной модели делает возможным автоматизированное создание типовых частей приложения, разработка которых поддается автоматизации.

4.2. Проектирование и реализация нестандартных пользовательских интерфейсов

Наиболее значимыми, с точки зрения информативности, о состоянии пациента и возможности проведения ангиопластики и одновременно наиболее сложными, с точки зрения реализации взаимодействия пользователя с системой, разновидностями событий в истории болезни института неотложной кардиологии стали коронарография и ЧКВ (проводимое под контролем коронарографии).

Данные обоих исследований «располагаются» на частях коронарных сосудов.

Коронарография задает информацию о степени стеноза и толщине сосудов, возможных перетоках крови между сосудами, способных в некоторых случаях как усилить, так и уменьшить негативное влияние стеноза на сердечную мышцу, шунтах, с одной стороны, обеспечивающих дополнительный кровоток, с другой, также являющихся потенциальным местом возникновения «бляшек». Задача усложнилась тем, что топология сосудов каждого человека индивидуальна и эти особенности также необходимо учесть при проведении анализа результата исследований.

ЧКВ вдобавок к коронарографии определяет разновидности и свойства следующих воздействий, примененных к сосудам с целью уменьшения стеноза:

- *Балонная ангиопластика* – уменьшение «бляшки» путем раздувания баллона, введнного в место сужения;
- *Проводниковая реканализация* – разрушение стеноза путем механического воздействия проводником;
- *Стентирование* – уменьшение «бляшки» и препятствие ее повторному возникновению за счет установки в месте сужения специального каркаса-стента.

Вся перечисленная информация, является важной для приятия решений и должна как вводится, так и визуализироваться средствами пользовательского интерфейса указанных исследований.

При разработке пользовательского интерфейса коронарографии и ЧКВ были использованы некоторые идеи *целеориентированного проектирования (Goal-Directed Design)*, предложенного Аланом Купером [1]. Подход подразумевает этап *проектирования взаимодействия пользователя с ситемой*, предшевствующего реализации пользовательского интерфейса. Ключевой задачей проектирования является создание точного описания пользователей программного продукта и их *целей (Goals)*. Описание пользователей создается при помощи моделирования поведения вымышленных пользователей, называемых *персонажами (Persons)*.

Совместно со специалистами клиники, проводящими коронарографию и ЧКВ, были составлены подробные описания их целей по отношению к системе, описаны сценарии «повседневного» проведения данных исследований и внесения информации о них в систему и выделены редкие «нештатные» сценарии.

Исходя из полученной информации, все функциональные возможности по заполнению данных коронарографии и ЧКВ были разделены на группы, в соответствие с частотой использования функций. Пользовательский интерфейс организован на основе идеи упрощения и облегчения выполнения пользователем наиболее частых

функций с сохранением возможности выполнения редких действий и доступности и наглядности важной и необходимой информации.

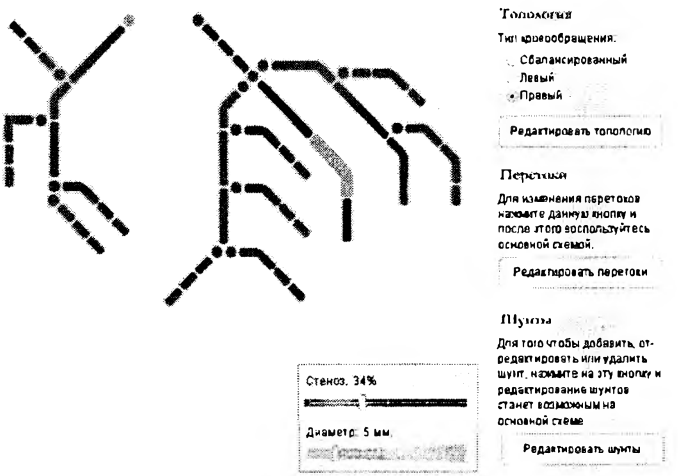


Рисунок 6. Скриншот фрагмента пользовательского интерфейса коронарографии

Скриншот пользовательского интерфейса коронарографии приведен на рис.6. Степень стеноза коронарных сосудов визуализируется цветом. С правой стороны перечислены режимы редактирования данных исследования. Реализация проведена на Adobe Flash 8.0. Следует отметить, что этот нестандартный пользовательский интерфейс был без проблем интегрирован в общий интерфейс системы, построенный на базе FastUI, за счет возможности расширения FastUI через API.

5. Заключение

В результате проведенной работы была предложена программная технология построения пользовательского интерфейса предметно-ориентированных информационных систем. Технология сочетает в себе функции моделирования пользовательского интерфейса, последующего автоматизированного построения прототипов пользовательского интерфейса с использованием целеориентированного проектирования для создания специализированных пользовательских интерфейсов.

Технология реализована на платформе .NET 2.0 и апробирована в проекте «Интеллектуальная история болезни клиники неотложной кардиологии».

Применение технологии позволит разработчикам информационных систем получать прототипы пользовательского

интерфейса на ранних этапах разработки, сократить общие трудозатраты на создание системы, повысить качество пользовательских интерфейсов.

В дальнейшие планы по развитию технологии входит ее перенос на платформу Windows Presentation Foundation, использование XAML в качестве языка описания форм, улучшение возможностей стилизации пользовательского интерфейса. Также планируется формализовать знания из области эргономики и дизайна и принимать их во внимание при автоматизированном создании прототипов пользовательского интерфейса.

Литература

1. А.Купер «Психбольница в руках пациентов» СПб: Символ-Плюс, 2004
2. Д.Норман «Дизайн привычных вещей» М.: Издательский дом «Вильямсч», 2006
3. Е. Марков «Архитектура, управляемая моделью», 2005 http://www.citcity.ru/integration/articles/11353/
4. Библиотека DockPanel Suite http://sourceforge.net/projects/dockpanelsuite/

Раздел II
Интеллектуальный анализ данных (Data Mining)

Е.А.Попова

Методы построения ансамблей деревьев решений[1]

Статистическая обработка информации используется при решении многих прикладных задач распознавания образов, классификации и выявление связей в данных. За последние несколько лет было разработано множество алгоритмов, называемых методами data mining, но все же ряд задач требует более устойчивых, быстрых и точных алгоритмов обработки данных. Одним из широко распространенных методов data mining является метод деревьев решений и ансамблей деревьев решений. В данной работе будет проведен теоретический анализ алгоритмов построения ансамблей деревьев решений, их применения к решению прикладных задач и выделены основные проблемы существующих на данный момент алгоритмов.

Пусть нам задано некоторое обучающее множество T, содержащее образы, каждый из которых характеризуется M признаками (атрибутами). Обозначим через $C_1,C_2,...C_l$ метки классов, тогда вектор признаков можно записать следующим образом

$$\mathbf{X}^p = \{x_1^p,...x_i^p,...x_M^p \mid C_j^p\}$$

где p- номер примера, j - номер класса, x_i^p i -ый признак p-го примера. Задача классификации состоит в построении функции (классификатора), разделяющей образы, принадлежащие различным классам. Классификатор строится на основе обучающего множества примеров T. При аккредитации классификатору предъявляется образ с таким же набором признаков, как и в обучающем множестве, но с неизвестной принадлежностью к какому-либо классу. Классификатор должен отнести указанный вектор к одному из имеющихся классов. Оценка качества классификатора определяется ошибкой классификацией или риском.

В качестве классификатора может быть использовано дерево решений. Дерево представляет собой связный неориентированный граф

[1] Работа выполнена при поддержке гранта РФФИ 05-07-90238.

без циклов. Дерево не имеет кратных ребер и петель. Обобщением понятия дерева является понятие леса. Лес – это граф без циклов (не обязательно связный).

Дерево решений — это такое дерево, каждая внутренняя вершина которого представляет элементарное решение. В зависимости от решения, принятого в таком узле, управление передается левому или правому поддереву. Результатом является решение, соответствующее листу. Алгоритмы построения одного дерева решений приведены в [1]. Ансамблем деревьев решений называется классификатор, который является комитетом деревьев решений.

Пусть ансамбль состоит из k деревьев решений. Для построения каждого дерева формируется случайный вектор $\Theta = \{\alpha_1,...,\alpha_M\}$, где $\alpha_j = \begin{cases} 1, j \in D_k \\ 0, j \notin D_k \end{cases}$,

где D_k - множество признаков, участвующих в построении дерева k. Таким образом определяется подмножество значений признаков из исходного множества T, на основе которого строится дерево решений.

Для каждого дерева j формируется случайный вектор Θ_j, не зависящий от предыдущих случайных векторов $\Theta_1,...,\Theta_{j-1}$, но с тем же распределением. Дерево решений строится по одному из существующих алгоритмов [1]-[3], формируя классификатор $h(\mathbf{X},\Theta_k)$, где $\mathbf{X}$ – входной вектор для задачи классификации. После построения каждого дерева в ансамбле подсчитывается общая ошибка классификации методом кроссвалидации или на основе проверочного множества, выделенного из обучающего множества.

Ансамбль деревьев решений - это классификатор, состоящий из k деревьев-классификаторов $\{h(\mathbf{X},\Theta_k), k = 1...\}$, где $\{\Theta_k\}$ - независимые случайные вектора с одинаковым распределением; и каждое дерево в ансамбле участвует в голосовании за класс при входном векторе $\mathbf{X}$.

Можно выделить три основных метода построения ансамблей деревьев решений для задач классификации данных: Bagging [1], Random Forest [2] и Boosting [3]. Существует множество модификаций этих методов, каждая из которых в какой-то степени улучшает созданные методы построения.

Одно из основных достоинств деревьев решений заключается в их возможности обрабатывать данные разного типа, а также данные, имеющие нетривиальные связи. Например, алгоритм CART (classification and regression trees) [4] основан на вероятностном структурировании данных. Очередной признак для построения дерева

выбирается, основываясь на критерии минимальной энтропии. Такой принцип моделирования оказывается широко распространенным при построении моделей предсказания. При этом дерево решений дает грубую, кусочно-постоянную аппроксимацию данных, устойчивую к шуму. В последнее время деревья решений находят свое применение не только в задачах прогнозирования, но также и при обработке сигналов, распознавании изображения и текста, разработке систем безопасности, моделировании лекарственных препаратов и т.д. Опишем алгоритмы построения ансамблей деревьев решений, их достоинства и недостатки при решении определенных задач.

Экспериментальные тесты показали, что построение одного дерева решений и работа с ним оказывается не совсем эффективной. Хранить все дерево в памяти, осуществлять по нему поиск требует большого времени и вычислительных ресурсов, а полученная ошибка классификации не является приемлемой для большинства класса решаемых задач. При построении множества классификаторов, каждый из которых не зависит от другого, ошибка классификации уменьшается, а также классификация осуществляется путем «голосования» комитета, каждый член которого является деревом решений.

Обобщая все существующие алгоритмы формирования ансамбля деревьев решений, неформально процесс построения можно разделить на два этапа: это обучения каждого дерева в ансамбле и объединение всех деревьев в единый ансамбль.

Выделяют две концепции обучения ансамбля. Первая, когда все члены ансамбля строятся последовательно, вторая, когда они строятся параллельно. К первому классу относится алгоритм Adaboost [5], который строит последовательность ансамблей деревьев решений, где обучающее множество каждого из них формируется на основе примеров, которые были неверно классифицированы предыдущем ансамблем. К этому классу также можно отнести алгоритмы Arc-x4 [6], fBoost [7], MiniBoost [8], MultiBoost [9], и т.д. К классу параллельно обучающихся деревьев относится Bagging [1], который формирует множества обучающих векторов для каждого дерева ансамбля путем селекции с замещением и потом независимо строит каждое дерево. Другими алгоритмами, относящимися к данному подходу, являются SEQUEL [10], Wagging [11], p-Bagging[12], GASEN [11], random subspaces [13], random forests [14], and randomized C4.5 [15] и т.д. При формировании обучающих множеств, для каждого дерева решающую роль играет исходная размерность задачи, которая и диктует выбор соответствующего метода.

В случае использования всего исходного множества примеров для построения каждого дерева решений, полученная ошибка будет лучше, нежели полученная при ограниченном подмножестве данных. С другой стороны, разбиение на подмножества исходного пространства,

позволит четко разбить задачу на независимые подзадачи, минимизировать память для хранения обучающей выборки для построения каждого дерева. Несколько подходов по построению дерева будут обозначены ниже. В [14],[1186] описаны алгоритмы, где множество признаков для построения каждого дерева решений формируется случайным выбором их из исходного множества. Данный метод находит широкое применение в классе таких задач как последовательная классификация и автоматическое выведение оптимальных стратегий принятия решений из экспериментальных данных, предсказание поведения системы на основе анализа различных сигналов, распознавание речи, защита информации.

В [167] разработан метод образования ансамблей деревьев решений, который использует все множество доступных примеров для построения каждого дерева. Каждое дерево строится путем итеративного процесса, который заключается в следующем: все обучающее множество разбивается на 2 подмножества. На каждой итерации, одно подмножество используется для построения дерева решения, начиная с дерева решения, построенного на предыдущей итерации. Это большое дерево дальше отсекается, используя другое подмножество. Роли каждого подмножества меняются на каждой итерации. В итоге процесс сходится к результирующему дереву решения, которое является устойчивым из-за комбинированного построения и отсечения ветвей. Начальное разбиение выбирается случайно. Этот метод эффективно применяется при online обучении и моделировании динамически меняющихся потоков данных в таких задачах как: обнаружение поддельных кредитных карт или использование чужих, поиск и вставка данных в XML документах, web навигация и т.д.

Экспериментально на тестовых базах данный показано в [15], что оптимальным решением для решения задач со сравнительно одинаковым размером пространства признаков и примеров, содержащих смешанные типы признаков, будет формирование случайных подмножеств для построения каждого дерева, а в случае баз данных огромной размерности случайный выбор из подмножеств множества. Выбор того или иного распределения обучающих множеств по деревьям сильно зависит от природы данных: размерности, количества категориальных и числовых признаков, пропущенных значений и т.д. Работа [18] посвящена изучению сравнительных тестов алгоритмов на обучающих данных сложной структуры. В результате было составлено несколько устойчивых, непарметризованных тестов для сравнения классификаторов.

С задачей формирования обучающих множеств тесно связана задача критерия выбора следующего признака для разбиения при построении дерева решения. Этой проблеме посвящено достаточно

много работ, так как она является важным параметром, влияющим на скорость работы алгоритма и его общую ошибку. Можно выделить три принципа выбора признака для разбиения: вероятностный, случайный и смешанный. Экспериментально результаты сравнения точности классификации байесовских деревьев решений и случайных деревьев решений описаны в работе [19]. Байесовские ДР имеют меньшую вероятность точности классификации, но сама классификация неверна в большем числе случаев, чем в случайных ансамблях.

Наиболее распространенными вероятностными методами [20] являются критерий Gini, который используется в C4.5 и CART, towing splitting rule для двуклассовых задач, Boolean splitting, impurity function и др. Вычислительная сложность вероятностного подхода зависит от размерности обучающего множества и количества признаков в этом множестве. Следует также отметить, что в случае числовых значений признаков, после выбора нужного признака для разбиения, выбирается пороговое значения для разбиения, что требует также немалых вычислительных затрат. Поэтому для обработки больших объемов данных вероятностный подход не является оптимальным. Другим подходом, будем случайный выбор признака на каждом шаге построения узла дерева, и случайный выбор порогового значения для разбиения. В работе [21] показана эффективность данного подхода для решения задач последовательной классификации и автоматического определения оптимальных стратегий выбора решения. Существуют смешанный алгоритмы, которые пытаются унифицировать критерий выбора для различного класса задач. В таких алгоритмах есть несколько стратегий. Первая, это случайное формирование множества признаков исходных данных, а потом вероятностный выбор одного из них для разбиения. Вторая, это случайный выбор признака разбиения из изменяющейся на каждом шаге выборке признаков[24]. Третий, это эволюционные алгоритмы выбора критерия [19], [22], [23], которые могут меняться от узла к узлу, например, с помощью генетического алгоритма [24]. Эффективность каждого из этих алгоритмов показана для решения задач биоинформатики: экспрессии генов, microarray data analysis, обнаружении подельных лекарств и создании новых.

Для всех обучающих множеств, которые имеют разные характеристики, одного наиболее эффективного критерия не существует. Можно выделить некоторые критерии разбиения, основанные на оценках размерности исходного пространства. Например, в работе [17] рассмотрен метод выбора следующего признака для разбиения при построении каждого дерева в ансамбле на основе гистограмм. Для обучающих множеств большой размерности, они уменьшают время, требующееся для анализа всех примеров множества при выборе признака разбиения, и требуют меньше вычислительных ресурс для их хранения. Обычно, для каждого признака строится

гистограмма, и подсчитанные границы используются как потенциальные точки для разбиения при построении дерева. Такой алгоритм находит применение при обработке больших объемов информации при решении задач обработки изображения снимков со спутника и распознавания рукописного текста. Если же количество признаков невелико по сравнению с количеством примеров, то можно увеличить их количество путем добавление специальных признаков, которые являлись бы случайной линейной комбинацией изначально заданных [13]. Или же полностью сменить признаки обучающего множества на их линейные комбинации, такие деревья решений носят названия oblique или «скошенные деревья решений»[21]. Они часто применяются при анализе последовательностей структур (ДНК, белков, протеинов и т.д.).

Если рассмотреть реальные экспериментальные данные, то многие из них сильно зашумлены или имеют пропущенные значения. Если для каждого дерева выбрать обучающее множество и в процессе обучения не менять его, то некоторые деревья будут строиться на некорректных данных (шуме). Избежать этого можно путем смены примеров обучающего множества при каждом выборе признака.

Следующей задачей в обучении ансамбля является задача критерия остановки построения деревьев. Например, в Random Forest Лео Бреймана теоретически доказано, что общая ошибка классификации зависит от индивидуальной силы каждого дерева в лесу, а также корреляцией деревьев в лесу.

Пусть дан ансамбль деревьев решений $h_1(\mathbf{x}), h_2(\mathbf{x}),..., h_k(\mathbf{x}),$ и обучающее множество, сформированное случайно из распределения Y на **X**, определяющие граничную функцию как

$$mg(\mathbf{X},Y) = av_k I(h_k(\mathbf{X}) = Y) - \max_{j \neq Y} av_k I(h_k(\mathbf{X}) = j),$$

где $I(\bullet)$ - функция индикатор, **X** – пространство признаков, Y – пространство меток класса. Граница измеряет расширение (расстояние мера степень предел), до которого среднее число голосов на **X**,Y правильного класса превосходит среднее число голосов для любого другого класса. Чем больше граница, тем более вероятна классификация. Общая ошибка классификации определяется как

$$PE^* = P_{\mathbf{X},Y}(mg(\mathbf{X},Y) < 0),$$

где индекс $\mathbf{X},Y$ показывает вероятность на всем пространстве $\mathbf{X},Y$.

Граничная функция для random forest определяется как

$$mr(\mathbf{X},Y) = P_\Theta(h(\mathbf{X},\Theta) = Y) - \max_{j \neq Y} P_\Theta(h(\mathbf{X},\Theta) = j),$$

а сила множества классификаторов, как

$$s = E_{\mathbf{X},Y} mr(\mathbf{X},Y),$$

Считая, что $s \geq 0$, неравенство Чебышева дает

$PE^* \leq \text{var}(mr)/s^2$ или общую ошибку можно ограничить сверху следующим образом

$$PE^* \leq \bar{\rho}(1-s^2)/s^2$$

Изначально, одна треть примеров используется для вычисления s (сила дерева), $\bar{\rho}$ (корреляция), общей ошибки классификации. Это множество примеров не участвует в построении деревьев решений и называется out of bag множество. Таким образом, на каждом шаге добавления дерева может быть вычислена оценка общей ошибки классификации ансамбля и принято решение об остановке построения. Одним из параметров при построении Random Forest является первоначальное количество признаков для построения каждого дерева.

Но тут возникает другой немаловажный вопрос. Следует ли хранить все деревья ансамбля, или же часть из них являются не решающими в голосовании. Среди существующих подходов можно выделить несколько: использовать весовые коэффициенты для деревьев ансамбля, основываясь на априорных оценках общей ошибки классификации, эволюционные алгоритмы отбора деревьев в итоговый ансамбль, случайный выбор деревьев. В работе [26] рассматривается мера сравнения ансамблей для выявления лучшего при случайном их формировании. В работе предложен алгоритм GASEN-b, который основан на отборе нескольких деревьев в результирующий ансамбль путем генетического алгоритма. К каждому классификатору ансамбля приписываются случайные веса, далее определяется фитнес функция ГА.

После построения окончательного ансамбля, определяется процесс голосования деревьев. В Random Forest, окончательный класс определяется большинством «голосов». Это не самый эффективный критерий голосования. В работе [27] демонстрируется, что увеличить точность предсказания random forest возможно на некоторых множествах путем замещения функции общей оценки каждого дерева. Предложена функция Dynamic Integration для вычисления класса после голосования всех деревьев. Она основана на определении весовых коэффициентов, которые в свою очередь зависят от меры близости примеров из обучающего множества друг к другу.

Пусть w – весовой коэффициент для дерева i, тогда его вес определяется формулой

$$w_i(x) = \frac{\sum_{j=1}^{k} I(x_j \in OOB_i) \cdot \sigma(x, x_j) \cdot mr_i(x_j)}{\sum_{j=1}^{k} I(x_j \in OOB_i) \cdot \sigma(x, x_j)},$$ где

OOB_i — это out-of-bag множество RF, $mr_i(x_j)$ - граничная функция, которая =1, если класс определен деревом i верно, и -1 иначе; $\sigma(x, x_j)$ - мера близости примеров из обучающего множества дерева i. Экспериментально на стандартных тестовых базах данных было показано, что dynamic integration уменьшает ошибку классификации на 50% баз данных.

Большинство разрабатываемых алгоритмов сравниваются уже с существующими Bagging, Boosting, как универсальными алгоритмами построения классификаторов, также и с широко распространенным Random Forest Лео Бреймана. В [28],[29] представлено сравнение таких алгоритмов построения ансамблей деревьев решений, как: boosting, random forest, bagging, random subspaces, random forests, randomized C4.5. Сравнение алгоритмов с помощью кросс-валидации показало, что статистически bagging был более точным только на 16% баз данных. И наоборот, проверяя алгоритмы параллельно на нескольких множеств данных результаты, показали, что boosting, random forests, randomized trees статистически лучше, чем bagging. На основе полученных результатов было выявлен один из решающих параметров ансамбля – количество классификаторов. В Автоматическое определение оптимального числа деревьев в лесу [30], путем введения двух множеств делает задачу непараметризованной, с помощью первого множества отсекается часть деревьев, с помощью другого оценивается общая ошибка классификации.

В [31] была попытка объединить несколько ансамблей, построенных разными алгоритмами (Boosting, Bagging, Random Forest и т.д.) в один решающий ансамбль путем введения ортогональных деревьев решений. Метод основывается на анализе принципиальных компонент функционального пространства, построении таких деревьев на основе преобразования Фурье деревьев решений и самоанализе построенного ансамбля на основе этого преобразования. Вычислительные затраты такого подхода значительно отличаются от обычного алгоритма, но уменьшает ошибку классификации. Одно из последних исследований по сравнению восьми алгоритмов построения ансамблей на разных базах данных показало, что никаких особо

выраженных отличий этих 8-ми нет. Отсюда следует, что спецификой задачи определяется эффективность выбранного алгоритма. Универсальные методы существуют, но каждая задача требует детального рассмотрения.

В результате данного исследования существующих подходов в построении ансамбля деревьев решений, можно сказать, что на данный момент открытии проблемами остаются минимизация ошибки и создание оптимальных по вычислительным затратам алгоритмов классификации сильно зашумленных данных больших размерностей, и уменьшения ошибки классификации при распределенном обучении ансамбля. Также не решенной остается вопрос влияния отсеченных в ансамбле ветвей на ошибку и скорость классификации.

Список литературы

1. L. Breiman, "Bagging predictors," Machine Learning, vol. 24, pp. 123–140
2. L. Breiman Random Forest – Random Features. Technical Report 567 September 1999
3. Yoav Freund and Robert E. Shapire. Experiments with a new boosting algorithm. In Lorenza Saitta, editor, *Machine Learning: Proceedings of the Thirteenth Inter-national Conference (ICML'96)*. Morgan Kaufmann, 1996.
4. Breiman, L., Friedman, J.H., Olshen, R.A., and Stone, C.J.: *Classification and Regression Trees*. Chapman & Hall, New York, (1984).
5. Freund Y., Schapire R. E.: A decision-theoretic generalization of on-line learning and an application to boosting. In: Proceedings of the 2nd European Conference on Computational Learning Theory (1995) 23-37.
6. Breiman L.: Arcing classifiers. Annals of Statistics 26 (1998) 801-849.
7. Harries M.: Boosting a strong learner: evidence against the minimum margin. In: Proceedings of the 16th International Conference on Machine Learning (1999) 171-179.
8. Quinlan J. R.: Miniboosting decision trees.http://www.cse.unsw.edu.au/~quinlan/miniboost.ps.
9. Webb G. I.: MultiBoosting: a technique for combining boosting and wagging. Machine Learning 40 (2000) 159-196.
10. Asker L., Maclin R.: Ensembles as a sequence of classifiers. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence (1997) 860-865.
11. Bauer E., Kohavi R.: An empirical comparison of voting classification algorithms: bagging, boosting, and variants. Machine Learning 36 (1999) 105-139.

12. Zhou Z.-H., Wu J., Tang W.: Ensembling neural networks: many could be better than all. Artificial Intelligence 137 (2002) 239-263.
13. T. Ho, "The random subspace method for constructing decision forests," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 832–844, 1998.
14. L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.
15. T. Dietterich, "An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization," *Machine Learning*, vol. 40, no. 2, pp. 139–157, 2000.
16. Martinez-Munoz, G. Suarez, A. Comput. Sci. Dept., Univ. Autonoma de Madrid, Spain. Using all data to generate decision tree ensembles, Systems, Man and Cybernetics, Part C, IEEE Transactions on Nov. 2004
17. Kamath, C., Cantú-Paz, E. and Littau, D. (2002). Approximate splitting for ensembles of trees using histograms. In Second SIAM International Conference on Data Mining (SDM-2002).
18. J. Demsar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, pp. 1–30, 2006.
19. Vitaly Schetinin, Jonathan E. Fieldsend, Derek Partridge, Wojtek J. Krzanowski, Richard M. Everson, Trevor C. Bailey, and Adolfo Hernandez, Comparison of the Bayesian and Randomised Decision Tree Ensembles within an Uncertainty Envelope Technique, Journal of Mathematical Modelling and Algorithms, 31.10.2004.
20. Markus Breitenbach, Rodney Nielsen, and G.Z. Grudic. Probabilistic random forests: Predicting data point specific misclassification probabilities. Department of Computer Science CU-CS-954-03, University of Colorado, 2003.
21. Creating ensembles of oblique decision trees with evolutionary algorithms and sampling, US Patent Issued on June 13, 2006.
22. Cantú-Paz, E and Kamath, C. (2003). Inducing oblique decision trees with evolutionary algorithms. IEEE Transactions on Evolutionary Computation. 7(1), 54-68.
23. Louis Wehenkel, Damien Ernst, Pierre Geurts. Ensembles of extremely randomized trees and some generic applications, Department of Electrical Engineering and Computer Science University of Li` ege - Sart-Tilman B28 - B-4000 Li` ege, 2006.
24. Creating ensembles of oblique decision trees with evolutionary algorithms and sampling, US Patent Issued on June 13, 2006
25. Hongshik Anh, Hojin Moon, Melissa J. Fazzari. Classification by Ensembles from Partitions, Division of Biometry and Risk Assessment, 2006.

26. R. E. Banfield, L. O. Hall, K. W. Bowyer, and W. P. Kegelmeyer, "A new ensemble diversity measure applied to thinning ensembles," in Fifth International Workshop on Multiple Classifier Systems, 2003, pp. 306–316
27. Alexey Tsymble, Mykola Pechenizkiy, Padraig Cunningham Dynamic Intergration with Random Forest, 2006.
28. Robert E. Banfield and Lawrence O. Hall, Kevin W. Bowyer, W. Philip Kegelmeyer. A Comparison of Decision Tree Ensemble Creation Techniques, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2006.
29. Corinne Dahinden, Seminar for Statistics, ETH Zurich, Classification with Tree-Based Ensembles Applied to the WCCI 2006 Performance Prediction Challenge Datasets, 2006 International Joint Conference on Neural Networks Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada July 16-21, 2006.
30. Praveen Boince, Alessandro De Angelis, and Gian Luca Foresti. Meta Random Forests, INTERNATIONAL JOURNAL OF COMPUTATIONAL INTELLIGENCE VOLUME 2 NUMBER 3 2005 ISSN 1304-4508.
31. Hillol Kargupta Byung-Hoon Park Haimonti Dutta, Orthogonal Decision Trees, IEEE Transactions on knowledge and data engineering, August 2006 (Vol. 18, No. 8), pp. 1028-1042.

Глазкова В. В., Петровский М.И.

Дообучаемый метод классификации многотемных документов для анализа и фильтрации интернет информации

Работа поддерживается грантом президента РФ МК-2111.2005.9 и грантами РФФИ 06-01-00691, 05-01-00744.

Введение

Задача классификации многотемных документов (multi-label классификации) является основной подзадачей в проблеме фильтрации Web-трафика на основе содержимого документов. Исследование существующих методов классификации многотемных документов применительно к задаче фильтрации Web-трафика показало, что эти методы недостаточно эффективны по времени, памяти и не имеют возможности дообучения с использованием только новых обучающих документов, которая очень важна для рассматриваемой задачи.

Elisseff и Weston [1,2] предложили ядерный метод multi-label классификации, основанный на методе ранжирования RankSVM с пороговой функцией в пространстве признаков. Идея метода RankSVM состоит в том, чтобы по обучающему набору найти функцию ранжирования, которая минимизирует обобщённую ошибку (основанную на Ranking Loss) и даёт наибольшую граничную полосу между классами для multi-label модели. Однако метод RankSVM имеет существенный недостаток – этот алгоритм требует в худшем случае количество памяти, пропорциональное mq^2 (m – количество обучающих документов, q – количество классов). Поэтому, учитывая специфику задачи фильтрации Web-трафика (большое количество классов и большой размер обучающего набора), метод [1,2] неприменим в качестве алгоритма multi-label классификации для рассматриваемой задачи.

Ещё два метода multi-label классификации AdaBoost.MH [3,4] и ADTBoost.MH [5] также основаны на методах ранжирования, но с нулевой пороговой функцией. Основная идея метода AdaBoost.MH состоит в том, чтобы найти точное правило классификации, комбинируя много гипотез, каждая из которых только умеренно точна. Метод AdaBoost.MH поддерживает набор весов, как для обучающих примеров, так и для их тем, причём обучающие примеры и соответствующие им темы, которые являются более трудными (простыми) для правильного предсказания, получают более высокие (низкие) веса. Однако для

достижения приемлемой точности этот метод требует большое число шагов на этапе обучения, а следовательно, очень долгое время обучения (особенно для задач текстовой классификации). Кроме того, этот метод не дообучаемый - он рассчитан на то, что обучающий набор даётся заранее целиком.

Метод ADTBoost.MH [5] – это расширение метода ADTBoost [6] для обработки multi-label данных. В этом методе число правил классификации, рассматриваемых как multi-label дерево решений, соответствует числу гипотез алгоритма AdaBoost.MH. Для алгоритма ADTBoost.MH время обучения и затраты памяти резко возрастают с увеличением размерности пространства признаков. Эти затраты становятся неприемлемыми для задач текстовой классификации, для которых размерность пространства признаков порядка количества слов в словаре.

Метод multi-label классификации [7] с использованием вероятностной модели смешивания, обученной с помощью метода максимизации математического ожидания (Expectation-Maximization – EM), основан на байесовском подходе для multi-label задачи. В этом методе определяется порождающая вероятностная модель, отражающая природу многотемного документа. Этот метод обеспечивает прямое определение наиболее вероятного набора тем, избегая проблем выбора пороговой функции, свойственных методам multi-label классификации, основанным на методах ранжирования. Однако этот метод недостаточно эффективен, когда количество тем q велико (так как исследование всех наборов требует 2^q вычислительных шагов), поэтому в этом случае необходимы приближённые схемы исследования пространства наборов тем. Но использование таких приближённых схем приводит к ухудшению точности метода.

В методе ML-kNN [8] для определения набора тем, релевантных для тестового образца, используется принцип максимизации апостериорных вероятностей принадлежности образца к категории, то есть этот метод, в отличие от предыдущего, не прибегает к перебору всех возможных наборов тем. Однако этот метод имеет существенные недостатки, свойственные методам, основанным на методе k ближайших соседей, - это большие вычислительные затраты на этапе классификации и большой объём памяти, необходимый для хранения и работы сразу со всем набором обучающих данных.

Метод ML-SVM [9] основан на подходе «каждый-против-остальных», согласно которому multi-label проблема декомпозируется в набор независимых бинарных проблем (по одной на каждую из q меток). В бинарной проблеме для метки r все примеры, помеченные этой меткой, считаются положительными, а все остальные примеры считаются отрицательными. Далее к каждой из q полученных бинарных

проблем применяется бинарный алгоритм обучения. В результате получаются q гипотез, которые должны быть объединены. В методе ML-SVM для решения проблемы бинарной классификации предлагается применять метод опорных векторов (SVM). Метод ML-SVM имеет следующие особенности: точность этого метода зависит от настраиваемых параметров, время обучения растёт квадратично с увеличением размера обучающего набора и для дообучения этого метода необходимы примеры, которые ранее уже использовались в процессе обучения (а следовательно, этот метод имеет достаточно долгое время дообучения и требует хранения обучающего набора).

Учитывая то, что ни один из существующих методов multi-label классификации не имеет возможности дообучения с использованием только новых обучающих документов, и важность информации о ранжировании тем для задачи фильтрации Web-трафика, мы решили разрабатывать наш метод multi-label классификации на основе дообучаемого метода ранжирования с пороговой функцией.

Среди методов ранжирования возможностью дообучения с использованием только новых обучающих документов обладают нейросетевые методы: алгоритм Multiclass-Multilabel Perceptron [10,11] и метод, основанный на декомпозиции multi-label проблемы в набор бинарных проблем, в котором в качестве метода бинарной классификации выступает двуклассовый алгоритм персептрона [10,11].

Для преобразования методов ранжирования в методы multi-label классификации существуют следующие методы:

- «отсечение по нулевому порогу» [3,4,5];
- нахождение пороговой функции в пространстве признаков методом наименьших квадратов [1,2].

Однако первый из этих методов даёт низкую точность с нейросетевыми методами ранжирования, а для второго необходим сразу весь обучающий набор, что приводит к тому, что полученный метод multi-label классификации не имеет возможности дообучения с использованием только новых обучающих документов.

Учитывая описанные выше недостатки существующих методов применительно к задаче фильтрации Web-трафика, мы разработали метод multi-label классификации с возможностью дообучения только на новых обучающих документах. То есть разработанный метод получает текущий обучающий документ, извлекает из него всю полезную для обучения информацию, а затем переходит к обработке следующего обучающего документа, уже не возвращаясь ни к одному из предшествующих.

1. Постановка задачи

Используя модель векторного пространства, каждый *документ* (текстовый или Web) будем представлять как вектор в пространстве $X = \mathbb{R}^n$ входных векторов, называемом пространством признаков: $\vec{x} \in \mathbb{R}^n$. Будем называть документ *многотемным (multi-label)*, если он может быть отнесён более чем к одной теме (классу, категории), которые называются *релевантными* для данного документа.

Алгоритм машинного обучения строит процедуру классификации документов, используя математические методы получения знаний из заданного набора обучающих документов, предварительно отрубрицированных человеком. Каждому из обучающих документов сопоставлен свой набор релевантных тем из предопределённого набора тем. Обозначим этот набор возможных тем через Y, а число различных тем в этом наборе обозначим через q (то есть $Y = \{1,...,q\}$, $q = |Y|$). Выходное пространство для алгоритмов multi-label классификации будем рассматривать как пространство, образованное всеми наборами целых чисел от 1 до q. Такое выходное пространство состоит из 2^q различных наборов, и каждый выход соответствует одному набору тем.

Таким образом, цель алгоритмов *multi-label классификации* состоит в том, чтобы на основе обучающей совокупности $S = \{(\vec{x}_i, y_i) \mid 1 \le i \le m, \vec{x}_i \in \mathbb{R}^n, y_i \subset Y\}$ построить классифицирующую темы функцию $f : X \to 2^q$, которая получает тестовый документ $\vec{x}$ и определяет релевантные для него темы. Вывод алгоритмов multi-label классификации есть набор релевантных тем $y \subset Y$ (размер которого заранее не известен), и качество предсказания измеряется точностью определения этого набора.

Пусть $\vec{r}(\vec{x}) = (r_1(\vec{x}),...,r_q(\vec{x}))$ - функция ранжирования тем, где $r_l(\vec{x})$ - релевантность темы l для документа $\vec{x}$. То есть функция ранжирования осуществляет упорядочение всех тем из набора Y в порядке их релевантности для заданного документа $\vec{x}$. Обозначим через $s(\vec{x})$ размер набора релевантных тем для документа $\vec{x}$. Будем считать, что тема l принадлежит набору релевантных тем для $\vec{x}$, если $r_l(\vec{x})$ есть среди $s(\vec{x})$ наибольших элементов $(r_1(\vec{x}), r_2(\vec{x}),..., r_q(\vec{x}))$. Для преобразования методов ранжирования в методы multi-label классификации необходим предсказатель размера $s(\vec{x})$ набора тем, релевантных для документа $\vec{x}$. Функция $s(\vec{x})$ может вычисляться на основе пороговой функции, разграничивающей релевантные для документа темы от нерелевантных: $s(\vec{x}) = |\{l \mid r_l(\vec{x}) > t(\vec{x}), l \in Y\}|$. Тогда для полного решения всей multi-label задачи остаётся выбрать пороговую функцию $t(\vec{x})$.

С учётом особенностей задачи фильтрации Web-трафика, нами была поставлена задача разработать метод классификации многотемных документов, удовлетворяющий следующим требованиям:

- классификация в режиме online текстовых и гипертекстовых документов;
- высокая точность multi-label классификации;
- возможность работы с наборами данных большого размера:
  - большая размерность пространства признаков n (порядка количества слов в словаре),
  - большая мощность набора Y возможных тем,
  - большое количество документов в обучающем наборе;
- возможность дообучения с использованием только новых обучающих документов.

Разработанный нами метод multi-label классификации обладает возможностью дообучения только на новых обучающих документах. Наш метод является модификацией дообучаемого метода ранжирования в метод классификации многотемных документов путём введения дообучаемой пороговой функции. Ранее в [1,2] предлагалось находить пороговую функцию в пространстве признаков с применением метода наименьших квадратов, но при этом для построения пороговой функции необходимо использовать сразу весь обучающий набор, а при поступлении новых обучающих документов необходимо полностью переобучать пороговую функцию. Нами же предлагается строить пороговую функцию, используя дообучаемый алгоритм персептрона. Весовые коэффициенты персептрона предлагается подбирать, исходя из векторов документов и соответствующих им пороговых значений, вычисленных по значениям функции ранжирования, обученной на всех предыдущих обучающих документах, включая данный.

2. Описание предложенного решения

Решение задачи multi-label классификации на основе метода ранжирования можно представить в виде двух подзадач. Первая подзадача – построение функции ранжирования, осуществляющей упорядочение тем согласно их релевантности для каждого конкретного документа. Вторая подзадача – построение функции multi-label классификации, используя найденную функцию ранжирования.

В [1,2] для решения второй подзадачи был предложен метод нахождения пороговой функции методом наименьших квадратов в пространстве признаков (Threshold on Feature Vector Space, TFVS). Однако применительно к задаче текстовой классификации этот метод имеет долгое время нахождения коэффициентов пороговой гиперплоскости ввиду большой размерности пространства признаков. Для сокращения времени обучения нами предлагается новый подход

[12] для сведения проблемы ранжирования к проблеме multi-label классификации. Нами предлагается строить простые линейные функции multi-label классификации, используя результат работы алгоритмов ранжирования как новое множество признаков анализируемого документа. В рамках предложенного подхода были разработаны два метода модификации методов ранжирования в методы multi-label классификации: TCRS (Threshold function on the Class Relevance Space) и LORM (Linear Operator from class Ranks to Multi-label decision functions, $f=Ar$) [12]. Общая идея этих методов состоит в переходе из пространства признаков в пространство векторов релевантностей тем (пространство векторов релевантностей тем есть пространство векторов, координатами которых являются значения функции ранжирования тем для документов). Для задачи текстовой классификации размерность пространства векторов релевантностей тем значительно меньше размерности пространства признаков, в котором ранее вводилась пороговая функция.

Однако наши методы TCRS и LORM имеют преимущество, по сравнению с методом A. Elisseeff, J. Weston [1,2], только по времени обучения и памяти, но также не имеют возможности дообучения с использованием только новых обучающих документов. В данной статье предлагается дообучаемый метод классификации многотемных документов. На рисунке 1 приведена блок-схема предложенного метода. Как видно из блок-схемы, алгоритм получает текущий обучающий документ, использует его для дообучения функции ранжирования и пороговой функции и после этого уже не использует. То есть алгоритм в каждый момент времени работает только с текущим обучающим документом.

Нами предлагается строить пороговую функцию, используя дообучаемый алгоритм персептрона. Весовые коэффициенты персептрона предлагается подбирать, исходя из векторов документов и соответствующих им пороговых значений:

$$t(\vec{x}_i) = \arg\min_t \left(|\{l \in y_i : r_l(\vec{x}_i) \le t\}| + |\{l \in \overline{y}_i : r_l(\vec{x}_i) > t\}| \right),$$

вычисленных по значениям функции ранжирования, обученной на всех предыдущих обучающих документах, включая данный документ.

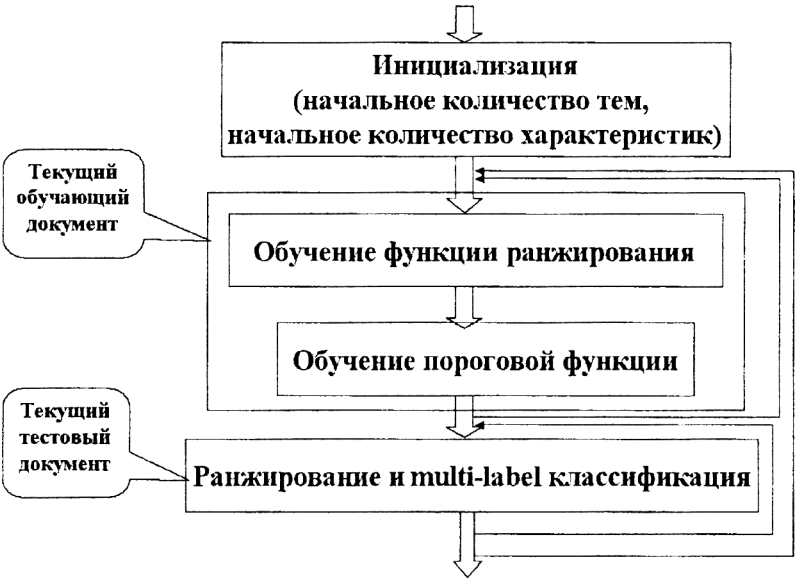


Рисунок 1. Блок-схема разработанного метода

Применение такой дообучаемой пороговой функции (Learning Threshold - LearnTh) для преобразования дообучаемого метода ранжирования в метод multi-label классификации позволило разработать метод классификации многотемных документов с возможностью дообучения с использованием только новых обучающих документов. Для построения функции ранжирования в этом методе используется алгоритм Multiclass-Multilabel Perceptron [10,11], так как этот нейросетевой метод показал лучшие результаты качества классификации с дообучаемой пороговой функцией.

Итак, разработанный метод имеет следующие достоинства:

- Возможность дообучения с использованием только новых обучающих документов:
- Относительно небольшое время обучения/дообучения,
- Эффективность по памяти (необходимо хранить только текущий обучающий документ);
- Высокая точность классификации (метод не имеет параметров, которые надо настраивать);
- Классификация новых документов в режиме online;
- Возможность динамического изменения списка тем (добавление/удаление тем).

Таким образом, разработанный метод полностью удовлетворяет специфике задачи фильтрации Web-трафика.

3. Эксперименты

Тестирование всех разработанных методов проводилось на стандартном наборе многотемных документов Reuters-2000. Набор Reuters-2000 содержит новостные статьи агентства Рейтер за период с августа 1996 по август 1997 года. Число различных тем, к которым классифицированы документы набора Reuters-2000, равно 101. Каждый документ в этом наборе помечается набором релевантных для него тем. Средняя мощность этого набора для Reuters-2000 равна 3.0. Для тестирования исследуемых методов использовались два непересекающихся подмножества (15000 обучающих, 15000 тестовых документов) набора Reuters-2000 в векторном представлении, доступном на сайте [13]. Подмножества обучающих и тестовых документов в каждом из случаев не пересекаются. Размерность пространства признаков для этого векторного представления равна 47236.

Пусть $Test = \{(\vec{x}_i, y_i) \mid 1 \le i \le m,\ \vec{x}_i \in \mathbb{R}^n,\ y_i \subset Y\}$ - совокупность тестовых документов. Пусть система обучения производит функцию ранжирования $r : \mathbb{R}^n \times Y \to \mathbb{R}$, упорядочивающую темы из Y в соответствии с их релевантностью для документа $\vec{x}$ (чем больше $r(\vec{x}, l)$, тем более тема l релевантна). Для оценки качества ранжирования используются следующие четыре общепринятые критерия [8,10,11]: OneError, Average Precision, Coverage, Ranking Loss.

Для оценки *точности multi-label классификации* обычно используется критерий *Hamming Loss* [8], который оценивает различие между предсказанным и истинным наборами тем:

$$HL_{Test}(f) = \frac{1}{m}\sum_{i=1}^{m}\frac{1}{|Y|}\left|(f(\vec{x}_i))\nabla(y_i)\right|,$$

где $a\nabla b = (a \cup b) \setminus (a \cap b)$, $a \subseteq Y$, $b \subseteq Y$,

$f(\vec{x})$ - функция multi-label классификации, которая может быть выражена через функцию ранжирования следующим образом: $f(\vec{x}) = \{l \mid r(\vec{x}, l) > t(\vec{x}), l \in Y\}$, где $t(\vec{x})$ - пороговая функция. Чем меньше значение $HL_{Test}(f)$, тем лучше точность multi-label классификатора.

В таблице 1 приведены результаты тестирования методов ранжирования и multi-label классификации на наборе Reuters-2000. Метод ML-SVM показал самую лучшую точность multi-label классификации, по сравнению со всеми рассмотренными методами. Это преимущество в точности классификации особенно проявляется при небольших размерах обучающего набора. Однако применительно к задаче фильтрации Web-трафика метод ML-SVM имеет ряд недостатков, описанных во введении. В экспериментах на наборе Reuters-2000 мы использовали следующие настройки SVM (из

библиотеки LIBSVM [13]): тип SVM – C-SVC (параметр $C = 1.5$); тип ядерной функции – линейная $K(\vec{x}_i, \vec{x}_j) = \vec{x}_i^T \vec{x}_j$.

При больших же размерах обучающего набора методы ранжирования, базирующиеся на нейросетевых алгоритмах, показывает близкие по точности результаты, а по некоторым критериям ранжирования даже имеет преимущество, по сравнению с методом ML-SVM. Кроме того, для методов, основанных на нейросетевых алгоритмах, время обучения растёт линейно с увеличением размера обучающего набора, а для методов, основанных на SVM, время обучения растёт квадратично.

Из таблицы 1 видно, что метод, основанный на декомпозиции в набор бинарных проблем с применением двуклассового алгоритма персептрона, показывает несколько лучшие результаты ранжирования по сравнению с алгоритмом MMP. Однако при преобразовании этих методов ранжирования в методы multi-label классификации, метод, основанный на MMP, показывает лучшие по точности результаты. Поэтому в качестве метода ранжирования в разработанном методе multi-label классификации был выбран алгоритм MMP.

Применение дообучаемой пороговой функции для преобразования дообучаемого метода ранжирования MMP в метод multi-label классификации позволило разработать метод классификации многотемных документов с возможностью дообучения только на новых обучающих документах. Как видно из таблицы 1, этот разработанный метод multi-label классификации показывает близкие по точности результаты, по сравнению с другими методами. Учитывая специфику задачи фильтрации Web-трафика, при большом размере обучающего набора этот метод становится наиболее эффективен и удобен за счёт возможности дообучения.

Таким образом, на основе проведённого анализа предлагается следующее решение задачи фильтрации Web-трафика. На начальном этапе, пока обучающий набор ещё не достаточно большой, наиболее целесообразно обучить как предложенный нейросетевой метод multi-label классификации, так и метод ML-SVM, и применять метод ML-SVM для классификации новых документов до поступления новой порции обучающих документов. При поступлении новых обучающих документов будем дообучать только предложенный нейросетевой метод, для дообучения которого не требуются документы, на которых ранее проходило обучение. Для классификации новых документов после этапа дообучения предлагается использовать уже не ML-SVM, а наш дообученный нейросетевой метод.

Метод	One Error *100%	AvgP *100%	Cover	Rank Loss	Ham-ming Loss
AdaBoost.MH	8.28	86.16	6.080	0.0169	0.0146
ML-kNN	9.41	85.67	5.721	0.0167	0.0135
ML-SVM	4.01	92.14	6.534	0.0162	0.0095
KNN +TFVS	8.09	86.79	5.802	0.0167	0.0131
KNN +TCRS					0.0130
KNN +(f=Ar)					0.0137
MMP+(>0)	6.57	89.59	5.657	0.0137	0.3031
MMP+TFVS					0.0126
MMP+TCRS					0.0126
MMP+(f=Ar)					0.0129
MMP+ (LearnTh)					0.0128
bPerc+(>0)	6.43	89.92	4.335	0.0099	0.0337
bPerc+TFVS					0.0128
bPerc+TCRS					0.0127
bPerc+(f=Ar)					0.0130
bPerc +(LearnTh)					0.0130

Таблица 1. Результаты тестирования различных методов ранжирования и multi-label классификации на наборе данных Reuters-2000 (101 тема, 47236 признаков, 15000 обучающих, 15000 тестовых образцов)

Заключение

В данной статье рассматривается задача классификации многотемных документов в рамках проблемы анализа и фильтрации интернет информации. Для решения этой задачи авторами был выбран подход, основанный на преобразовании методов ранжирования в методы классификации многотемных документов. Авторами был предложен метод классификации многотемных документов с возможностью дообучения только на новых обучающих документах. То есть разработанный метод получает текущий обучающий документ, извлекает из него всю полезную для обучения информацию, а затем переходит к обработке следующего обучающего документа, уже не возвращаясь ни к одному из предшествующих. До настоящего момента метода классификации многотемных документов с такой возможностью дообучения не существовало.

Эксперименты с разработанным методом на стандартном наборе многотемных данных Reuters-2000 показали высокое качество ранжирования тем и высокую точность классификации многотемных документов. Таким образом, разработанный метод полностью удовлетворяет специфике задачи фильтрации Web-трафика.

Литература

1. Elisseeff A., Weston J. Kernel methods for multi-labelled classification and categorical regression problems // Technical report. BIOwulf Technologies, 2001

2. Elisseeff A., Weston J. A kernel method for multi-labelled classification // Proceedings of the 14th Neural Information Processing Systems (NIPS) Conference, Cambridge, 2002. P. 681–687

3. Schapire R. E., Singer. Y. Improved boosting algorithms using confidence-rated predictions // Machine Learning. 1999. 37. N 3. P. 297-336

4. Schapire R. E., Singer. Y. BoosTexter: A boosting-based system for text categorization // Machine Learning. 2000. 39. N 2-3. P. 135-168

5. Comite F. D., Gilleron R., Tommasi M. Learning multi-label alternating decision tree from texts and data // Machine Learning and Data Mining in Pattern Recognition, MLDM 2003 Proceedings, Lecture Notes in Computer Science 2734. Berlin, 2003. P. 35–49

6. Freund Y., Mason L. Alternating decision tree learning algorithm // In Proc. 16th International Conf. On Maching Learning. San Francisco, USA, 1999. P. 124-133

7. McCallum A. Multi-label text classification with a mixture model trained by EM // Working Notes of the AAAI'99 Workshop on Text Learning, Orlando, FL, 1999

8. Zhang M.-L., Zhou Z.-H. A k-nearest neighbor based algorithm for multi-label classification // Proceedings of the 1st IEEE International Conference on Granular Computing (GrC'05). Beijing, China, 2005. P.718-721

9. Boutell M. R., Luo J., Shen X., Brown C.M. Learning multi-label scene classification // Pattern Recognition. 2004. N 37. P. 1757-1771

10. Crammer C., Singer Y. A new family of online algorithms for category ranking // Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval. Tampere, Finland, 2002. P. 151 – 158

11. Crammer C., Singer Y. A family of additive online algorithms for category ranking // Journal of Machine Learning Research. 2003. N 3. P. 1025–1058

12. Mikhail Petrovskiy, Valentina Glazkova. Linear Methods for Reduction from Ranking to Multilabel Classification // Springer-Verlag, Lecture Notes in Artificial Intelligence, 2006, vol. 4304, pp. 1152-1156

13. Chih-Chung Chang, Chih-Jen Lin. LIBSVM : a library for support vector machines (http://www.csie.ntu.edu.tw/~cilin/libsvm)

Курынин Р.В., Машечкин И.В., Петровский М.И.

Об одном методе нечетких деревьев решений в задаче анализа и прогнозирования качества продукции в производственном процессе[1]

Введение

Производственный процесс (процесс производства какого-либо продукта) определяется рядом измеримых характеристик и качеством полученного продукта. При организации любого производственного процесса необходимо проводить анализ его качества, то есть не только фиксировать качество продукта, но и определять, какая из характеристик производственного процесса влияет на результат.

Итак, пускай имеется некоторый автоматизированный процесс, с некоторых этапов которого происходит съем информации о текущих его характеристиках (будь то давление, температура и пр. — см. рисунок 1).

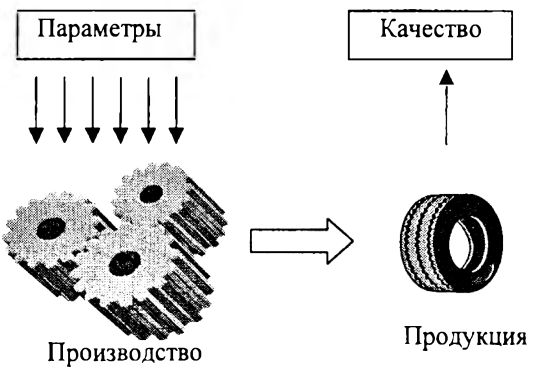


Рисунок 1.

И требуется построить классификатор, строящий модель зависимости качества продукции от характеристик производственного процесса. Кроме того, от модели требуется, чтобы она позволяла прогнозировать ожидаемое качество изделия, и чтобы эта модель была пригодной для интерпретации экспертом.

[1] Работа выполнена при поддержке грантов РФФИ 06-01-00691 и НШ 02.445.11.7427.

1.Постановка задачи

Разработка алгоритмов и экспериментальной программной системы, обеспечивающей:

- построение модели зависимости качества продукции от характеристик производственного процесса, представимой в аналитическом виде, пригодном для интерпретации;
- прогнозирование ожидаемого качества изделия по характеристикам производственного процесса;
- упорядочивание характеристик процесса по степени влияния на качество.

2.Обзор существующих подходов

В настоящее время для текущего контроля качества применяются в основном статистические методы, основанные на изучении характеристик получаемого продукта, а не процесса его производства. Одним из таких методов анализа является так называемый метод контрольных карт Шуерта [1].

Задача анализа контроля качества продукции может быть сформулирована в терминах задачи классификации. Объектом в этом случае будет являться один вектор, характеризующий процесс производства продукта, набор значений векторов является набором значений атрибутов объектов, а качество продукта будет определять класс объекта.

Существует несколько классов алгоритмов классификации, самыми распространёнными из которых являются [2, 3]: статистические методы; нейронные сети; деревья решений.

Современные статистические методы, такие как линейный дискриминант Фишера [4] и наивный алгоритм Байеса [4, 5], используют гибкие классы моделей, многие из которых позволяют оценивать совместное распределение признаков внутри класса, по которому можно построить классифицирующее правило. Но и эти статистические подходы характеризуются тем, что строят расширенную вероятностную модель, определяющую вероятность принадлежности какому-либо классу, а не проводят простую классификацию. Эта модель не может быть представлена в виде системы правил «если...то...», следовательно, статистические методы не годятся для решения задачи анализа качества.

Нейронные сети — это самообучающиеся системы, использующие для обработки сигналов явления, аналогичные происходящим в нейронах живых существ [4]. Искусственные нейронные сети состоят из элементов, функциональные возможности которых аналогичны большинству элементарных функций биологического нейрона. В общем случае нейронная сеть состоит из

слоёв соединённых между собой нейронов, каждый из которых выдаёт на выходе значение нелинейной функции от данных на своих входах. Как правило, построенные классификаторы достаточно эффективны, если выбрана удачная начальная конфигурация сети. Однако нейронные сети в большинстве своём действуют по принципу «чёрного ящика» - внутреннее устройство сети практически невозможно проинтерпретировать. То есть нельзя определить наиболее используемые значимые атрибуты объекта и тем более нельзя упорядочить их по степени влияния.

Таким образом, статистические методы, равно как и нейронные сети, не дают системы правил, а нейронные сети, кроме того, не позволяют выявить наиболее значимые атрибуты и упорядочить их по степени влияния на качество продукта.

Метод деревьев решений — наиболее популярный на сегодняшний день метод решения задачи классификации [6]. Его популярность основывается на том, что построенная модель данных легко интерпретируется. Дерево решений представляет собой древовидную структуру, в которой внутренним вершинам приписаны атрибуты объекта, листовым — классы, а ветвям — условия, относящиеся к значениям атрибута, приписанного родительскому узлу. Эта структура легко может быть преобразована в систему правил «если...то...» и поэтому алгоритм деревьев решений был выбран в качестве основы предлагаемого алгоритма.

На сегодняшний день существует значительное число алгоритмов, реализующих деревья решений CART, C4.5, NewId, ITrule, CHAID, CN2 и т.д. [3].

Недостатком классических деревьев решений является то, что они не предназначены для работы с непрерывными атрибутами, поэтому если область определения атрибута x представляет собой отрезок $[a, b]$, то этот отрезок разбивается на k частей и, таким образом, проводится дискретизация: атрибут x принимает фиксированное число значений, равное k. Такое решение для непрерывных атрибутов делает деревья решений неустойчивыми, поскольку малейшее отклонение от порогов разбиения, возможно, вызванное шумом в данных, может привести к тому, что атрибуту x присвоится неверное дискретное значение, что серьёзно повлияет на результат.

Нечёткие деревья решений ([6, 7]) решают обозначенную проблему и потому обладают большей устойчивостью и точностью.

Нечёткое дерево имеет структуру, описанную выше: узлам дерева приписаны атрибуты, листьям — классы, и для каждого потомка ведущей к нему дугой задаётся функция «меры принадлежности» этому потомку A_j. Но в отличие от классических деревьев решений, при построении и классификации в нечётком дереве пример может быть отнесён и правой, и левой ветвям.

Таблица 1.

	скорость	точность	работа с дискретными/ пропущенными значениями	интерпретируемость	сортировка характеристик по значимости
Байесовские методы	высокая	невысокая	нет	низкая	да
Нейронные сети	низкая	высокая	нет	низкая	нет
Классические деревья решений	высокая	невысокая	да	высокая	да
Нечеткие деревья решений	высокая	высокая	нет	высокая	да

Достоинства и недостатки указанных выше методов приведены в Таблице 1.

3.Предлагаемое решение

Предлагаемый алгоритм основывается на алгоритме нечётких деревьев решений, но скорректирован так, что позволяет использовать в дереве как непрерывные, так и дискретные атрибуты.

Процедура построения нечёткого дерева решений аналогична процедуре построения классических деревьев решений (см. Таблицу 2).

Таблица 2. Модифицированный алгоритм нечетких деревьев решений (Soft Decision Trees — SDT).

Шаг 1	Поставить в соответствие корню множество всех примеров тренировочного набора, пометить корень как «необработанный».
Шаг 2	Проверить, есть ли ещё необработанные узлы, если есть — выбрать узел и перейти к шагу 3. Если нет — остановка, дерево построено.
Шаг 3	Проверить, не выполнено ли условие отсечения. Если да — удалить текущий узел, перейти к шагу 2. Если нет — перейти к шагу 4.
Шаг 4	Проверить, не выполнено ли условие достижения листа.

	Если ***да***, то	– текущий узел — лист; – приписать ему класс C_j; – пометить узел как «обработанный»; – перейти к шагу 2.
	Если ***нет***, то	– выбрать атрибут разбиения; – если атрибут разбиения дискретный и принимает значения $a_1 \ldots a_m$, то образовать m потомков, если непрерывный — образовать двух потомков; – связать образованные узлы дугами с родителем, приписать потомкам множество примеров S и вычислить для каждого примера ***меру принадлежности узлу***; – присвоить ветвям дерева условие $A_j(s_{x_i})$; – пометить все полученные узлы как «необработанные», а родительский узел — как «обработанный»; – перейти к шагу 2.

Данный алгоритм отличается от алгоритма нечетких деревьев SDT способом вычисления меры принадлежности узлу. Предлагается использование минимаксной нечеткой логики вместо суммарно-продуктивной. Такой подход позволяет работать не только с непрерывными атрибутами, но и с дискретными, а также обрабатывать пропущенные значения.

Для каждого примера вычисляется мера принадлежности узлу, как если бы последний был потомком корня $A^t(s_j)$, а потом учитывается мера принадлежности родителю.

$$A^{k+1}(s_j) = \min\left(A_t(s_j); A^k(s_j)\right) \tag{1}$$

При определении критерия, по которому производится выбор атрибута (а для непрерывных атрибутов — и точки разбиения), используется ***нечеткая энтропия разбиения*** — сумма энтропий каждого из полученных после разбиения множеств.

Пусть при разбиении множества S по атрибуту X образовано k потомков, каждому из которых приписано подмножество S_k множества S (в случае, если мера принадлежности примера ветви равна 0, можно исключить его из множества).

Тогда энтропия вычисляется по формуле:

$$Info_X(S) = \sum_{j=1}^{k} \frac{N^{S_j}}{N^S} Info(S_j) \tag{2}$$

где $N^S = \sum_{i=1}^{|S|} A(S_{x_i})$ — степень покрытия всех примеров родительским узлом; $N^{S_j} = \sum_{i=1}^{|S|} A_j(a_i), j = 1..k$ — степени покрытия всех примеров потомками; $Info(S_j) = -\sum_{t=1}^{K} p(c_t, S_j) \log_2\left(p(c_t, S_j)\right), j = 1..k$ — энтропии подмножеств примеров, приписанных потомкам (где $p(c_t, S_j) = \frac{N^{S_j c_t}}{N^{S_j}}, j = 1..k$, а $c_1..c_K$ — имеющиеся классы), $\overline{N}^{S_j} = \sum_{i=1}^{|S_j|} A_j(a_i)$ —степени покрытия примеров каждым из потомков; $N^{S_j c_t} = \sum_{a_i \in c_t} A_j(a_i)$ — степени покрытия класса c_t каждым из потомков.

Для дискетного атрибута энтропия подсчитывается один раз — для разбиения по всем возможным значениям. Для непрерывного атрибута энтропия подсчитывается для каждого возможного варианта разбиения. Варианты определяются следующим образом. Пусть $a_1 ... a_N$ — все встречающиеся в родительском наборе значения исходного атрибута, причём учитываются только значения тех примеров, мера принадлежности которых данному узлу не ниже некоторого порогового значения (>0.7), — это позволяет упростить и ускорить процесс построения дерева, не снижая точности построенной модели. Тогда подсчитывается значение среднего расстояния между значениями атрибутов: $Average = \frac{1}{N} \sum_{i=1}^{N-1} (a_{i+1} - a_i)$. В качестве порогов разбиения выбираются точки $T_i = \frac{a_i + a_{i+1}}{2}, i = 1..N-1$, лежащие посередине между известными значениями атрибутов и являющиеся порогами разбиения. Тогда левая и правая границы нечёткого разбиения определяются, как $T_i - shift$ и $T_i + shift$, где $shift = \frac{Average}{2}$.

Из всех энтропий, подсчитанных для дискретных атрибутов, и всех вариантов разбиений нечётких атрибутов, выбирается тот атрибут и то разбиение, энтропия которого минимальна.

Потомок (которому приписано множество S_j) считается листом и дальше не разбивается, если ветви принадлежат примеры одного класса и их число не менее некоторого порогового значения ($\max_{j=1..k, t=1..K} (N^{S_j c_t}) > 0.95$). Ветвь отсекается, если сумма мер принадлежностей всех примеров ниже определённого порога ($N^{S_k} < 0.01$).

4.Применение построенного классификатора

Степень принадлежности объекта может быть записано в следующей форме, с использованием функции ***is***:

x ***is*** $A = \mu_A(x)$ (3)

Атрибут x представляет собой нечёткую переменную. Так же, как и в обычной логике, в нечеткой логике можно определить операторы ***and*** и ***or***. Возможны различные реализации этих операций. В случае классического алгоритма SDT эти операции определены, соответственно, как произведение и сумма. Предложенный алгоритм использует минимаксную нечёткую логику, и реализует операцию ***and*** как *min*, а операцию ***or*** как *max*. Более подробная информация о теории нечётких множеств может быть найдена в [8].

Для каждого листа в дереве определён единственный путь к нему из корня дерева, состоящий из k узлов (X_{j1}, X_{j2},…,X_{jk}), где X_{jt} — атрибуты, $\{j1,\ldots,jk\} \in \{1,\ldots n\}$.

Каждому узлу в этой цепочке, кроме корня X_{j1} и листа X_{jk} поставим в соответствие условие правила X_{jt} ***is*** $A^{jt}(x)$, где $A^{jt}(x)$ — функция меры принадлежности узлу.

Правило может трактоваться по-разному, в зависимости от типа атрибута X_{jt}.

Если атрибут дискретный, то мера принадлежности узлу равна нулю для всех значений атрибута, кроме одного. Предположим, что это значение a_m. Тогда условие X_{jt} ***is*** $A^{jt}(x)$ можно трактовать как $X_{jt} = a_m$.

Если атрибут непрерывный, то $A^{jt}(x)$ является нечётким множеством, а функция ***is*** — функцией степени принадлежности объекта, используемой в нечёткой логике, ***and*** — оператором нечёткой логики.

Полученное правило будет иметь вид:

if X_{j1} ***is*** $A^{j1}(x)$ ***and*** X_{j2} ***is*** $A^{j2}(x)$ ***and*** … ***and*** X_{jk} ***is*** $A^{jk}(x)$

then $X \in C_t$ с мерой принадлежности ***Membership***(X, C_t) (4)

$$Membership(X, C_k) = \bigvee_{\substack{\text{по всем} \\ \text{листам}\ i}} \min\left(\bigwedge_t A^{j_t}(x), \frac{Count(i, C_k)}{Count(i)} \right) \quad (5)$$

При анализе дерева будет получено столько правил, сколько листьев имеется в дереве.

Классификация примера с использованием полученной системы правил происходит следующим образом:

1. Для каждого правила:
   - Вычисляются все значения X_{jt} ***is*** $A^{jt}(x) = A^{jt}(x^{Xjt})$. Если значение атрибута неизвестно, то значение $A^{jt}(x^{Xjt})$ считается как $1/n$, где $n = 2$ для непрерывных атрибутов, для дискретного $n = m$ — число значений из области определения атрибута.
   - К подсчитанным значениям применяется оператор ***and***.

2. Для каждого класса применяется оператор ***or*** к значениям, полученным по правилу, вывод которого — рассматриваемый класс.
3. Результат — набор вероятностей принадлежностей каждому классу.

Построенное дерево решений может быть использовано для **упорядочивания** атрибутов по степени значимости, если кроме мер принадлежности узлу для каждого узла сохранять значение энтропии разбиения, полученное в ходе построения дерева.

Уровнем дерева будем называть множество узлов дерева, равноудалённых от корня.

Атрибуты, принадлежащие уровню $m + 1$ менее значимы, чем атрибуты, принадлежащие уровню m.

Пусть $(X_{j1}, X_{j2}, ..., X_{jk})$ — узлы дерева (рассматриваются не листовые узлы), принадлежащие уровню m, а соответствующие им значения энтропии разбиений $(S_{j1}, S_{j2}, ..., S_{jk})$. Если отсортировать значения энтропий в порядке возрастания, то мы получим порядок убывания значимости атрибутов.

Если в ходе построения дерева для некоторых векторов значения атрибутов **неизвестны**, то такие вектора не учитываются при подсчёте энтропий по этим атрибутам, а если производится разбиение по одному из этих атрибутов, то вектора приписываются всем образованным потомкам.

Мера принадлежности такого вектора потомку равна мере его принадлежности родителю.

5.Эволюционная настройка дерева

Процесс оптимизации сводится, в общем случае, к созданию всех возможных комбинаций значений характеристик, влияющих на целевую функцию, и нахождение такой комбинации, которая приводит к ее экстремальному значению. Все возможные комбинации аргументов при этом образуют пространство поиска задачи, размерность которого определяется числом аргументов целевой функции. А каждая из указанных комбинаций образует точку в данном пространстве.

Эволюционный алгоритм — алгоритм поиска, оптимизации или обучения, основанный на некоторых формализованных принципах естественного эволюционного процесса, обеспечивающий улучшение находимого решения. Информация о работе эволюционных алгоритмов может быть найдена в [9]. Описание применения эволюционных алгоритмов к генерации и настройке систем нечётких правил приведено в [10].

В данной работе используется эволюционный алгоритм настройки структуры нечёткого дерева решений.

Кодирование особи. Особь представляет собой вектор вещественных значений, кодирующих правую и левую границы нечёткого разбиения. Формирование особи, участвующей в дальнейшем в эволюционной настройке, происходит следующим образом: производится обход дерева и для каждого нечеткого атрибута разбиения добавляются две координаты в вектор особи, которым присваиваются значения нечетких границ. Настройка дерева на некоторую полученную в ходе эволюции особь представляет собой обратное преобразование — обход дерева и присвоение соответствующим границам разбиения значений, содержащихся в полученной особи.

Определим целевую функцию эволюционного процесса как функцию ошибки нечёткого дерева. Функция ошибки задаётся следующим образом. Кроме тренировочного набора S в каждом случае при построении дерева имеется тестовый набор S^T, для каждого объекта из которого, также как и для объекта из S, известен его класс. Пусть истинные значения классов из S^T $(C_L^1, C_L^2, ...C_L^{|S^T|})$.

Примеры из S^T классифицируются с помощью нечёткого дерева, то есть работает построенная функция классификации $f(s)$ для $\forall s \in S^T$. Функция ошибки равна:

$$F(S^T) = \sum_{s_i \in S^T} \sigma(s_i), \qquad \sigma(s_i) = \begin{cases} 1, & f(s_i) = C_L^{\,j} \\ 0, & f(s_i) \neq C_L^{\,j} \end{cases} \tag{6}$$

Таким образом, для каждой особи, представляющей собой набор настроек границ, определена функция ошибки на тестовом наборе, на минимизацию которой и будет направлена эволюционная настройка.

В ходе работы эволюционного алгоритма к набору особей, называемому ***популяцией***, применяется ряд **операторов**, моделирующих естественный эволюционный процесс. Генерация новых особей происходит на основе моделирования процесса размножения, за счёт работы **оператора скрещивания**. Моделирования процесса мутации новых особей осуществляется за счёт работы **оператора мутации**.

Поскольку размер популяции фиксирован, то порождение потомков должно сопровождаться уничтожением других особей. Выбор пар родителей из популяции для порождения потомков осуществляет **оператор отбора**, а выбор особей для уничтожения — **оператор редукции**.

- **Оператор мутации** производит сдвиг произвольного количества элементов векторов на величину фиксированной размерности (в предлагаемом решении — 0.001).

- **Оператор скрещивания** на основе двух старых особей позволяет получить две новые особи путем перестановки некоторых соответствующих компонент в исходных векторах.
- **Операторы отбора и редукции** работают на основе значений функций качества: M лучших особей участвуют в размножении, а остальные отбрасываются.

В представленном решении предлагается осуществить 10 смен поколений, причем каждое новое поколение формируется из 50 лучших особей предыдущего поколения (получаемые применением оператора отбора и редукции), 50 потомков лучших особей предыдущего поколения (получаемые после применения оператора скрещивания) и 50 мутантов (получаемых в ходе применения оператора мутации к лучшим 50 особям предыдущего поколения).

6.Апробация

У каждого из алгоритмов классификации есть своя область применения и каждый из них обладает своими возможностями и ограничениями. Для такой характеристики, как точность предсказания, например, доказан закон сохранения эффективности обобщения Шеффера (Schaffer's conservation law of generalization performance) [11], подтверждающий, что один алгоритм не может постоянно превосходить другие по точности предсказания.

По этой характеристике, как по наиболее значимой, было проведено сравнение с некоторыми существующими алгоритмами классификации.

Сравнение проводилось с результатами тестирования следующих классических реализаций алгоритмов, находящимися в свободном доступе:

- наивный алгоритм Байеса;
- нейронные сети (многослойный персептрон);
- классические деревья решений (C4.5).

Также учитывались результаты работы алгоритма SDT без эволюционной настройки и модифицированной меры принадлежности узлу.

Точность модели, построенной с помощью предлагаемого алгоритма, проверялась на реальных производственных данных одной из западных компании по производству автомобильных шин.

Сравнение проводилось путём серии из 30 перекрёстных проверок (cross-validation). Исходные наборы данных разбивались в отношении 2:1: на большей части алгоритмы обучались, на меньшей — тестировались.

Первый набор содержит 178 записей по 14 измерений в каждом: 13 измерений — параметры процесса производства,

непрерывные атрибуты, 14-е измерение — собственно, класс полученной продукции, то есть уровень её качества. Качество может быть трёх уровней: «хорошее», «необходима доработка» и «брак».

Как можно видеть из первой строки Таблицы 3, нейронные сети (MLP) и наивный алгоритм Байеса (Bayes) показали лучший результат по сравнению с предлагаемым алгоритмом. Однако они не могут быть применены для решения поставленной задачи, так как построенная ими модель не представима в виде системы правил «если... то...». Из алгоритмов, которые могут построить такую модель и отвечают остальным требованиям, реализованный алгоритм показал лучший результат.

Таблица 3. Средняя ошибка (в процентах) при построении модели на двух наборах данных.

	Bayes	**MLP**	**C4.5**	**SDT**	**Предложенный подход**
Набор 1	5.846%	3.33%	8.954%	8.584%	6,66%
Набор 2	58,13%	—	54,99%	—	47,25%

Второй набор содержит 84 записи по 48 измерений в каждом (20 дискретных и 28 непрерывных атрибутов). Набор данных небольшой и достаточно зашумлённый, в нем высок процент записей с пропущенными значениями. Во второй строке Таблицы 3 изложены результаты тестирования по второму из имеющихся наборов. Высокий уровень ошибки объясняется тем, что данных в наборе чрезвычайно мало и многие атрибуты отсутствуют. Алгоритм нейронных сетей не может обрабатывать отсутствующие значение, а классический алгоритм SDT не предназначен для работы с дискретными значениями, поэтому тестирование этих алгоритмов не проводилось. Как видно, алгоритм Байеса и классические деревья решений не смогли построить сколько-нибудь точной модели (уровень ошибки приближается к 60%), предложенный же алгоритм смог построить модель, имеющей какую-то значимость (поскольку уровень ошибки хоть и высок, но все-таки чуть ниже 50%). Отметим, что результаты тестирования для второго набора приведены здесь с иллюстративной целью.

Заключение

В ходе работы были исследованы существующие методы и подходы построения алгоритмов анализа производственного процесса. В результате исследования в качестве основы предлагаемого алгоритма анализа качества был выбран алгоритм нечётких деревьев решений. Алгоритм был реализован и модифицирован, а также дополнен системой генетической тонкой настройки параметров. Результирующий

алгоритм удовлетворяет требованиям постановки задачи. Апробация на реальных экспериментальных данных одной из западных компаний по производству автомобильных шин выявила, что наибольшую точность показал предложенный метод, а также метод нейронных сетей и наивного алгоритма Байеса. Но методы нейронных сетей и наивного алгоритма Байеса не удовлетворяют поставленным требованиям, так как не могут построить систему правил вида «если... то...». Предлагаемый алгоритм строит внятную и легко интерпретируемую модель процесса и более точен, чем другие алгоритмы решения задачи, которые могли бы быть использованы.

Литература

1. Henry R. Neave, Donald J. Wheeler. Shewhart's Charts and Probability Approach (1996).
2. M.Vazirgiannis. Data Mining: Concepts and Techniques. Tutorial. Dept. of Informatics, Athens Univ. of Economics & Business, Athens, Greece, ADBIS, Sept 2001.
3. J.Gama, P.Brazdil. Characterization of Classification Algorithms.
4. D.Michie, D.J.Spiegelhalter, C.C. Taylor. Machine Learning, Neural and Statistical classification, Ellis Horwood, 1994.
5. David D. Lewis. Naïve (Bayes) at Forty: The Independence Assumption in Information Retrieval, In ECML-98: Proc. of the 10th European Conference on Machine Learning, pp. 4-15, Chemnitz, Germany, April 1998. Springer.
6. Yonghong Peng, Peter A. Flach: Soft Discretization to Enhance the Continuous Decision Tree Induction, Department of Computer Science, University of Bristol, Bristol, BS8 1UB, UK.
7. Jens Zeidler, Michael Schlosser: Continuous-Valued Attributes in Fuzzy Decision Trees, In IPMU 96, Proc. of the 6th International Conference Information Processing and Management Uncertainty in Knowledge-based systems, pp.395-400, 1996.
8. Нечёткие множества и теория возможностей. Последние достижения: Пер. с англ./Под ред. Р.Р.Ягера. – М.: Радио и связь, 1986.
9. Осовский С.: Нейронные сети для обработки информации / Пер. с польского И.Д.Рудинского, М.:Финансы и статистика,2002.
10. O. Cordón, M.J. del Jesus, F. Herrera, M.Lozano: MOGUL: A Methodology to Obtain Genetic fuzzy rule-based systems under the iterative rule Learning approach (1998), Department of Computer Science and Artificial Intelligence.
11. Shaffer C.: A conservation Law for Generalization Performance (1994), in Machine Learning, Proc. Of 11th International Conference, ed. W.Cohen and H.Hirsh, Morgan Kaufmann Publishers.

Машечкин И. В., Петровский М. И., Трошин С. В.

Система мониторинга и анализа поведения пользователей компьютерной системы[1]

1. Введение

В статье рассматриваются вопросы построения эффективных программных систем защиты от внутренних вторжений, основанных на несигнатурных методах и обладающих свойствами автономности, адаптируемости и самообучаемости. Отдельно рассмотрены проблемы консолидации исходных данных из журналов регистрации и протоколов ОС, методы промежуточного представления, передачи данных и хранения собранных данных. Предложена архитектура системы консолидации и рабочего места аналитика безопасности. Предложены методы применения технологии OLAP для анализа собранных данных об активности пользователей, а также алгоритмы интеллектуального анализа данных (Data Mining) для построения модели поведения пользователя на основе ассоциативных правил. Построенная модель поведения может быть использована для визуального представления аналитику безопасности в виде сети зависимостей, а также для автоматического поиска аномалий в поведении пользователей и оценки степени потенциальной угрозы, исходящего от каждого пользователя. Реализована экспериментальная пилотная версия такой системы, которая была верифицирована по методике DARAP Intrusion Detection Evaluation Program, с использованием эталонных наборов данных. Результаты экспериментальной верификации приведены в статье.

2. Актуальность

Развитие и повсеместное внедрение информационных технологий, увеличение объемов и ценности хранимой информации приводит к необходимости ее защиты. Основную угрозу безопасности представляют вторжения в компьютерные системы. Под вторжением (или атакой) в компьютерную систему понимается любая деятельность человека или программы, нарушающая целостность, конфиденциальность и/или доступность данных. Наибольший ущерб причиняется внутренними вторжениями, поскольку такое вторжение осуществляется пользователем (инсайдером), который уже имеет

[1] Работа поддерживается грантом Президента РФ № МК-2111.2005.9 и грантами РФФИ № 06-07-08035-офи, №05-0100744.

доступ к компьютерной системе и даже возможно является легальным пользователем.ю

Традиционно в системах обнаружения вторжений используется сигнатурный метод [1, 2], основная идея которого заключается в сравнении записей о событиях в системе с определенными шаблонами — правилами, описывающими атаки. Так как набор правил фиксирован, такие системы зависят от эксперта и невосприимчивы к новым типам вторжений, пока эксперт не опишет новые правила. В подобных системах обычно используется техника периодического просмотра собранных системой журналов (лог-файлов) аудита и приложений, однако, набор журналов, используемых такими системами, зачастую достаточно ограничен, отчасти из-за необходимости для каждого вида журналов разрабатывать свои правила.

На сегодняшний день актуально создание программных технологий построения эффективных систем защиты от внутренних вторжений, основанных на несигнатурных методах [3, 4] и обладающих свойствами автономности, адаптируемости и самообучаемости.

Подобного рода системы защиты должны решать следующие задачи:

- сбор и анализ статистики работы пользователей и приложений;
- построение и визуализация моделей поведения пользователей;
- выявление аномалий в работе пользователей и ПО;
- выявление внутренних угроз (со стороны инсайдеров).

3. Архитектура

Система мониторинга и анализа поведения пользователей обеспечивает сбор данных, формирование статистических отчетов и интеллектуальный анализ (Data Mining) собранных данных, а именно: построение ассоциаций и моделей поведения, поиск аномалий работы как в рамках отдельной вычислительной системы, так и в рамках сети в целом. Система является мультиагентной. Источником данных являются журналы регистрации ОС и ПО.

Система состоит из сервера консолидации, агентов сбора, рабочего места аналитика (рис. 1). Исходные данные читаются агентом из журналов вычислительных систем и консолидируются на сервере. По собранным данным вычисляются статистические значения, производится анализ данных. Статистические отчеты, найденные ассоциации, построенные модели и выявленные аномалии передаются аналитику в виде интерактивных динамических отчетов.

Рабочее место аналитика — программный модуль, обеспечивающий работу пользователя с системой. Предоставляются следующие возможности.

Машечкин И. В., Петровский М. И., Трошин С. В.

Система мониторинга и анализа поведения пользователей компьютерной системы[1]

1. Введение

В статье рассматриваются вопросы построения эффективных программных систем защиты от внутренних вторжений, основанных на несигнатурных методах и обладающих свойствами автономности, адаптируемости и самообучаемости. Отдельно рассмотрены проблемы консолидации исходных данных из журналов регистрации и протоколов ОС, методы промежуточного представления, передачи данных и хранения собранных данных. Предложена архитектура системы консолидации и рабочего места аналитика безопасности. Предложены методы применения технологии OLAP для анализа собранных данных об активности пользователей, а также алгоритмы интеллектуального анализа данных (Data Mining) для построения модели поведения пользователя на основе ассоциативных правил. Построенная модель поведения может быть использована для визуального представления аналитику безопасности в виде сети зависимостей, а также для автоматического поиска аномалий в поведении пользователей и оценки степени потенциальной угрозы, исходящего от каждого пользователя. Реализована экспериментальная пилотная версия такой системы, которая была верифицирована по методике DARAP Intrusion Detection Evaluation Program, с использованием эталонных наборов данных. Результаты экспериментальной верификации приведены в статье.

2. Актуальность

Развитие и повсеместное внедрение информационных технологий, увеличение объемов и ценности хранимой информации приводит к необходимости ее защиты. Основную угрозу безопасности представляют вторжения в компьютерные системы. Под вторжением (или атакой) в компьютерную систему понимается любая деятельность человека или программы, нарушающая целостность, конфиденциальность и/или доступность данных. Наибольший ущерб причиняется внутренними вторжениями, поскольку такое вторжение осуществляется пользователем (инсайдером), который уже имеет

[1] Работа поддерживается грантом Президента РФ № МК-2111.2005.9 и грантами РФФИ № 06-07-08035-офи, №05-0100744.

доступ к компьютерной системе и даже возможно является легальным пользователем.ю

Традиционно в системах обнаружения вторжений используется сигнатурный метод [1, 2], основная идея которого заключается в сравнении записей о событиях в системе с определенными шаблонами — правилами, описывающими атаки. Так как набор правил фиксирован, такие системы зависят от эксперта и невосприимчивы к новым типам вторжений, пока эксперт не опишет новые правила. В подобных системах обычно используется техника периодического просмотра собранных системой журналов (лог-файлов) аудита и приложений, однако, набор журналов, используемых такими системами, зачастую достаточно ограничен, отчасти из-за необходимости для каждого вида журналов разрабатывать свои правила.

На сегодняшний день актуально создание программных технологий построения эффективных систем защиты от внутренних вторжений, основанных на несигнатурных методах [3, 4] и обладающих свойствами автономности, адаптируемости и самообучаемости.

Подобного рода системы защиты должны решать следующие задачи:

- сбор и анализ статистики работы пользователей и приложений;
- построение и визуализация моделей поведения пользователей;
- выявление аномалий в работе пользователей и ПО;
- выявление внутренних угроз (со стороны инсайдеров).

3. Архитектура

Система мониторинга и анализа поведения пользователей обеспечивает сбор данных, формирование статистических отчетов и интеллектуальный анализ (Data Mining) собранных данных, а именно: построение ассоциаций и моделей поведения, поиск аномалий работы как в рамках отдельной вычислительной системы, так и в рамках сети в целом. Система является мультиагентной. Источником данных являются журналы регистрации ОС и ПО.

Система состоит из сервера консолидации, агентов сбора, рабочего места аналитика (рис. 1). Исходные данные читаются агентом из журналов вычислительных систем и консолидируются на сервере. По собранным данным вычисляются статистические значения, производится анализ данных. Статистические отчеты, найденные ассоциации, построенные модели и выявленные аномалии передаются аналитику в виде интерактивных динамических отчетов.

Рабочее место аналитика — программный модуль, обеспечивающий работу пользователя с системой. Предоставляются следующие возможности.

1. Управление консолидацией (установка и настройка агентов).
2. Управление анализом данных (фильтрация данных, обучение алгоритмов анализа и т.п.).
3. Просмотр отчетов по заданным параметрам.

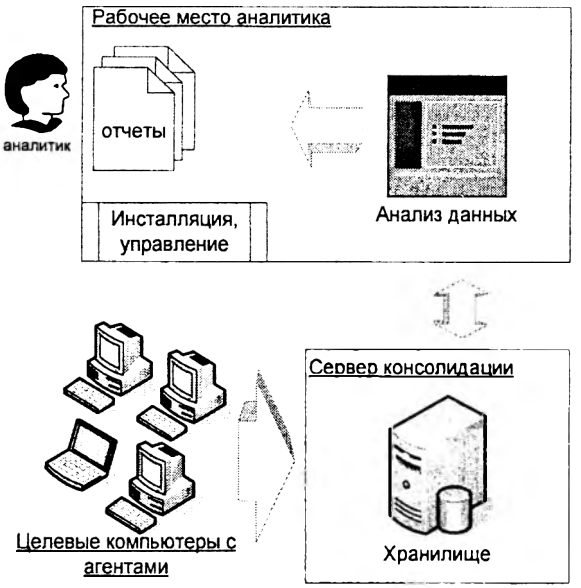


Рис. 1. Схема Система мониторинга и анализа поведения пользователей.

Задачу объединения всех записей журналов в хранилище с предоставлением унифицированного интерфейса доступа, не зависящего от структуры консолидируемых данных, решает подсистема консолидации записей журналов регистрации (логов) — система сбора и предобработки информации из журналов ОС и ПО. Архитектура подсистемы консолидации логов представлена на рис. 2.

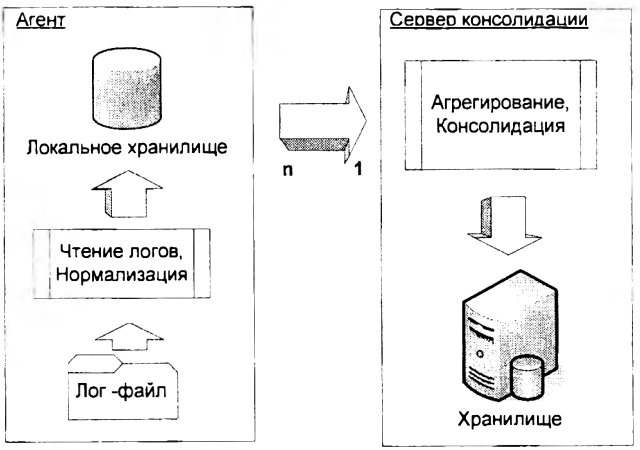


Рис. 2. Схема системы консолидации логов.

Подсистема консолидации состоит из двух частей: серверная часть (сервер консолидации) и агенты, работающие на компьютерах, с которых собираются записи лог-файлов. Агентов может быть произвольное количество. Агенты могут работать на разных платформах и собирать информацию из разных журналов регистрации. Система позволяет добавлять новые агенты или модифицировать существующие «на лету» — без перезапуска сервера консолидации. Агент содержит локальное хранилище для промежуточной буферизации собранных данных перед отправкой, если это подразумевается настройкой передачи данных.

3.1. Агенты

Агент обеспечивает сбор информации о событиях из различных журналов ОС и ПО, установленного на целевом компьютере. Преобразует собранные записи в универсальный формат, разработанный на основе XML [5, 6, 7] и согласно расписанию передает данные на сервер консолидации. Передача собранных агентом данных может осуществляться по одной из следующих стратегий.

1. **Фиксированными объемами данных.** Агент накапливает определенный объем информации или фиксированное количество записей журналов и затем передает их на сервер консолидации.
2. **Через равные промежутки времени.** Агент через равные промежутки времени передает все имеющиеся у него в локальном хранилище данные независимо от их объема.
3. **По расписанию.** Агент передает данные только в указанное время.

4. **В режиме реального времени.** Агент немедленно передает данные о каждой вновь прочитанной записи в журнале регистрации. Данная стратегия наиболее требовательная к ресурсам.

Для обеспечения распределения возникающих при работе агента накладных расходов на чтение и первичную обработку записей журналов регистрации может использоваться механизм разделения нагрузки между целевым компьютером и дополнительным, выделенным для преобразования данных в универсальный формат. Механизм основан на возможности удаленного чтения некоторых журналов регистрации.

3.2. Сервер консолидации

Получая данные от агента, сервер консолидации преобразует записи во внутренний формат и помещает их в хранилище.

Важным параметром работы хранилища и системы консолидации в целом является производительность. Нагрузка на хранилище складывается из нагрузки на добавление вновь получаемых данных лог-файлов и нагрузки на извлечение данных для анализа. Характер этих действий различный. Извлечение данных обычно подразумевает быстрое получение всех собранных записей за некоторый временной период, и эта проблема решается размещением записей в отсортированном по дате порядке. Сложнее дело обстоит с добавлением вновь полученных данных, поскольку один сервер консолидации должен объединять в себе данные от большого количества агентов и еще от большего количества лог-файлов.

Среднее количество записей, помещаемых в журнал регистрации ОС Windows за единицу времени можно оценить по тесту DARPA99 [9]. Примерно 400000 событий в день, что эквивалентно приблизительно 10 событиям в секунду. И, например, для корпоративной сети из 100 компьютеров сервер консолидации должен обрабатывать и добавлять в хранилище порядка 1000 событий в секунду. Каждая запись журнала безопасности ОС Windows состоит приблизительно из 20 параметров (содержательных признаков). Что приводит к 20000 параметрам в секунду.

Проблемы производительности удалось решить, разделив файлы хранилища на три информационные части.

- **Основные параметры**, присутствующие практически во всех записях всех логов: тип события, время генерации и т.п.
- **Вспомогательные параметры**, описывающие дополнительную информацию о событии.

- **Справочники** для хранения реальных значений как основных, так и вспомогательных параметров. Для доступа к значениям справочников используется механизм хэширования.

Запись о событии из лога разделяется между двумя файлами и файлами справочников (рис. 3): файл для хранения идентификаторов основных параметров, файл для хранения идентификаторов вспомогательных параметров и файлы справочников. Файлы справочников позволяют по идентификаторам параметров получать их реальные значения.

Рис. 3. Иерархия файлов.

Сохранение данных в файлах основных и вспомогательных параметров и файлах справочников позволяет в некоторых случаях повысить производительность при обращении к хранилищу. Например, если для анализа собранных данных применять технологию OLAP [9], то для заполнения некоторых фактов в витрине данных (Data Mart) иногда можно обойтись без вспомогательных параметров. Даже если и приходится обращаться к вспомогательным параметрам, далеко не всегда существует необходимость получать их фактические значения, иногда достаточно просто их идентификатора.

Структура специализированного хранилища представлена на рис. 4. Хранилище организованно в виде дерева, в корне которого находятся каталоги с именем домена, внутри каждого каталога с именем домена находятся каталоги с именем компьютеров этого домена. Внутри каталогов с именем компьютера лежат файлы с собранными записями лог-файлов, разделенные по датам, например, дням. Файлы-справочники могут храниться в произвольном месте.

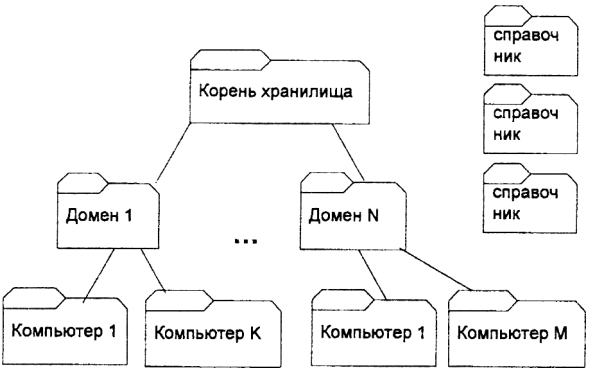


Рис. 4. Размещение объектов в файловой системе.

Иерархия файлов по каталогам (домен — компьютер — дата) не очень принципиальна, однако позволят более легко получать, например, все события из домена или с определенных компьютеров, поскольку обычно для анализа данных требуется получить из хранилища необходимый временной срез и дополнительно применить к нему фильтр (обычно по компьютерам, пользователям, типам логов).

Сервер консолидации может работать только с зарегистрированными агентами, что обеспечивает защиту от «подмены» агента и загрузки на сервер консолидации некорректных данных.

В случае недостаточной производительности сервера консолидации имеется возможность использования промежуточных серверов консолидации.

4. Анализ данных

Сервер консолидации и хранилище не накладывают никаких ограничений на подмножество методов и механизмов анализа данных, которые могут быть применены к данным журналов регистрации. Имеется возможность подключать различные аналитические средства: построение статистики, выявление ассоциативных правил, выявление аномалий и исключений и т.д. В рамках построенной системы реализуются две технологии анализа данных: OLAP и Data Mining.

4.1. Статистика на основе OLAP

Технологии комплексного многомерного анализа данных – OLAP(рис. 5), используемая для анализа собранных данных, допускает возможность осуществления любого логического и статистического анализа, а так же многомерное концептуальное представление данных пользователю.

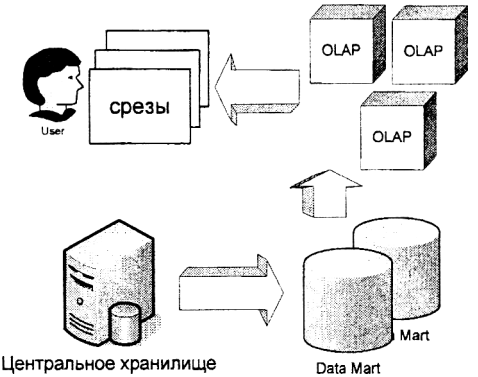


Рис. 5. Применение технологии OLAP.

Для иллюстрации возможностей статистического анализа журнализированной информации с использованием технологии OLAP рассмотрим модельный пример: построение OLAP кубов по данным журнала регистрации ОС Windows.

Из логов ОС Windows выделены факты:

- входа в систему;
- запуска процессов;
- обращения к файлам, папкам и реестру.

По фактам построены OLAP кубы – многомерные наборы значений, отражающие статистику работы пользователей.

Факты легко определить, зная коды событий журнала регистрации и форматы представления записей нужных типов событий. Для построения OLAP куба [8] следует заполнить Data Mart – реляционная база данных определенной (типа «звезда» или «снежинка») структуры, которая состоит из таблицы фактов и таблиц - справочников. Фактами в строящейся Data Mart являются выявленные факты активности в системе. А доступный набор параметров у событий [9] определяет максимальный набор справочников (OLAP измерения).

Так, например, из записей лога безопасности для фактов запуска процессов (куба) выделены измерения: имя запущенного процесса, пользователь, компьютер, родительский процесс и т.д. Введена иерархия компьютеров: Домен – компьютер; иерархия пользователей, иерархия времени. Иерархия в технологии OLAP позволяет получать интересующие нас значения – OLAP меры как для отдельного объекта или явления, так и для их группы, которую допускает иерархия.

Для всех типов событий определены следующие параметры - OLAP Измерения и иерархии:

1. компьютеры (Иерархия: Домен > Компьютер);
2. время (Иерархия: Год > Месяц > Неделя > День > Час > …);

3. процессы (Иерархия: Компьютер > Имя файла);
4. родительские процессы (Иерархия: Компьютер > Имя файла);
5. ресурсы (Иерархия: Компьютер > Путь > Файл);
6. типы ресурсов (без иерархии: расширения файлов и т.д.);
7. типы входов в систему (без иерархии);
8. операции (без иерархии, например, открытие/закрытие дескриптора, стоп/старт процессов, изменение прав и т.д.);
9. пользователи (Иерархия: Домен > Пользователь);
10. статус завершения (без иерархии).

OLAP мерой являются те количественные показатели, которые требуется получить для определенных фактов, выбрав нужные измерения (параметры записи лога) и уровень иерархии. В качестве мер выбрано: количество запусков процессов, продолжительность работы процесса, средняя продолжительность работы процессов.

Обобщение на все рассматриваемые факты приводит к следующим OLAP Мерам:

1. число операций (счетчик);
2. время работы (сумма) – между «парами операций» типа старт/стоп, открытие-закрытие и т.д.;
3. число операций в единицу времени (вычислимое поле).

Например, срез куба для фактов запуска процессов с мерой количества входов, измерениями - пользователи, типы входов в систему и процессы входа, будет выглядеть как на рис. 6.

	Login Types ▾	Processes ▾	
	⊞ Console logon	⊟ Network logon	
		KSecDD	Total
Users ▾	Login Facts Count	Login Facts Count	Login Facts Count
Administrator	17	7	7
alie	2		
ANONYMOUS LOGON		5	5
huws	1		
IUSR_HUME	7		
orionc	3		
triav	3		
Grand Total	33	12	12

Рис. 6. Пример среза для куба входов в систему.

Технология OLAP может быть применена практически к любым типам журналов регистрации ОС и ПО. Например, из лога антивирусной программы можно выделить факты и параметры. Построить и заполнить по ним Data Mart, построить кубы.

Но технология OLAP позволяет строить только статистические отчеты и никак не решает задачи выявления изменений и их характера в поведении системы, приложений и пользователей.

4.2. Data Mining.

Интеллектуальный анализ данных (Data Mining) [11], примененный к собранным записям лог-файлов, позволяет находить скрытые, содержательные и потенциально полезные закономерности, строить ассоциативные правила, находить исключения и аномалии.

В системе используются следующие «Модели поведения».

- Поиск корреляции в атрибутах фактов, которые описывают типичное поведение. Для этого применяется алгоритм на основе ассоциативных правил [10].
- Анализ последовательности действий, выявление часто выполнимых сценариев с целью предсказания и поиска аномалий (на настоящий момент еще не реализовано, планируется использовать методы кластеризации).
- Поиск статистических аномалий (в OLAP срезах) – отклонение значений мер от их усредненных показателей (мат. ожидания, медианы и т.д.).

4.2.1. Ассоциативные правила

Для построения ассоциативных правил, из собранных данных так же как при заполнении Data Mart для OLAP выделяются факты за требуемый временной период. Основная идея подхода заключатся в использовании алгоритмов поиска ассоциативных правил для выявления корреляций между элементами в транзакции, что в нашем случае соответствует корреляциям между значениями атрибутов факта. Пусть любой факт x описывается набором из n атрибутов (характеристик), где область определения атрибутов задана как $x = (x_1, ..., x_n) \in X = dom(x_1) \times ... \times dom(x_n)$ Результатом работы алгоритма является система из m ассоциативных правил $\{R_s(x)\}_{s=1}^{m}$ вида:

$$R_s(x) = "A_{i1}(x), ..., A_{il}(x) \Rightarrow B_{j1}(x), ..., B_{jk}(x)" \qquad [c, s]$$

где:

$A_l(x), B_l(x)$ - предикаты, задающие условия на значения *l-го* атрибута $x_l \in X_l$, в частности, это может быть диапазон значений для числовых атрибутов (например "Duarion = 77-1384"), и конкретные значения для дискретных атрибутов (например, "Process=cmd.exe");

s- поддержка (частота встречаемости) правила, определенная как (число фактов, для которых выполнены все предикаты правила, как в левой так и в правой части):

$$s = \text{support}(R(x)) = \left| \left\{ x \in X \mid A_{i1}(x) \wedge ... \wedge A_{il}(x) \wedge B_{j1}(x) \wedge ... \wedge B_{jk}(x) \right\} \right|$$

c- достоверность правила, определенная как (число фактов, в которых, из выполнения всех предикатов левой части правила, следует выполнение всех предикатов правой):

$$c = \text{confidence}(R(x)) = \frac{\left|\left\{x \in X \mid A_{i1}(x) \wedge \ldots \wedge A_{il}(x) \wedge B_{j1}(x) \wedge \ldots \wedge B_{jk}(x)\right\}\right|}{\left|\left\{x \in X \mid A_{i1}(x) \wedge \ldots \wedge A_{il}(x)\right\}\right|}$$

Ассоциативное правило *R(x)* может быть проинтерпретировано так: «если атрибуты факта x удовлетворяют предикатам $A_{i1},\ldots,A_{il}$, то с вероятностью c данный факт будет удовлетворять предикатам $B_{j1},\ldots,B_{jk}$». Например, первое правило на рис. 7: "User=orionc, Process=cmd.exe=> Duration=77-1384" (probability=1.0) означает, что «если пользователь orionc запустил процесс cmd.exe, то этот процесс всегда проработает не меньше 77 и не больше 1384 секунд». Помимо, простой и эффективной интерпретации самих правил, сильной ассоциативного подхода является простое и эффективное определение «частого эпизода», т.е. часто встречаемой устойчивой комбинации атрибутов. Любой «частый эпизод» также напрямую определяется ассоциативным правилом *R(x)*, где частота определяется как значение поддержки (support) правила.

Построенные ассоциации можно визуализировать либо в виде правил (рис. 7), либо в виде сети зависимостей (рис. 8). На рис. 7 и 8 представлены примеры визуализации для фактов запуска процессов в ОС Windows.

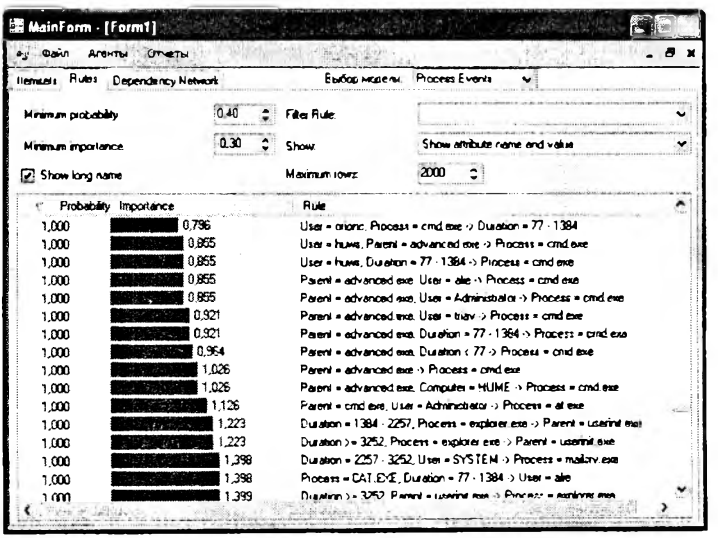


Рис. 7. Визуализация ассоциативных правил.

Каждое правило представимо в виде «если ..., то...» При этом для каждого правила определяется его вероятность. Имеется возможность фильтровать правила по количеству «предпосылок» и по вероятности, а также задавать фильтры по вероятности.

Сеть зависимости позволяет наглядно отобразить вероятные следствия и предшествия пары «атрибут = значение». На данном примере если атрибут Process = User32, то вероятно это будет следовать и впредь из Type = Unlock Workstation, а из этого вероятно будет следовать, что Source = HUME, Duration < 20, User = Administrator.

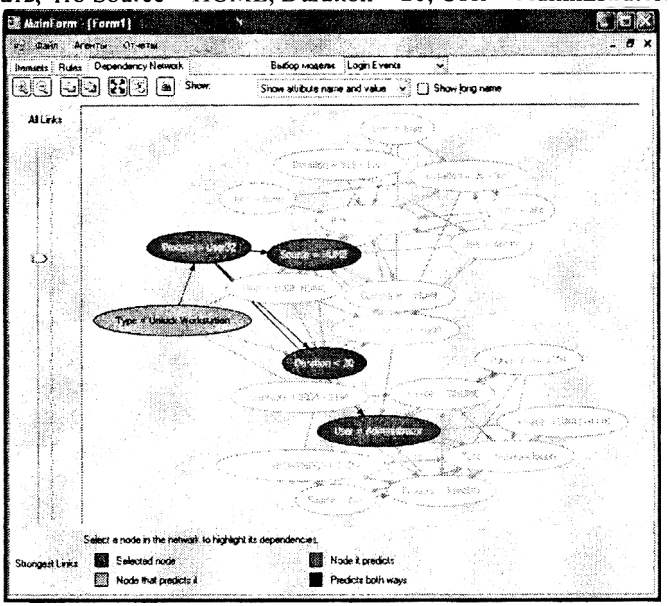


Рис. 8. Визуализация в виде сети зависимостей.

4.2.2. Поиск аномалий

Основная идея применения методов обнаружения аномалий состоит в предположении о том, что активность пользователя системы может быть отслежена и построена ее математическая модель, которая позволит обнаруживать нарушения политики безопасности и аномалии в поведении пользователей. По свидетельству многих специалистов по компьютерной безопасности, такой подход является особенно перспективным в задачах обнаружения именно внутренних вторжений, поскольку он позволяет создавать так называемые системы «раннего обнаружения» (early warning). Опыт эксплуатации систем обнаружения внутренних вторжений показывает, что в большинстве случаев непосредственно внутреннему вторжению предшествует некоторое аномальное (хотя возможно и разрешенное) поведение пользователя,

т.е. пользователь еще до атаки или кражи информации начинает совершать действия не характерные для его предыдущей активности или активности пользователей той же группы.

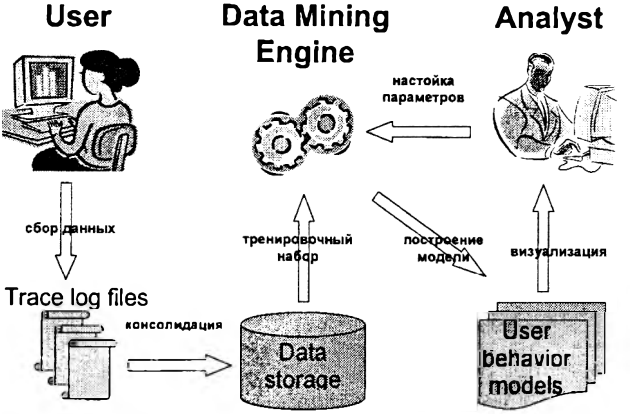


Рис. 9. Схема применения методов Data Mining для построения моделей поведения пользователей.

Из хранилища выбирается тренировочный набор записей журналов регистрации (обычно записи за временной период, в который по мнению аналитика не было вторжений). На выбранном тренировочном наборе «обучается» алгоритм выявления аномалий. Затем алгоритм применяется к вновь поступающим записям, классифицируя их как нормальные или как аномальные. При необходимости аналитик может «дообучать» алгоритм.

Для поиска аномалий можно использовать построенные ассоциативные правила. Для этого сначала алгоритм поиска ассоциативных правил применяется к данным о предыдущей активности пользователей, строится модель на основе ассоциативных правил $\{R_i(x)\}_{i=1}^{m}$, описывающая эту активность. Эта модель применяется к новым данным о текущей активности пользователей и позволяет оценить насколько новая активность отличается от предыдущей, оценить анормальность отдельных событий и их атрибутов. Идея применения модели на основе ассоциативных правил для поиска аномалий базируется на том, что ассоциативные правила $\{R_i(x)\}_{i=1}^{m}$ можно использовать для прогнозирования значений одних атрибутов по другим. Для этого на основе системы правил $\{R_i(x)\}_{i=1}^{m}$ строится функция $P(x_i \mid x_1,...,x_{i-1},x_{i+1},...,x_n)$, которая вычисляет распределение условной вероятности значений *i-го* атрибута, в зависимости от остальных атрибутов.

В простейшем случае такая функция может быть определена следующим образом. Пусть область определения *i-го* атрибута $dom(x_i) = \{a_1,..,a_s\}$ содержит *s* различных значений a_s если *i-й* атрибут дискретный или *s* числовых интервалов, если *i-й* атрибут непрерывный. Тогда вероятность того, что $P(x_i = a_s \mid x_1,...,x_{i-1},x_{i+1},...,,x_n)$ определяется как максимальная достоверность правила :

$$P(x_i = a \mid x_1,...,x_{i-1},x_{i+1},...,,x_n) = \frac{\sum_j \text{confidence}(R_j(x_1,...,x_{i-1},a,x_{i+1},...,,x_n))}{\sum_{s,j} \text{confidence}(R_j(x_1,...,x_{i-1},a_s,x_{i+1},...,,x_n))}$$

В этом случае, уровень достоверности (нормальности) значения *i-го* атрибута как отношение условной вероятности реально наблюдаемого значения a_s к вероятности наиболее ожидаемого значения:

$$Score(x_i \mid x_i = a_s) = \frac{P(x_i = a_s \mid x_1,...,x_{i-1},x_{i+1},...,,x_n)}{\max_l P(x_i = a_l \mid x_1,...,x_{i-1},x_{i+1},...,,x_n)}$$

Очевидно, что если значение *i-го* атрибута совпадает с наиболее ожидаемым, т.е. тем, которое прогнозируется на основе найденной ранее ассоциативной модели, то уровень достоверности такого атрибута равен 1, т.е. абсолютно «ожидаемое» значение. Если же такое значение ранее вообще не встречалось, то его условная вероятность будет равна нулю и уровень достоверности атрибута также будет равен нулю, что соответствует абсолютно «аномальному» значению. В остальных случаях значение достоверности будет меняться от 0 до 1, чем меньше это значение, тем аномальнее значение атрибута. Достоверность всего события *x* в этом случае можно определить как произведение достоверностей его атрибутов:

$$Score(x) = \prod_i Score(x_i)$$

Такой подход дает возможность не только обнаружить аномальные факты (события), но найти причину аномальности, т.е. те атрибуты, которые являются не нормальными с точки зрения предыдущей активности пользователей.

5. Тестирование системы

В 1998 году MIT Lincoln Lab совместно с Defense Advanced Research Projects Agency (DARPA ITO) выпустила методику для оценки алгоритмов обнаружения вторжений под названием DARPA 1998 Intrusion Detection Evaluation Program. Методика содержит набор данных, симулирующих вторжения в локальную сеть. Для получения этих данных была смоделирована типичная локальная сеть,

используемая в военно-воздушных силах США, кроме того, эта сеть была подвергнута множеству различных атак.

Система мониторинга и анализа поведения пользователей была протестирована на эталонном тестовом наборе данных DARPA 1999. Тестовый набор состоит из логов работы операционной системы ОС Windows NT и описания воздействия на систему, результатом которых стали данные логи. Что позволяет оценить вероятность обнаружения атак и вероятность ложного срабатывания систем обнаружения вторжений на данном тесте. При этом определяется и список атак, которые выявляются системой и атаки, которые не выявляются.

5.1. Обнаружение аномалий

Используемый для тестирования набор включает запись журнала регистрации (security log) за 5 недель. При этом были недели без атак, что может использоваться для обучения интеллектуальных систем. В другие же были совершены тестовые атаки. Набор DARPA 99 не содержит записей по работе с файлами и реестром, поэтому при обнаружении вторжений (аномалий) анализировались только факты запуска процессов.

Среди тестовых NT атак в используемом тесте были следующие успешно обнаруженные:

- crashIIS – Одиночный специально сформированный некорректный HTTP запрос, обработка которого приводит к аварийному завершению работы Web сервера.
- yaga – атака, направленная на создание нового пользователя в группе администраторов, путем взлома реестра.
- CascSen – атака, направленная на получение администраторских прав пользователем. Атака использует чувствительность к регистру каталога объектов NT.
- netbus – атака заключается в установке на машине жертвы NetBus сервера. В последствии атакующий может подключаться к серверу удаленно и выполнять практически любые действия от имени работающего в данный момент пользователя.
- netcat – удаленная атака. Атакующий устанавливает программу netcat на 53 порт, которая затем используется как «черный ход», позволяющий заходить на компьютер-жертву, не вводя пароль.

Для решения задачи обнаружения аномалий в результате использования алгоритма поиска ассоциаций были построены ассоциативные правила для периода пользовательской активности, в который не было атак, после чего построенная ассоциативная модель была применена для обнаружения аномалий в периоды пользовательской активности и вторжений.

Для оценки качества работы алгоритма требовалось определить два показателя: *коэффициента обнаружения (detection rate)* и *коэффициента ложных положительных ошибок (false positive rate)*. Обычно сравнение качества работы IDS производится с помощью ROC-кривых (Receiver Operating Characteristic curves). По оси абсцисс графика кривой откладываются значения коэффициента ложных положительных ошибок, по оси ординат – значения коэффициента обнаружения. Каждому результату работы алгоритма на таком графике будет соответствовать одна точка. При этом в зависимости от параметров алгоритма получается множество точек.

При тестировании системы на тестовом наборе DARPA были получены следующие результаты – рис. 10.

Рис. 10. Точность обнаружения атак методом выявления аномалий в эталонном наборе данных DARPA 1999.

Оптимальная настройка соответствует detection rate 87%, false positive 0.5%, экстремальная (все атаки блокированы) - detection rate 100%, false positive 4%.

Обнаруженные/пропущенные вторжения и ложные срабатывания при оптимальной настройке представлены в Таблице 1.

Количество фактов	Атака/Нормальное поведение	Обнаружение/пропуск
12	casesen	detected
7	dos	detected
6	netbus	detected
9	netcat	detected
4	sechole	detected
1	secret	detected
14	yaga	detected
3	casesen	missed
5	dos	missed
2046	normal	allowed
11	normal	blocked

Таблица 1. Обнаруженные и пропущенные вторжения.

Таблица показывает, сколько фактов атак (аномалий) и каких было обнаружено (detected) или пропущено (missed) за время тестирования, сколько было нормальных фактов (allowed) и сколько нормальных фактов было принято за аномалии (blocked) – false positive. Под фактом подразумевался запуск процесса или несколько запусков, если они в совокупности образуют вторжение.

5.2. Характеристики загрузки

Достигнутая производительность системы при работе с тестовыми данными DARPA на каждом этапе отражена в Таблице 2.

Этапы	Количество обрабатываемых событий в секунду
Сбор данных агентом	~1700
Передача данных на ЦМ	~800
Добавление в Хранилище	~400
Заполнение Data Mart	~550

Таблица 2. Скорость обработки данных.

Тестирование производилось на компьютере Pentium4, 2.4ГГц, 512Mb ОЗУ, ОС Windows 2000 Server. Агент, сервер консолидации, хранилище и Data Mart – все элементы системы располагались на одном компьютере. При разнесении агента и сервера консолидации, количество передаваемых данных на ЦМ в единицу времени может быть ограниченно средой передачи.

Литература

1. А.Лукацкий. Обнаружение атак. СПб.: БХВ-Петербург, 2001: с. 50 - 200.
2. Theuns Verwoerd, Ray Hunt. Intrusion Detection Techniques and Approaches // Department of Computer Science University of Canterbury, New Zealand, 2002, с. 2-14.
3. Kathleen A. Jackson. Intrusion detection system (ids) product survey // Distributed Knowledge Systems Team Computer Research and Applications Group Computing, Information, and Communications Division Los Alamos National Laboratory Los Alamos, New Mexico USA, 1999: с. 6-22.
4. Cristina Abadyz, Jed Taylory, Cigdem Senguly, William Yurcik. Log Correlation for Intrusion Detection: A Proof of Concept // Department of Computer Science, University of Illinois at Urbana-Champaign, 2003: с. 3-6.

5. Ли Доддз. XML и базы данных? Доверьтесь своей интуиции. [HTML] (http://www.iso.ru/journal/articles/206.html)
6. Марк Грейвс. Проектирование баз данных на основе XML. М.: Вильямс, 2002: с. 12-70.
7. Даниил Фертф. Где хранить надежду электронной коммерции? // Сетевой. 2001. №1. [HTML] (http://www.setevoi.ru/cgi-bin/text.pl/magazines/2001/1/58)
8. Алексей Федоров, Наталия Елманова. Введение в OLAP. [HTML] (http://olap.ru/basic/OLAP_intro1.asp)
9. Lincoln Laboratory Massachusetts Institute of Technology. The Detectability of Attacks in NT Audit Logs [HTML] (http://www.ll.mit.edu/IST/ideval/docs/2000/ntaudit-table.html)
10. SQL Server 2005 Books Online. Microsoft Association Algorithm [HTML](http://msdn2.microsoft.com/en-us/library/ms174916.aspx)
11. Han J., Kamber M. - Data mining: concepts and techniques., 2000: c. 279-310.

Курынин Р.В., Машечкин И.В., Петровский М.И.

О некоторых методах интеллектуального анализа данных для мониторинга технологических процессов[1]

Введение

Интеллектуальный анализ данных (ИАД, или Data Mining) — междисциплинарная область, базирующаяся на теории и методах баз данных, математической статистике и методах искусственного интеллекта. Основные особенности анализируемых данных, которые необходимо учитывать в задачах Data Mining, — это большой объем, а также разнородность и структурированность. Примером таких данных могут служить временные ряды, реляционные и многомерные данные, содержащие числовые и символьные атрибуты, структурированные системные журналы, протоколы и пр.

В настоящее время Data Mining применяется практически везде, где требуется анализировать большие объемы разнородной структурированной информации, в том числе и на высоко технологичных производствах, в задачах, связанных с анализом технологических процессов (ТП).

Одной из таких задач является задача мониторинга технологического процесса с целью выявления нештатных ситуаций в режиме реального времени. В этой задаче рассматривается изменение состояния модельного ТП во времени. В каждый момент времени состояние ТП описывается вектором характеристик ТП, а поведение ТП за некоторый период времени — многомерным временным рядом. В случае появления необычного значения одной или нескольких характеристик ТП или появления новой тенденции (тренда) в поведении ТП можно предполагать, что произошла нештатная ситуация и необходимо вмешательство оператора. Традиционные методы, основанные на определении диапазонов допустимости для каждой из характеристик, в настоящее время не всегда себя оправдывают, поскольку, во-первых, число анализируемых в реальном времени характеристик может быть очень велико и оператору тяжело за ними уследить, во-вторых, определение допустимых диапазонов также является нетривиальной задачей. Поэтому применение для данной задачи методов Data Mining также является перспективным направлением. С точки зрения Data Mining задача мониторинга с целью

[1] Работа выполнена при поддержке грантов РФФИ 06-01-00691 и НШ 02.445.11.7427.

выявления нештатных ситуаций есть задача обнаружения аномалий (или, в более общепринятой терминологии Data Mining, поиск исключений) во временных рядах в режиме реального времени. Основная проблема поиска аномалий во временных рядах заключается в том, что не существует общего определения аномалии для временного ряда, поскольку аномалией может быть и шумовой выброс, и аномальный тренд. Поэтому обычно для задач обнаружения аномалий выбирается и определение аномальности, и соответствующий алгоритм поиска, исходя из реальных данных о поведении анализируемого ТП.

1.Постановка задачи

Разработка алгоритмов и экспериментальной программной системы, обеспечивающей:

- построение модели закономерностей в многомерных временных рядах значений ТП,
- выявление нештатных ситуаций в поведении ТП на основе построенной модели.

2.Обзор существующих подходов

В рамках обзора существующих решений было проведено патентное исследование, в котором использовались базы данных российских, европейских и американских патентов за последние 20 лет. В ходе анализа особое внимание уделялось патентам, в которых подробно описывалась суть того или иного метода анализа данных и способ его применения. Такой подход позволил достаточно полно охватить предметную область.

Проведенный обзор показал, что в настоящее время существует острая потребность в применении методов интеллектуального анализа данных в технологических процессах.

Вместе с тем, не существует методов, которые бы полностью удовлетворяли запросам потребителей. Поэтому задача дальнейшего исследования и развития методов интеллектуального анализа данных является оправданной.

Для решения поставленной задачи были выбраны следующие алгоритмы Data Mining: одноклассовый метод опорных векторов, регрессионный метод опорных векторов и скрытые Марковские модели.

Одноклассовый метод опорных векторов (SVM — Support Vector Machine) — это класс алгоритмов, который относится к так называемым граничным мстодам, которые определяют классы, используя границы областей. Они совмещают в себе теорию статического обучения и различные способы оптимизации. В простейшем случае эти методы находят гиперплоскость, разделяющую два множества точек и максимизирующую расстояние между

плоскостью и ближайшей точкой одного из множеств. Их плюсом является независимость от размерности пространства и, как следствие, применимость для анализа состояний многопараметрических систем. Более подробную информацию о данном методе можно найти в [1, 2].

Другим алгоритмом выявления нештатных ситуаций во временных рядах является алгоритм, основанный на т.н. **регрессионном методе опорных векторов** (SVR — Support Vector Regression). SVR хорошо применим к моделям временных последовательностей и имеет несколько особенностей:

- полученная функция регрессии может аппроксимировать любую нелинейную зависимость между входным вектором и целевым значением (если задано ядро функции);
- SVR обладает хорошими обобщающими свойствами;
- SVR эффективно работает с данными из пространств с высокой размерностью.

Более подробную информацию о данном методе можно найти в [3 – 5].

Модификация метода авторегрессии SVR для поиска аномалий. Одним из существенных недостатков метода авторегрессии SVR для поиска аномалий является то, что поведение сложной динамической системы, каковой является ТП, как правило, сложно приблизить единственной функцией авторегрессии. В реальности же возможно несколько допустимых последующих состояний. Руководствуясь этой идеей, была предложена модификация метода SVR путем ввода несколько функций авторегрессии, каждая из которых определяет доступное состояние. Это повлекло за собой модификацию всех стадий работы алгоритма, включая обучение, прогнозирование следующего состояния и определение инцидента. Итак, максимальное число допустимых состояний C задается априори, и наш алгоритм строит C функций SVR

$$M^C{}_x(t_0) = f^C(x_D^{t_0}) = \sum_{t=1}^{t_0-1} \theta^C{}_i K(x_D^t, x_D^{t_0}) + b^C \qquad (1)$$

каждая из которых определяет допустимое следующее состояние. Соответственно, изменяется определение коэффициента соответствия $V(t_0) \in R$:

$$V(t_0) = \min_C \left\{ F(M^C{}_x(t_0 - 1), x(t_0)) \right\} \qquad (2)$$

Существенно изменился этап обучения алгоритма. Основная идея предложенной модификации обучения заключается в комбинации подходов четкой кластеризации (разбиение на множества Вороного [6]) с методами построения SVR. Предложенный алгоритм идеологически близок к методу, предложенному в работе [6], хотя применяется не для задачи кластеризации, а для задачи прогнозирования временного ряда.

Алгоритм:

Шаг 0. «Инициализация»

Случайное разбиение тренировочного набора $T_D(t_0) = \{(x'_D, y'), t = D..t_0 - 1\}$, где $x'_D = [x(t-D+1)....x(t)]^T$, $y' = x(t+1)$ на C непересекающихся наборов $T_D(t_0) = \{T^1{}_D(t_0),...,T^C{}_D(t_0)\}$.

Шаг 1. «Обучение»

Для каждого из C тренировочных наборов $T^C{}_D(t_0)$ решается задача построения SVR и строится модель — авторегрессионная машина опорных векторов $M^C{}_x(t_0) = f^C(x'_D) = \sum_{t=1}^{t_0-1} \theta^C{}_t K(x'_D, x^t_D) + b^C$, которая связывается с соответствующим кластером.

Шаг 2. «Перегруппировка»

Исходный тренировочный набор снова перегруппировывается, на этот раз каждый элемент тренировочного набора (x'_D, y') относится к тому кластеру c, для которого минимально значение функции соответствия, связанной с этим кластером, $F(M^C{}_x(x'_D), y')$.

Шаг 3. «Проверка»

Если в результате Шага 2, ни одна точка не поменяла свой кластер, алгоритм сошелся и ВЫХОД, иначе переход на Шаг 1.

Таким образом, предложенный алгоритм можно рассматривать как алгоритм кластеризации временных рядов, но каждый кластер описывается не типичным представителем (эталоном) или вероятностным распределением, а авторегрессионной функцией, порождающей данной кластер. В результате работы алгоритма получается C максимально простых (из-за минимизации границы опорных векторов) и максимально не похожих друг на друга (из-за кластеризации) авторегрессионных функций SVR, которые более точно описывают поведение временных рядов в тренировочном наборе, чем единственная сложная SVR функция авторегрессии. Хотя вопросы сходимости и ее скорости в предложенной модификации еще не исследованы, практика показала, что алгоритм сходится на всех примерах за достаточно небольшое число итераций.

Еще одним методом решения поставленной задачи является метод **скрытых Марковских моделей** (HMM — Hidden Markov Models), детальное описание которого представлено в [7, 8].

3.Предлагаемое решение

Для решения поставленной задачи была разработана программная система, являющаяся частью разработанного комплекса программных средств, предназначенных для встраивания в существующие автоматизированные системы управления (АСУ)

технологическими процессами (ТП) предприятий той или иной индустриальной области с целью проведения анализа и прогнозирования характеристик ТП, а также мониторинга самого ТП. Архитектура предлагаемой подсистемы схематично изображена на рисунке 1.

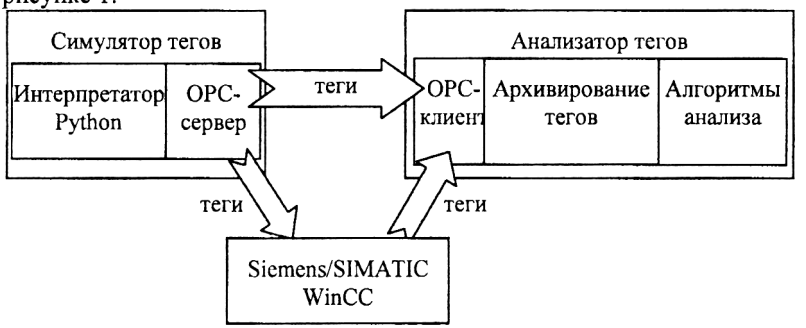


Рисунок 1.

Как видно на рисунке 1, разработанная подсистема способна взаимодействовать с таким продуктом, как Siemens/SIMATIC WinCC, поскольку данный продукт выступает в качестве **системы визуализации процесса** (SCADA), дифференцируемой по цене и производительности и предлагающей наиболее эффективные функции для управления автоматизированными процессами. Одним из главных преимуществ WinCC является ее полная открытость. Эта система может использоваться как со стандартными программами, так и с пользовательскими программами, позволяя создавать человеко-машинные интерфейсы, которые наиболее полно удовлетворяют предъявляемым требованиям. На сегодняшний день WinCC является промышленным стандартом, лидером на европейском рынке и номером 2 на мировом рынке. В силу ориентированности разработанной системы на взаимодействие с WinCC (посредством стандартов OPC, речь о которых пойдет ниже) далее повествование будет вестись именно в терминах WinCC.

Основными компонентами предлагаемого программного решения являются *симулятор тегов* и *анализатор тегов*. Анализатор тегов является главным компонентом системы, его цель — сбор данных, поступающих от того или иного контроллера, участвующего в ТП, при необходимости обеспечение их накопления для проведения обучения того или иного аналитического алгоритма и, собственно, проведение анализа получаемых от контроллеров данных с помощью того или иного реализованного в анализаторе алгоритма выявления аномалий. Для осуществления сбора данных анализатор обращается к некоторой области памяти контроллера, называемой ***тегом***. Этот механизм построен на основе спецификаций **OPC** (OLE for Process Control [9], хотя встречается и такая аббревиатура, как Open Process Control).

В предлагаемом программном продукте используется одна из свободно распространяемых клиент-серверных реализаций OPC, разработанная в Лаборатории сетевых информационных систем Института проблем информатики РАН, — библиотека LightOPC [10]. LightOPC позволяет быстро создавать полноценные и высокопроизводительные серверы OPC для произвольного оборудования, функционирующие в среде Win32. Поддерживаются спецификации OPC/DA v1 и v2 (custom interface). Внутренняя архитектура этого сервера обладает многими достоинствами и позволяет добиться исключительно высокой производительности. Размещение данных оптимизировано для лучшего использования процессорного КЭШа. Синхронизация нитей выполнена в стиле SysV (реализованы блокировки чтения/записи и условия). В сочетании с ограниченным использованием OLE/COM и Win32 API (и неиспользованием ATL) это делает исходный код весьма мобильным.

Поскольку не всегда есть возможность подключения к реальным контроллерам, то в данной системе реализован такой программный компонент, как симулятор тегов. Его главной задачей является генерирование значений тегов во времени. Такой подход позволяет произвести настройку и тестирование анализатора до этапа ввода последнего в эксплуатацию на производстве.

Симулятор тегов — программный компонент, предназначенный для генерации значений тегов. Он позволяет описать на языке программирования Python [11] алгоритмы генерации значений тегов как для нормального режима, так и для аномального (включая считывание значений из файлов), а также примешивать к выдаваемым значениям белый шум с заданной дисперсией.

Анализатор тегов обеспечивает сбор значений тегов, поступающих от того или иного контроллера, участвующего в ТП, при необходимости производит их накопление для проведения обучения одного из реализованных аналитических алгоритмов и, собственно, производит анализ получаемых от контроллеров данных с помощью выбранного алгоритма выявления аномалий.

В анализаторе тегов реализованы описанные выше алгоритмы выявления аномалий: одноклассный метод опорных векторов, регрессионный метод опорных векторов и скрытые Марковские модели. Имеется возможность задавать параметры алгоритмов, производить их обучение и запускать в режиме анализа тегов.

Как уже отмечалось, анализатор построен таким образом, что он может получать данные для анализа как непосредственно с контроллеров (симулятора), так и с любого иного источника, выполняющегося в режиме OPC-сервера. Таким источником может служить и упомянутый выше программный продукт Siemens/SIMATIC WinCC.

4.Экспериментальные исследования

Для проведения экспериментов были использованы несколько временных рядов, входящих в набор данных synthetic_control, полученных из [12]. Этот набор данных представляет собой искусственно сгенерированный временной ряд. Его размерность равна 60. Набор состоит из шести классов временных рядов, каждый из которых состоит из 100 записей (отвечающих, соответственно, 100 точкам времени):

- нормальные данные;
- циклические данные;
- данные с восходящим трендом;
- данные с нисходящим трендом;
- данные с положительным сдвигом;
- данные с отрицательным сдвигом.

Для проведения исследований использовалась следующая методика. Бралась пара временных рядов, один из которых рассматривался как нормальный, а другой — как содержащий ту или иную аномалию. После обучения каждого из алгоритмов проводились замеры меры аномальности для нормального и аномального наборов. Для максимизации степени выявления аномалий в аномальном наборе и минимизации таковой в нормальном проводилась ручная корректировка параметров алгоритмов.

Эксперимент 1. Наборы данных: нормальные/циклические.

Таблица 1 — Параметры алгоритма SVM

anomaly_low	GAMMA	length	LENGTH	OUTLIERS
от -0.35 до -0.27 с шагом 0.2	1.6	8	5	0.2

Таблица 2 — Параметры алгоритма HMM

anomaly_low	length	states_num
от -350.0 до -150.0 с шагом 20.0	5	5

Таблица 3 — Параметры алгоритма SVR

anomaly_high	GAMMA	length	LENGTH	C	EPS	STATES	FUZZIFIER
от 8.0 до 22.0 с шагом 2.0	10	5	5	1	0.01	5	0.5

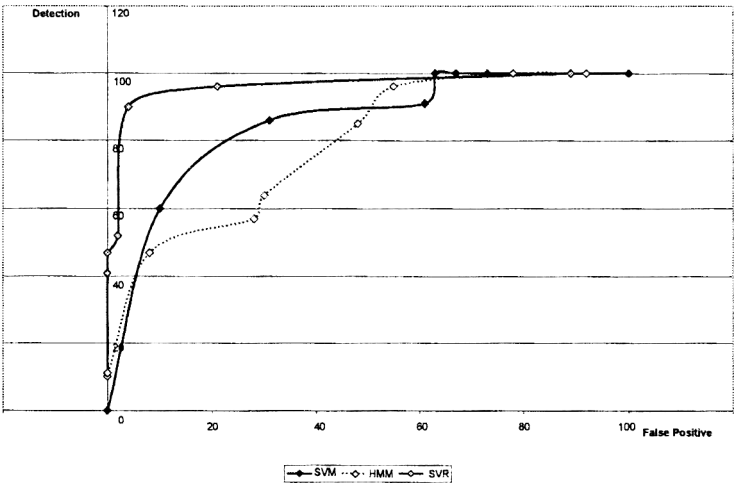


Рисунок 2. Наборы данных: нормальные/циклические.

Эксперимент 2. Наборы данных: нормальные/с восходящим трендом; нормальные/с нисходящим)трендом.

Таблица 4 — Параметры алгоритма SVM

anomaly_low	GAMMA	length	LENGTH	OUTLIERS
от -0.35 до -0.27 с шагом 0.2	1.1	8	5	0.2

Таблица 5 — Параметры алгоритма HMM

anomaly_low	length	states_num
от -350.0 до -150.0 с шагом 20.0	5	5

Таблица 6 — Параметры алгоритма SVR

anomaly_high	GAMMA	length	LENGTH	C	EPS	STATES	FUZZIFIER
от 8.0 до 22.0 с шагом 2.0	10	5	5	1	0.01	5	0.5

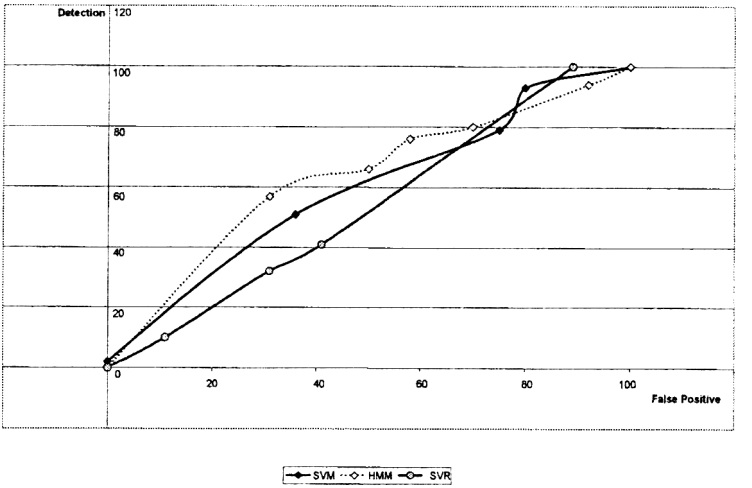


Рисунок 3. Наборы данных: нормальные/с восходящим трендом.

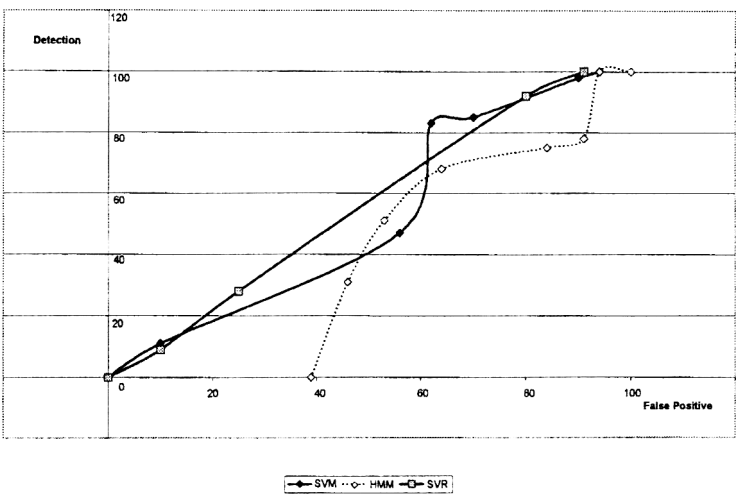


Рисунок 4. Наборы данных: нормальные/с нисходящим трендом.

Эксперимент 3. Наборы данных: нормальные/с положительным (отрицательным) сдвигом.

Таблица 7 — Параметры алгоритма SVM

anomaly low	GAMMA	length	LENGTH	OUTLIERS
от -0.35 до -0.27 с шагом 0.2	1.6	8	5	0.2

Таблица 8 — Параметры алгоритма HMM

anomaly_low	length	states_num
от -350.0 до -150.0 с шагом 20.0	5	5

Таблица 9 — Параметры алгоритма SVR

anomaly_high	GAMMA	length	LENGTH	C	EPS	STATES	FUZZIFIER
от 8.0 до 16.0 с шагом 1.0	10	5	5	1	0.01	5	0.5

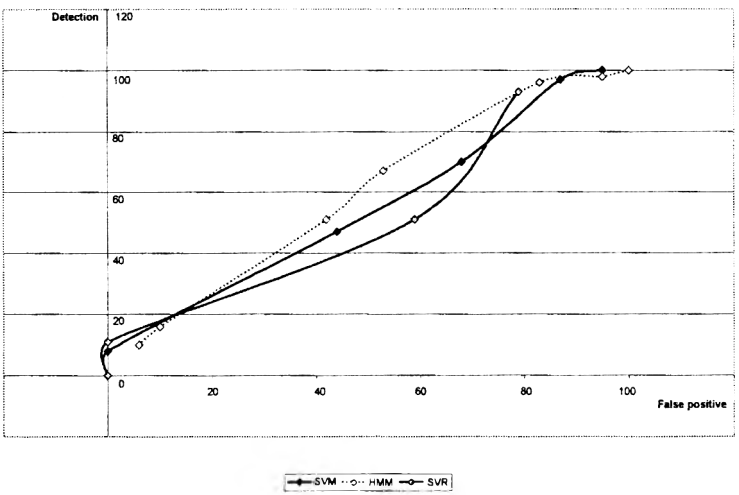


Рисунок 5. Наборы данных: нормальные/с положительным (отрицательным) сдвигом.

Заключение

Проведенные эксперименты показали, что реализованные в системе методы интеллектуального анализа данных (метод опорных векторов, скрытые Марковские модели и модифицированный метод авторегрессии) способны решать задачу по выявлению аномалий во временных рядах в ходе функционирования технологического процесса. Наилучшие результаты были достигнуты на наборе с циклическими аномалиями. Для аномальных наборов с восходящим и нисходящим трендами и со сдвигами результаты экспериментов оказались хуже, что объясняется более сложной структурой аномалий и неоптимальностью

значений используемых параметров. Подбор оптимальных параметров требует дополнительных исследований.

Литература

1. Junshui Ma, Simon Perkins. Time-series Novelty Detection Using One-class Support Vector Machines. IJCNN, 08.03.2003.
2. N.H. Packard, J.P. Crutchfield, J.D. Farmer, and R.S. Shaw. Geometry from a Time Series, Physical Review Letters, vol. 45, pp. 712-716, 1980.
3. Smola, A. J., and B. Schölkopf (1998). A Tutorial on Support Vector Regression, NeuroCOLT Technical Report NC-TR-98-030, Royal Holloway College, University of London, UK.
4. Junshui Ma, Simon Perkins. Online Novelty Detection on Temporal Sequences. SIGKDD '03, August 24-27, 2003, Washington, DC, USA.
5. Junshui Ma, James Theiler, Simon Perkins. Accurate Online Support Vector Regression, to appear in Neural Computation, 2003.
6. Francesco Camastra. A Novel Kernel Method for Clustering, IEEE Transactions On Pattern Analysis And Machine Intelligence, Vol. 27, No. 5, P.801–805, May 2005.
7. Valery A. Petrushin. Hidden Markov Models: Fundamentals and Applications. Part 1: Markov Chains and Mixture Models. Online Symposium for Electronics Engineer 2000.
8. Valery A. Petrushin. Hidden Markov Models: Fundamentals and Applications. Part 2: Discrete and Continuous Hidden Markov Models. Online Symposium for Electronics Engineer 2000.
9. The OPC Foundation [Электронный ресурс]. — http://www.opcfoundation.org.
10. Light OPC. Библиотека для разработки OPC серверов [Электронный ресурс]. — http://www.ipi.ac.ru/lab43/lopc-ru.html.
11. The Python Programming Language [Электронный ресурс]. — http://www.python.org.
12. UCI Knowledge Discovery in Databases Archive [Электронный ресурс]. — http://kdd.ics.uci.edu.

Раздел III
Имитационное моделирование

Смелянский Р.Л., Шалимов А.В.

Метод оценки частот выполнения фрагментов кода последовательной программы

Введение

Необходимость определения редко выполняемых частей программы возникает во многих прикладных задачах:

- при оптимизации и распараллеливании программ, для того чтобы сконцентрировать усилия именно на активных участках программ.
- при разработке для операционных систем более эффективных стратегий управления и распределения ресурсов.
- при выборе резидентной части в системах реального времени.
- При построении оптимизирующих компиляторов. Затраты на компиляцию можно будет сократить, если компилятор будет оптимизировать не всю программу, а лишь активные ее части.
- распределении нагрузки в распределенных и многопроцессорных системах.
- для выделения наиболее интенсивно используемых фрагментов кода программы (везде далее просто фрагментов, если не оговорено противное), которые целесообразно подвергать наиболее тщательной проверке при тестировании.

Задача определения редко выполняемых фрагментов программы разбивается на две подзадачи:

- разметка исходной программы на фрагменты (здесь под фрагментов мы будем иметь ввиду линейные участки программы) и определение частот их выполнения.
- определение редко выполняемых фрагментов программы.

Рассмотрим эти задачи подробнее.

1. Определение средних частот выполнения фрагментов программы на основе функций распределения ее входных параметров

В этом разделе мы опишем модель последовательной программы для получения частотных характеристик исполнения ее фрагментов.

Постановка задачи

Пусть дан текст программы, для каждого входного параметра этой программы известен диапазон его значений и функция распределения значений на этом диапазоне. Требуется определить средние частоты выполнения каждого фрагмента, т.е. линейного участка исследуемой программы.

Описание модели программы

Модель состоит из двух частей: статической (специальное представление программы) и динамической (моделирующей выполнение программы).

Программа представляется в виде множества функций и множества входных параметров. Функция – это множество базовых блоков. Базовый блок – это пара (линейный участок, функция перехода). Линейный участок – это последовательность операторов, не содержащая операторов управления. Функция перехода определяет условие перехода от одного базового блока к другому.

Входной параметр программы - это переменная исследуемой программы, через которую в программу передают данные извне. Назовем множеством состояний памяти программы декартово произведение множества значений переменных программы. Состоянием модели программы назовем элемент множества состояний памяти программы. Оператор программы будем представлять как отображение из множества состояний памяти в множество состояний памяти одной и той же программы. Функция перехода - это отображение из множества состояний памяти программы в множество базовых блоков программы.

Алгоритм функционирования модели

Функционирование модели состоит из двух этапов.

1. Формирование начального состояния модели. Происходит инициализация входных праметров программы значениями, сгенерированными по закону распределения их значений согласно [3,4].
2. Выполнение модели состоит из последовательности шагов. Каждый шаг состоит из следующих действий:

1) Выполнение всех операторов базового блока над текущим состоянием модели программы. Происходит последовательное выполнение всех операторов базового блока. Заметим, что если встретился оператор чтения входного параметра из внешнего источника, то этому параметру присваивается значение, сгенерированное по закону распределения для значений данного входного параметра.
2) Выбор преемника. Происходит вычисление функции перехода базового блока над состоянием программы, в результате определяется базовый блок, который становится преемником. Если преемник не найден, то текущий базовый блок считается заключительным и выполнение модели считается завершенным.

3) Передача управления преемнику. Передача текущего состояния модели программы преемнику.

Определение средних частот

При исполнении модели программы можно подсчитать сколько раз управление передавалось на каждый базовый блок модели. Для этого каждому базовому блоку сопоставлен счетчик. Этот счетчик увеличивается на единицу каждый раз, когда происходит переход управления от базового блока. После окончания функционирования модели счетчики базовых блоков будут содержать искомые величины.

Произведем N запусков модели программы. Получим N значений счетчиков. Заметим, что результаты каждого запуска будут отличаться от результатов предыдущего, поскольку каждый раз при функционировании модели генерируются новые значения входных параметров. Рассмотрим среднее арифметическое значений каждого счетчика базового блока модели, т.е.

$$F_j = \frac{1}{N} \times \sum_{i=1}^{N} f_j^i \qquad (1),$$

где f_j^i значение счетчика j-го базового блока после i-го запуска модели. Покажем, что указанная величина является средней частотой базового блока. Для этого докажем следующее утверждение.

Утверждение. Величина (1), равная сумме значений всех счетчиков базового блока на каждой итерации, деленная на N, является средней частотой выполнения базового блока, и эта величина стремится

к истинной частоте выполнения базового блока при стремлении числа запусков модели к бесконечности.

Доказательство.

Рассмотрим случайную величину $\xi_j^n = I(E_j^n)$, где E_j^n – событие, что на n-ом шаге выполнения модели программы будет выполняться j-тый базовый блок. Тогда средняя частота выполнения j-го базового блока после завершения работы модели будет равна $F_j = \sum_{n=1}^{M} \mathrm{m}(\mathrm{I}(\mathrm{E}_j^n))$, где M – число шагов, которое сделала модель в процессе работы (предполагается, что модель завершает свою работу за конечное число шагов).

Случайная величина ξ_j^n имеет вид $\xi_j^n = \mathrm{f}(x_1, \ldots, x_v) = \mathrm{f}(B_1(x_1, \ldots, x_v), \ldots, B_b(x_1, \ldots, x_v))$, где $\mathbf{x_i}$ – случайная величина, сопоставленная входному параметру исследуемой программы, **v** – число входных параметров, $\mathbf{B_j}$ – функция базового блока, которая отражает в себе суперпозицию операторов этого блока, функцию перехода и распределения значений входных параметров программы, **b** – число базовых блоков программы. В общем случае найти распределение случайной величины ξ_j^n аналитически невозможно.

Для оценки значения математического ожидания случайной величины ξ_j^n воспользуемся методом статистических испытаний (метод Монте-Карло) [3,4]. Этот метод заключается в том, что со случайной величиной проводят опыты, получают значения случайной величины и на их основе рассчитывают статистические характеристики случайной величины. Статистическим аналогом математического ожидания случайной величины является среднее арифметическое ранее наблюдаемых значений случайной величины: $m_\xi^* = \frac{1}{N} \times \sum_{i=1}^{N} \xi_{ji}^n$, где ξ_{ji}^n – значение случайной величины ξ_j^n, наблюдаемое в i-том опыте, N – число опытов. Тогда приближенное значение средней частоты выполнения j-го базового блока будет равна $F_j^* = \sum_{n=1}^{M} 1/N \times \sum_{i=1}^{N} \xi_{\mathrm{ji}}^{\mathrm{n}} = \frac{1}{N} \times \sum_{i=1}^{N} \sum_{n=1}^{M} \xi_{ji}^n$. Величина $\sum_{n=1}^{M} \xi_{ji}^n$ равна значению счетчика базового блока после i-го запуска модели программы. Таким образом, величина F_j^* совпадает с величиной (1). Следовательно, величина (1) является средней частотой выполнения базового блока.

Введем новые обозначения $\xi^* = m_\xi^*$. Из теории математической статистики известно, что $m_{\xi^*} = m_\xi$, т.е. в среднем значение статистического аналога математического ожидания случайной величины будет совпадать с самим математическим ожиданием случайной величины [4]. Так же известно, что $\sigma^2_{\xi^*} = \sigma^2_\xi / N$ [4].

Рассмотрим $P(|\xi^* - m_\xi| < \varepsilon) = \beta$, где β – некоторая вероятность того, что значение случайной величины ξ^* будет отклоняться от m_ξ на величину, не превосходящую ε. Покажем, что $\varepsilon \to 0$, при $N \to \infty$.

В силу центральной предельной теоремы случайная величина ξ^* асимптотически нормальна. Поэтому при большом числе N при установлении зависимости между ε, β и N можно воспользоваться стандартным нормальным законом распределения.

$$P\left(\frac{|\xi^* - m_\xi|}{\sigma_{\xi^*}} < t_\beta\right) = \frac{2}{\sqrt{\pi}} \int_0^{t_\beta} e^{-x^2} dx = \beta$$

$$P\left(|\xi^* - m_\xi| < t_\beta * \sigma_{\xi^*}\right) = \beta$$

$t_\beta * \sigma_{\xi^*} = \varepsilon \quad \Rightarrow \quad \varepsilon = \dfrac{\sigma_\xi^* * t_\beta}{\sqrt{N}} \quad \Rightarrow \quad$ **$\varepsilon \to 0$, при $N \to \infty$.**

Аналогично, для $P(|F^*_j - F_j| < \varepsilon_1) = \beta_1 \quad \Rightarrow \quad$ **$\varepsilon_1 \to 0$, при $N \to \infty$.** Таким образом получается, что чем больше число запусков модели программы, тем более точным получится результат. Доказательство окончено.

Таким образом, чтобы определить средние частоты базовых блоков, необходимо как можно больше раз запустить модель программы и вычислить средние частоты по формуле (1).

Заметим, что аналогичным способом можно определить средние значения различных характеристик программы. Например, количество переходов между базовыми блоками, среднее число шагов работы программы и т.п.

Важно отметить, что предлагаемая техника позволяет избежать исследования вопроса об условных и безусловных вероятностях переходов между базовыми блоками. Обычно в работах, где исследовались частотные характеристики программ, явно или не явно использовалось предположение, что вероятность перехода к некоторому базовому блоку не зависит от того, по какому пути мы пришли к нему. Такое предположение является весьма спорным. Предложенная здесь техника учитывает зависимости в программе по управлению и их влияние на значения вычисляемых частот.

Практическое применение метода

На основе описанного выше метода была построена экспериментальная система для определения средних частот выполнения линейных участков программ, называемая Frequency System. Входными данными системы служат текст программы на языке Си и функции распределения входных параметров программы по диапазонам их значений. В качестве результата работы системы можно получить средние частоты выполнения базовых блоков программы.

Покажем достоинства разработанной системы по сравнению со стандартными профилировщиками компиляторов llvm (www.llvm.org), gcc (gcc.gnu.org). Для этого сначала опишем принцип их работы. Исследуемая программа модифицируется добавлением в начало каждого базового блока команды увеличения счетчика базового блока на 1. Измененная программа запускается и начинает свое выполнение, как обычно. В результате выполнения измененной программы генерируется файл профилировки. В этом файле находится информация о числе выполнений каждого базового блока программы при данном запуске программы.

Преимуществами описываемой системы Frequency system по сравнению со стандартными профилировщиками являются:

1. Для получения средних частот выполнения базовых блоков программы при использовании профилировщиков пользователю необходимо N ($N \to \infty$) раз запустить программу, ввести необходимые входные данные, получить результат. При использовании FrequencySystem пользователю необходимо лишь один раз задать распределения значений входных данных программы, и FrequencySystem сама в нужный момент будет "вводить" значения, сгенерированные по заданным распределениям.

2. При использовании стандартных профилировщиков для получения средних частот выполнения базовых блоков программы необходимо, чтобы программа была исполняемой, поэтому исходная программа не должна содержать неразрешенных ссылок на внешние переменные. Это ограничение отсутствует при использовании разработанной системы: наличие неразрешенных внешних переменных возможно, необходимо лишь задать распределения их значений.

3. При использовании стандартных профилировщиков для получения средних частот выполнения базовых блоков программы нельзя проанализировать отдельную функцию программы без запуска всей программы. Разработанная система позволяет выполнить анализ отдельно взятой функции, для этого пользователю нужно лишь выбрать распределение значений для предлагаемых программой параметров: формальных аргументов анализируемой функции и используемых данной фукцией глобальных переменных.

Недостатками системы FrequencySystem по сравнению со стандартными профилировщиками являются:

1. Невозможность определения средних частот выполнения базовых блоков программ, которые имеют специфический формат входных данных, т.е. входные данные, для которых нельзя задать правильное распределение (например, jpeg-файлы, html-файлы).

2. Небольшая скорость работы, т.к. происходит выполнение модели программы, а не самой программы.

2. Определение редко выполняемых частей кода программы

Определение редко выполняемого кода начнем с выбора управляющего параметра θ, $0 \leq \theta < 1$, который будем называть **порогом**. Порог задает максимальную долю редко выполняемого кода программы в общем количестве выполняемых команд программы. Пользователь, задавая порог, указывает, что код, доля выполняемых команд которого не превышает порога, является редко выполняемым. Базовые блоки, кандидаты на включение в редко выполняемый код, должны выбираться с учетом этого параметра, т.е. доля выполняемых команд этих блоков в общем числе выполняемых команд не должна превышать порога.

Под **весом** базового блока будем понимать число команд в базовом блоке, умноженное на частоту выполнения этого блока. Вес задает вклад этого базового блока в общее количество выполненных команд исследуемой программы. Вес обозначается **weight**(b) = **size**(b) * **freq**(b), где size(b) и freq(b) обозначают размер (количество команд) и частоту выполнения базового блока соответственно.

Рассмотрим следующие методы определения редко выполняемого кода.

1. **Весовой метод.** Этот метод основывается на *весе* базового блока, т.е. при определении, является ли базовый блок редко выполняемым, смотрим на вклад числа выполняемых команд базового блока в общее число выполняемых команд программы. Базовый блок называется *редко выполняемым*, если его вес вместе с весом всех более легких базовых блоков не превышает θ * T, где T - общее число выполняемых команд исследуемой программы.

Этот метод опирается на идеи, описанные в [7].

С точки зрения весового метода, блок называется редко выполняемым, если его вклад в общее количество выполненных команд программы не превышает определенного числа выполненных команд. Это количество определяется исходя из порога θ.

2. **Частотный метод.** Базовый блок называется *редко выполняемым*, если его частота выполнения не превышает некоторой

предельной частоты выполнения. Предельная частота - это наибольшая частота (N) выполнения базового блока исследуемой программы, для которой выполнено следующее условие: сумма весов блоков, частота выполнения которых не превышает N, должна быть меньше θ * T: $\sum_{b:freq(b)\le N} weight(b) \le \theta * T$, где T - общее число выполняемых команд программы.

Этот метод опирается на идеи, описанные в [8].

С точки зрения весового метода, блок называется редко выполняемым, если его частота выполнения не превышает определенной частоты. Эта частота определяется исходя из порога θ.

3. **Частотно-весовой метод.** В этой работе предложен новый метод, который является *комбинацией* частотного и весового методов. При использовании частотного метода остается неизрасходованным максимально допустимое количество редко выполняемого кода программы. Заметим, что неизрасходованная часть появилась из-за того, что вес уже определенных как редко выполняемых базовых блоков вместе с весом всех блоков с частотой, следующей за предельной частотой в упорядоченном по возрастанию списке частот базовых блоков (будем называть такую частоту 'запредельной'), превысил допустимое количество редко выполняемого кода. Неизрасходованная величина кода равна $\theta * T - \sum_{b:freq(b)\le N} weight(b)$. Идея частотно-весового метода заключается в использовании этой неизрасходованной части, а именно в распределении неизрасходованной части между базовыми блоками с 'запредельной' частотой. Способов выбрать из некоторого множества элементы так, чтобы удовлетворить заданным ограничениям, достаточно много. Рассмотрим следующие пошаговые способы выбора базовых блоков: а) на каждом шаге выбираем максимальный по весу базовый блок, который помещается в неизрасходованную часть; б) на каждом шаге выбирать минимальный по весу базовый блок. На основе этих способов рассмотрим два варианта частотно-весового метода: частотно-весовой метод с понижением веса и частотно-весовой метод с повышением веса.

3.1. **Частотно-весовой метод с понижением веса.** Базовый блок называется *редко выполняемым*, если выполнено одно из следующих условий:

a) Частота выполнения базового блока не превышает предельной частоты (N). Понятие предельной частоты аналогично понятию предельной частоты в частотном методе.

b) Частота выполнения базового блока равна частоте следующей за предельной частотой (в упорядоченном по

возрастанию списке частот), и вес базового блока вместе с весом *всех более тяжелых редко выполняемых базовых блоков* не более величины $\theta * T - \sum_{b:\ freq(b) \le N} weight(b)$.

3.2. **Частотно-весовой метод с повышением веса.** Базовый блок называется *редко выполняемым*, если выполнено одно из следующих условий:

a) Частота выполнения базового блока не превышает предельной частоты. Понятие предельной частоты аналогично понятию предельной частоты в частотном методе.

b) Частота выполнения базового блока равна частоте следующей за предельной частотой (в упорядоченном по возрастанию списке частот), и вес базового блока вместе с весом *всех более легких базовых блоков* не более величины $\theta * T - \sum_{b:\ freq(b) \le N} weight(b)$.

Покажем, как выбирать подходящий метод определения редко выполняемого кода программы на примере системы компактного представления программы, основанной на использовании схемы компрессор/декомпрессор [7]. Основная идея этой системы заключается в том, что редко выполняемые фрагменты кода программы хранятся в неисполняемой (сжатой) форме и динамически, по мере необходимости, распаковываются в исполняемую форму.

В такой системе необходимо, чтобы процесс распаковки происходил как можно реже для заданного количества редко выполняемого кода. Исходя из этого требования, для применения в системе компактного представления программы весовой метод не подходит, т.к. в результирующий набор базовых блоков может попасть маленький базовый блок с большой частотой выполнения.

Для заданного порога хотелось бы получить максимально возможное количество редко выполняемого кода. При использовании же частотного метода остается неизрасходованным допустимое количество редко выполняемого кода.

Минимальной единицей сжатия в указанной системе компактного представления программы является базовый блок. При использовании частотно-весового метода с понижением веса в первую очередь выбираются базовые блоки с большим числом команд, такие базовые блоки выгоднее использовать для увеличения коэффициента сжатия программы, так как коэффициент сжатия при работе компрессора больше для большего объема данных. При использовании частотно-весового метода с повышением веса в первую очередь выбираются базовые блоки с меньшим числом команд. Такие базовые блоки выгоднее использовать для повышения скорости выполнения программы, так как, во-первых, этот базовый блок может быть

невыгоден для сжатия (следовательно, не будет сжат) и, во-вторых, декомпрессор работает быстрее для меньшего объема команд.

Таким образом, если целью является уменьшение времени выполнения сжатой программы, то в системе компактного представления программы следует выбирать частотно-весовой метод с повышением веса, а если цель – получить меньший размер сжатой программы, то следует выбрать частотно-весовой метод с понижением веса.

Заключение

В данной работе описан новый подход к определению средних частот выполнения фрагментов кода последовательной программы на основе функций распределения входных параметров. Важным достоинством предлагаемого подхода является то, что он позволяет избежать исследования вопроса об условных и безусловных вероятностях переходов между базовыми блоками программы. Обычно в работах, где исследовались частотные характеристики программ, явно или не явно использовалось предположение, что вероятности перехода к базовым блокам не зависят от того, по какому пути мы пришли к ним. Такое предположение является весьма спорным. Предложенная здесь техника учитывает зависимости в программе по управлению и их влияние на значения вычисляемых частот.

Показаны преимущества такого подхода по сравнению с существующими методами определения частот. Изложены и предложены новые методы определения редко выполняемого кода программы. Предложенные методы позволяет более тонкую настройку под задачу.

Описанные выше методы применяются в разрабатываемой авторами системе компактного представления программ для встроенных систем и показывают хорошие результаты работы.

Литература

1. Смелянский Р.Л., Гурьев Д.Е., Бахмуров А.Г. Об одной математической модели для расчета динамических характеристик программы. Программирование, N6, 1984
2. R.L. Smelianski, T. Alanko. On the calculation of control transition probabilities in a program Inform.Processing Letters N.3, 1986
3. Гмурман В.Е. Теория вероятности и математической статистики - 9е издание. М.:Наука, 2003., 363с.

4. Скрипкин В.А., Моисеенко Е.А. Математические методы исследования операций в военном деле. М.:Военное издательство министерства обороны СССР, 1979.
5. Documentation for the LLVM System, [HTML] (http://llvm.cs.uiuc.edu/docs/).
6. Susan L., Graham Peter B., Kessler Marshall K. McKusick. gprof: a Call Graph Execution Prifiler. // Symposium on Compiler Construction, Proceedings of the 1982 SIGPLAN symposium on Compiler construction, Boston, Massachusetts, United States. pp. 120 – 126
7. Saumya Debray и William S. Evans. COLD CODE Decompression at Runtime. // Communications of the ACM August 2003, *Vol. 46, No.8, 55-60.*
8. Saumya Debray и William S. Evans. Profile-Guided Code Compression. [PDF] (www.acm.org).

Балашов В.В., Ларионов А.А.

Верификация имитационных моделей в среде ДИАНА с применением средства SPIN[1]

Введение

Для современных распределённых программно-аппаратных систем актуальна задача обоснования корректности их работы. Тестирование таких систем часто не даёт удовлетворительных результатов, поскольку два вычисления распределенной программы на одних и тех же исходных данных способны, вообще говоря, привести к различным результатам. Альтернативой тестированию является формальная проверка (верификация) правильности программ. При выполнении верификации, требования к правильности функционирования системы формулируются на специальном языке, и проверка соответствия программы этим требованиям осуществляется автоматически. В рамках подхода, называемого *проверкой на модели* (model checking), строится модель системы, и на ней проверяются формально описанные требования (свойства системы) [2]. Распространёнными языками описания свойств являются темпоральные логики LTL и CTL [3], а также языки, спецификации на которых транслируются в формулы этих логик.

В рамках среды имитационного моделирования ДИАНА [5], в 1998-2000 гг. была разработана система верификации распределённых имитационных моделей, описанных на языке ММ [4]. Система позволяла проверять выполнимость на имитационных моделях свойств, заданных на языке MMSpec, методом проверки на модели. Перед выполнением верификации, модели автоматически транслировались в промежуточный язык MDL, а описания (спецификации) свойств – в формулы логики CTL или LTL. Процесс верификации выполнялся полностью автоматически.

Система верификации была применена в рамках международного проекта DrTesy [6] для проверки свойств модели бортового авиационного вычислительного комплекса. При этом часть свойств проверить не удалось в связи тем, что система верификации имела недостаточную производительность и требовала большого количества вычислительных ресурсов. Для обеспечения возможности практического применения системы верификации в рамках среды ДИАНА было принято решение о необходимости замены существующего ядро системы верификации [4] на более производительное.

[1] Работа выполнена при частичной поддержке РФФИ, грант N 04-01-00556.

Были выдвинуты следующие требования к программному средству-верификатору, которое должно заменить существующее ядро системы верификации: 1) высокая производительность; 2) положительный практический опыт применения при верификации распределённых программных и программно-аппаратных систем; 3) наличие входного языка описания моделей, на который может быть корректно отображены описания моделей на языке MDL[2]; 4) поддержка описания проверяемых свойств в виде формул логики LTL или CTL; 5) доступность в исходных кодах по свободной лицензии (open source); 6) наличие активного сообщества разработчиков.

По перечисленным выше критериям был выбран верификатор SPIN [8, 9]. В данной статье приведены результаты работ по разработке ядра системы верификации в среде ДИАНА на основе SPIN. Раздел 1 содержит описание архитектуры исходной системы верификации. В разделе 2 приведены основные особенности верификатора SPIN. В разделе 3 приведена схема интеграции SPIN в систему верификации среды ДИАНА и основные принципы отображения языка MDL на PROMELA, входной язык SPIN. Раздел 4 содержит результаты экспериментального исследования системы верификации, построенной на основе SPIN.

1. Структура и возможности системы верификации в среде ДИАНА

На рисунке 1 показана структура системы верификации в среде ДИАНА на момент применения в проекте DrTesy [6] Описанная на языке моделирования ММ среды ДИАНА [5] имитационная модель (далее – ММ-программа) состоит из процессов, взаимодействующих посредством обмена сообщениями. Проверяемое свойство описывается в виде спецификации на языке MMSpec. При верификации ММ-программы, она транслируется в промежуточное представление на языке MDL (далее – MDL-программа). Процессам ММ-программы при этом сопоставляются описанные на языке MDL недетерминированные конечные параметризованные автоматы [4]. ММ-программе как совокупности взаимодействующих процессов сопоставляется композиция автоматов, взаимодействующих через общие переменные. По спецификации на языке MMSpec строится формула темпоральной логики CTL (поддерживается также логика LTL). MDL-программа и CTL-формула поступают на вход ядру верификации, которое осуществляет проверку выполненности формулы на модели программы. В версии системы верификации, применявшейся в проекте DrTesy, проверка выполнялась на основе механизма двоичных решающих диаграмм (BDD).

[2] Понятие корректности отображения определено в разделе 3.

Рисунок 1. Схема системы верификации среды ДИАНА.

Возможности системы верификации в среде ДИАНА ограничиваются проверкой свойств, относящихся к классу *безопасность* (safety) [1]. Эти свойства соответствуют требованиям недостижимости системой нежелательного состояния. В силу наличия существенных абстракций при переходе от MM-программы к MDL-программе, невозможна проверка системой свойств из класса *живучесть* (liveness) [1], которые соответствуют требованиям достижимости требуемого состояния.

Описание программы на языке MDL состоит из трех основных частей: 1) описание глобальных переменных; 2) описание параметризованных автоматов, в т.ч. их формальных параметров; 3) Определение «экземпляров» автоматов с указанием их фактических параметров-переменных, в т.ч. набора разделяемых между автоматами переменных.

Все переменные на MDL имеют целочисленный тип с явно задаваемым диапазоном значений.

Описание автомата на языке MDL включает: 1) заголовок с указанием формальных параметров; 2) тело, включающее набор описаний переходов. Для каждого перехода заданы: начальное состояние; условие осуществимости перехода; конечное состояние после перехода; действия над переменными при переходе.

2. Верификатор SPIN

Верификатор SPIN [8, 9] изначально был разработан Ж. Гольцманом и Д. Пеледом в Bell Labs. В настоящее время SPIN является проектом с открытым исходным кодом, распространяется по свободной лицензии и поддерживается сообществом разработчиков. SPIN успешно применялся в ряде проектов NASA [10], Lucent Technologies [11] и других организаций [9]. В рамках проектов NASA выполнялась разработка и верификация сложных распределённых бортовых вычислительных систем для космических аппаратов.

Входной язык SPIN, PROMELA, имеет С-подобный синтаксис; спецификации свойств задаются в логике линейного времени LTL. Программа на PROMELA включает: 1) описание глобальных переменных; 2) описание прототипов процессов; 3) описание каналов, по которым передаются сообщения между процессами. Средствами PROMELA может быть описано как взаимодействие процессов через передачу сообщений, так и взаимодействие через общие переменные. Подмножество PROMELA позволяет описать процесс в виде недетерминированного конечного автомата.

Наличие в PROMELA перечисленных выше синтаксических конструкций и возможностей делает возможным трансляцию MDL-программ в программы на PROMELA. Принципы отображения, лежащего в основе этой трансляции, а также обоснование корректности отображения, приведены в разделе 3.

В SPIN реализован ряд эффективных алгоритмов сокращения числа состояний проверяемой системы. Это позволяет получать приемлемую производительность на практических задачах. Отсутствие поддержки аналогичных алгоритмов в исходной версии системы верификации в среде ДИАНА приводило к невозможности хранения полного пространства состояний в оперативной памяти вычислительной машины и резкому падению производительности из-за необходимости подкачки с жёсткого диска.

3. Применение SPIN для верификации моделей в среде ДИАНА

3.1 Задачи интеграции SPIN в систему верификации

Для выполнения интеграции SPIN в систему верификации среды ДИАНА необходимо решить следующие задачи:

1. Разработать отображение языка MDL на язык PROMELA
2. Обосновать корректность разработанного отображения.
3. Разработать структуру и выполнить реализацию ядра подсистемы верификации на основе SPIN.
4. Исследовать производительность системы верификации с новым ядром для обоснования роста производительности в сравнении с предшествующей версией системы верификации.

3.2 Отображение языка MDL на язык PROMELA

В рамках разработанного отображения, программа на MDL транслируется в программу на PROMELA по следующей схеме:

1. Каждой глобальной переменной MDL-программы сопоставляется глобальная переменная PROMELA-программы. Однотипные переменные группируются в массивы.
2. Каждому автомату MDL-программы сопоставляется прототип процесса на PROMELA с формальными параметрами целого типа.
3. Локальным переменным MDL-автомата сопоставляются локальные переменные прототипа PROMELA-процесса.
4. Переходу в составе тела MDL-автомата, сопоставляется переход в процессе на PROMELA.
5. «Экземплярам» автоматов на MDL сопоставляются PROMELA-процессы с фактическими параметрами-индексами в массивах глобальных переменных. Передача индексов необходима для компенсации отсутствия в PROMELA механизма передачи параметров по ссылке.

Для повышения производительности системы верификации был разработан ряд оптимизаций отображения. В частности, было решено задавать переходы, не использующие разделяемых переменных, в виде атомарных действий. Также были реализованы оптимизации, сокращающие объём кода на PROMELA, что позволило уменьшить время компиляции генерируемого SPIN кода на Си.

В качестве примера приведём результат трансляции описания перехода в MDL-программе в описание перехода на PROMELA.

```
MDL:
; переход из состояния 3, при ненулевом значении local_var
; в состояние 2, с изменением значений m1 и m2
  3, local_var!=0 => 2, m1=1, m2=2, local_var=0;

PROMELA:
:: atomic {(state==3)&&(local_0!=0) ->
        bit_globals[i0]=1; byte_globals[i1]=2; local_0=0; state = 2};
```

3.3 Обоснование корректности отображения

Введём обозначения:

- M – исходная MDL-программа;
- V – множество её переменных, в т.ч. не задаваемых явно счётчиков состояний процессов;
- M' – полученная в результате трансляции PROMELA-программа;
- V' – множество переменных M'.

В рамках описанной выше схемы трансляции, переменные из V' взаимно однозначно соответствуют переменным из V. Для каждой переменной $v \in V$, соответствующая ей переменная $v' \in V'$ имеет не менее широкое множество значений. Множество значений переменной v' может быть шире, чем множество значений переменной v, за счёт того, что тип v' выбирается из предопределённого набора целочисленных типов в PROMELA. Так, если переменная имеет диапазон значений [0..7], для v' будет выбран тип **byte** с диапазоном [0..255].

Под *состоянием* MDL-программы будем понимать набор значений всех её переменных. Аналогично определим состояние PROMELA-программы. Введём также следующие обозначения для M:

1. S – множество состояний программы, каждому из которых соответствует уникальный вектор значений переменных из V;
2. S0 – множество начальных состояний MDL-программы, определяемое возможными допустимыми начальными значениями переменных программы;
3. R – множество переходов между состояниями. Каждый переход из R связан с выполнением одного MDL-оператора перехода, входящего в какой-либо процесс M.

Аналогичные множества для M' обозначим за S', S0', R'. Каждый переход из R' связан со срабатыванием одной атомарного оператора PROMELA из M'.

Под *путём* будем понимать бесконечную последовательность состояний, в которой каждая пара соседних состояний связана переходом. Будем нумеровать состояния в путях M', начиная с состояния, соответствующего завершению выполнения единственного атомарного оператора процесса init (этот оператор порождает остальные процессы M'). Будем ограничиваться рассмотрением только свойств из класса *безопасности* [1], поскольку исходная система верификации позволяет проверять свойства только их этого класса. Для свойств из этого класса, невыполненность свойства на программе равносильна существованию для этой программы пути-контрпримера, на котором не выполнено свойство.

Определение: преобразование TR программы P_1 в программу $P_2 = TR(P_1)$ будем называть **корректным**, если $verify_1(P_1, S) = false \Rightarrow verify_2(P_2, S) = false$. Здесь $verify_1$ и $verify_2$ – процедуры верификации (в общем случае – различные), применимые, соответственно, к P_1 и P_2.

Для свойств из класса безопасности корректность отображения равносильна справедливости утверждения: «если для программы P_1 существует путь-контрпример к свойству S, то для программы P_2 также существует путь-контрпример к этому свойству». Таким образом, для доказательства корректности отображения языка MDL на язык PROMELA достаточно доказать следующее утверждение:

Утверждение 1: для любого пути Π программы M существует путь $\Pi`$ программы $M`$, в любом *i*-м состоянии которого значения переменных из $V`$ совпадают со значениями соответствующих переменных из V в *i*-м состоянии пути Π (будем называть такие состояния *соответствующими*).

Доказательство:

1. Существует путь в $M`$ с начальным состоянием, соответствующим начальному состоянием пути $\Pi$, поскольку означивание переменных из $V`$, не являющихся счётчиками команд, значениями соответствующих переменных из начального состояния Π, является допустимым. Это обусловлено тем, что эти переменные из $V`$ имеют не менее широкие множества значений, чем соответствующие переменные из V.
2. Если *i*-е состояние Π (обозначим за $\Pi(i)$) соответствует *i*-му состоянию некоторого пути в $M`$, то существует переход из $R`$, ведущий из *i*-го состояния второго пути в состояние,

соответствующее состоянию $\Pi(i+1)$.

Обоснование: пусть переход из $\Pi(i)$ в $\Pi(i+1)$ связан с выполнением некоторого MDL-оператора перехода T из M. Ему соответствует атомарный оператор T' из M'. T является разрешённым в состоянии $\Pi(i)$, т.е. условие, стоящее в левой части T, является истинным. Значит, истинным является и условие срабатывания оператора T' (условие в T' идентично условию в T', следовательно, значения этих условий равны в соответствующих состояниях). Присваивания, выполняемые оператором T', идентичны присваиваниям, выполняемым T – следовательно, выполнение T' в рассматриваемом i-м состоянии переводит M' в состояние, соответствующее $\Pi(i+1)$.

3. Рассматривая пункт 1 как базу индукции, а пункт 2 как шаг индукции, получаем доказательство Утверждения 1 по индукции. Путь Π' построен.

Из Утверждения 1 следует, что если для свойства P существует путь-контрпример в программе M, то путь-контрпример для этого свойства есть и в программе M'. Корректность отображения **доказана**.

Следует отметить, что доказательство велось в предположении, что из выполненности условия срабатывания оператора T' следует возможность включения в Π' состояния, возникающего в результате выполнения T'. На практике это означает, что для выполнения корректной верификации в SPIN необходимо настроить учёт требования *справедливости* [2] так же, как это сделано в исходной версии ядра системы верификации в среде ДИАНА.

3.4 Структура ядра системы верификации на основе SPIN

Структура ядра подсистемы верификации среды ДИАНА, построенного на основе SPIN, показана на рисунке 2. Структура построена, исходя из требования совместимости по интерфейсам нового ядра с прежним. Это позволило эффективно провести интеграцию нового ядра в подсистему верификации с минимальными изменениями в окружении ядра. В связи с ограниченной поддержкой в исходной версии системы верификации отображения путей-контрпримеров MDL-программы на пути в ММ-программе, реализация отображения путей-контрпримеров PROMELA-программы на MDL в настоящее время не поддерживается.

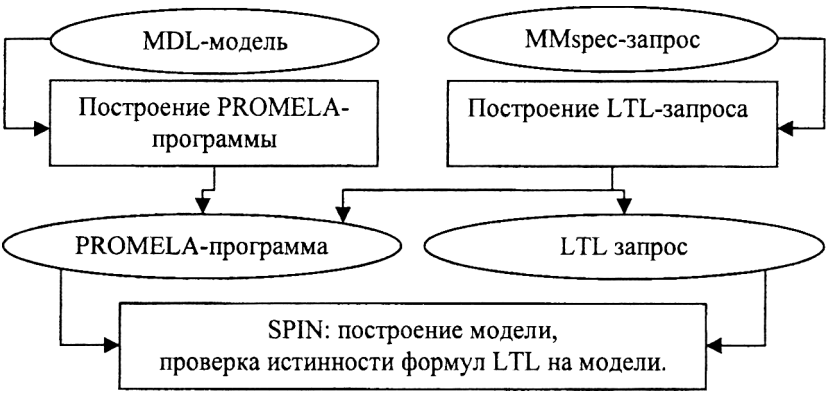


Рисунок 2. Ядро системы верификации на основе SPIN.

4. Экспериментальное исследование

4.1. Объект и цели исследования

В данном разделе приведены результаты исследования производительности системы верификации, основанной на средстве SPIN и применяемой в рамках среды ДИАНА. Производительность системы сравнивается с производительностью следующих версий системы верификации для среды ДИАНА: 1) версии, применявшейся в проекте DrTesy, и использующей логику CTL для промежуточного представления свойств; 2) версии, разработанной в 2005 г. в рамках работ по верификации симметричных параметризованных программ [7] и использующей логику LTL.

При исследовании ставились цели: 1) показать преимущество в производительности системы верификации на основе SPIN перед версией системы, применявшейся в проекте DrTesy; 2) провести сравнительный анализ производительности системы на основе SPIN по отношению к системе из [7]; 3) провести анализ масштабируемости системы верификации при увеличении сложности проверяемой ММ-программы.

4.2. Исходные данные для экспериментов

Для проведения тестирования использовались следующие три имитационные модели, написанные на ММ-языке:

1. Модель MILS [6] – шина MIL-STD-1553B с контроллером и набором абонентов. Параметр модели: число абонентов шины (N_{ab}). Проверялось свойство «ни один из данного подмножества абонентов никогда не получает команду инициализации».

2. Модель MEM [6] – модель общей памяти с 4 абонентами, имеющими различные приоритеты доступа к памяти. Проверялись свойства SM1, SM2 и SM3 из проекта DrTesy, отражающие правильность соблюдения приоритетов абонентов при доступе к разделяемой памяти.
3. Классическая модель «обедающие философы». Параметр модели: число философов (N_{phil}). Проверялось отсутствие блокировки.

4.3. Результаты экспериментов и их анализ

Первый этап исследования включал замеры производительности системы верификации на основе SPIN и системой, применявшейся в проекте DrTesy. Замеры выполнялись при верификации модели MEM в силу относительной простоты этой модели.

Таблица 1. Время верификации свойств модели MEM.

Модель	Свойство	Время, ядро на основе SPIN	Время, ядро образца DrTesy,
MEM	SM1	0m1.233s	около 5 часов
	SM2	0m26.379s	-
	SM3	0m26.228s	-

Система верификации, применявшаяся в проекте DrTesy, проверяла свойство SM1 около 5 часов. Это даёт представление о порядке затрат времени на верификацию сложности с использованием этой версии системы. Выполнение верификации для свойств SM2 и SM3 прерывалось по истечении 10 минут с начала запуска; по данным из отчёта по проекту DrTesy, время проверки этих свойств не меньше, чем для свойства SM1.

Вывод: система верификации на основе SPIN существенно превосходит по производительности систему верификации, применявшуюся в проекте DrTesy.

Второй этап исследования включал сравнительное исследование производительности и масштабируемости системы верификации на основе SPIN и альтернативной системы верификации на основе логики LTL [7]. Исследование проводилось на моделях MILS и «обедающие философы», в которых предусмотрена возможность масштабирования.

В приведённых ниже таблицах **жирным** шрифтом выделены лучшие результаты.

Таблица 2. Время верификации модели «обедающие философы».

N_{phil}	**Время, ядро на основе SPIN**	**Время, Альтернативное ядро с использованием LTL**
40	**0m5.037s**	0m11.239s
60	**0m6.511s**	0m21.706s
80	**0m8.097s**	0m37.624s
100	**0m9.993s**	0m58.401s
110	**0m11.030s**	1m10.039s

Таблица 3. Время верификации модели MILS.

N_{ab}	**Время, ядро на основе SPIN**	**Время, Альтернативное ядро с использованием LTL**
4	0m5.051s	**0m3.844s**
6	0m5.273s	**0m4.349s**
8	0m6.900s	**0m5.870s**
10	**0m6.710s**	0m11.724s
12	**0m13.041s**	0m54.150s
16	**2m43.193s**	21m54.479s

Эксперименты на модели «обедающие философы» и модели MILS показали, что ядро системы верификации на основе SPIN является значительно более масштабируемым, чем альтернативное ядро. Это свойство особенно значительно проявляется на моделях с большим числом взаимодействующих процессов: преимущество в скорости работы возрастает с единиц до десятков раз. Отставание от альтернативного ядра на моделях с малой сложностью имеет место за счёт значительного вклада в суммарное время верификации этапа компиляции Си-кода, генерируемого SPIN. Следует отметить, что отставание (в тех случаях, когда оно имело место) не превышало 1-2 секунд, что не существенно на практике.

Общий вывод: разработанное авторами ядро системы верификации на основе SPIN имеет значительно более высокую производительность и масштабируемость, чем два других рассмотренных ядра.

Заключение

В результате проведённой авторами работы была создана и испытана новая версия системы верификации, входящей в среду ДИАНА. Обоснована корректность используемого при верификации преобразования программ на языке MDL в программы на языке PROMELA. На примерах распределённых имитационных моделей и спецификаций их свойств из проекта DrTesy [6] продемонстрирована

высокая производительность и масштабируемость системы верификации, в том числе в сравнении с ранее разработанными версиями системы верификации в среде ДИАНА.

К перспективам развития системы верификации на основе SPIN следует отнести: 1) поддержку преобразования пути-контрпримера в терминах операторов PROMELA в представление в терминах операторов MDL, что позволит проводить анализ причин невыполненности свойства на привычном уровне ММ-программы; 2) устранение из процедуры верификации промежуточного представления на языке MDL, что позволит регулировать уровень абстракции при преобразовании ММ-программы в PROMELA-программу и более полно задействовать поддерживаемые в SPIN механизмы оптимизации числа состояний модели.

В рамках практического применения разработанной системы верификации планируется её использование для проверки свойств моделей современных бортовых вычислительных комплексов.

Литература

1. L. Lamport. Proving the correctness of multiprocessor programs, Transitions on Software Engineering, 1, 1977
2. Э. М. Кларк, О. Грамберг, Д. Пелед. Верификация моделей программ: Model checking. М.: МЦНМО, 2002
3. Смелянский Р.Л. Применение темпоральной логики для спецификации поведения программных систем // М., Программирование, № 1, 1993, стр. 3-27
4. Царьков Д.Ю. Верификация распределённых программ методом проверки на модели. Диссертация на соискание степени к.ф.-м.н. // М., 2001
5. Бахмуров А.Г., Смелянский Р.Л., Чистолинов М.В. Инструментальная поддержка проектирования распределенных встроенных вычислительных систем // Труды Международной конференции РАСО'2001. -М. ИПУ РАН, 2001, с.33-43
6. Verification and Testing Report (DrTesy project deliverable D3). [HTML] (http://citeseer.ist.psu.edu/337450.html)
7. Коннов И.В. Захаров В.А. О верификации параметризованных симметричных распределенных программ // Труды Международной конференции РАСО'2003. -М. ИПУ РАН, 2003
8. Gerard J. Holzmann. The Spin Model Checker: Primer and Reference Manual. M.: Addison Wesley, 2003
9. Spin HomePage [HTML] (http://spinroot.com)
10. Патрик Риган, Скотт Хемилтон. NASA: миссия надёжна // Открытые системы. 2004 №3 [HTML] (http://www.osp.ru/os/2004/03/045.htm)
11. Gerard J. Hozman. Margaret H. Smith. Automatic Software Feature Verification. [PDF] (http://spinroot.com/gerard/pdf/bltj2000.pdf)

Волканов Д.Ю., Григорян М.В.

Методика поиска оптимального набора механизмов отказоустойчивости для бортовых вычислительных систем[1]

Введение

Под *вычислительной системой реального времени* обычно понимают такую вычислительную систему, работоспособность которой зависит не только от логических результатов вычислений, но и от промежутка времени, за который эти результаты были получены [1]. *Бортовая вычислительная система (БВС)* - это частный случай системы реального времени, на которую, помимо ограничений на время выполнения вычислений, могут накладываться ограничения на габариты, массу и потребляемую мощность.

При разработке БВС необходимо уделять должное внимание средствам отказоустойчивости и повышения надёжности. *Надёжность* - это способность системы безотказно выполнять определенные функции при заданных условиях в течение заданного периода времени с вероятностью не ниже заданной. Важными понятиями при исследовании надёжности вычислительной системы являются понятия *неисправности*, *ошибки* и *отказа* [2].

Под *неисправностью* в системе будем понимать некий дефект в программе или аппаратуре, который в определенных случаях может помешать эксплуатировать систему согласно ее спецификации. Говорят, что в какой-то момент времени в системе произошла *ошибка*, если в этот момент времени состояние системы отличается от ожидаемого. Говорят, что в системе произошёл *отказ*, если она не обеспечивает того поведения и функциональности, которого можно было ожидать от неё заранее, то есть, результат работы системы на каких-то входных данных отличается от ожидаемого по спецификации. При наличии неисправности в системе, система может перейти в ошибочное состояние, а ошибочное состояние привести к отказу системы. Одним из основных свойств системы, составляющих надёжность, является отказоустойчивость. *Отказоустойчивость* – это свойство системы или компонента системы, заключающееся в способности обнаружить и ликвидировать ошибочные состояния системы без влияния на её работоспособность.

[1] Работа выполнена при частичной поддержке РФФИ, грант № 04-01-00556

В статье рассматривается задача выбора оптимального набора механизмов отказоустойчивости для БВС, а в качестве решения этой задачи была предложена методика, работающая с использованием генетических алгоритмов [3]. Первые работы, посвященные применению генетических алгоритмов для решения различных задач обеспечения надёжности вычислительных систем, принадлежат Томпсону [4]. Его идеи использовал и развил в своих работах Хартманн [5]. В этих работах решаются задачи проектирования отказоустойчивых интегральных схем, состоящих из различных логических элементов, вентилей и прочих низкоуровневых компонентов вычислительных систем. В данной статье рассматривается более высокий уровень представления вычислительной системы, а именно *системный.*

1. Задача выбора оптимального набора механизмов отказоустойчивости

Рассматривается БВС, описанная при помощи модели на основе инварианта программы [6]. В описании БВС выделяются аппаратные и программные компоненты. Для данной БВС известен набор неисправностей, которые могут возникать в компонентах. Также известен набор доступных механизмов отказоустойчивости [7]: резервирование компонентов, переконфигурирование системы, метод кодирования, метод с использованием контрольных точек, многоверсионное программирование, которые могут применяться к каждому из компонентов.

Необходимо выбрать такой набор механизмов отказоустойчивости, который бы предотвращал возможность возникновения отказа системы для максимального количества неисправностей при заданных ограничениях на систему.

В качестве ограничений на систему выступают: ограничение на применение данного механизма отказоустойчивости для конкретного компонента и сохранение директивных сроков выполняемых в системе задач.

2. Описание методики

Методика включает в себя следующие шаги:

1. Создание имитационной модели с нужным уровнем детальности.
2. Внесение неисправностей различных типов в модель.
3. Поиск при помощи генетического алгоритма набора механизмов отказоустойчивости.

4. Построение новой модели с полученным на шаге 3 набором механизмов отказоустойчивости и проверка соблюдения директивных сроков задач.
5. Если не выполнен критерий останова на шаге 3 или не выполнены ограничения, проверенные на шаге 4, то переход к шагу 3, иначе текущее решение принимается в качестве решения задачи.

Ниже приводится более подробное описание каждого из шагов методики.

Ввиду невозможности использования натурных компонентов, вычислительная система (её программные и аппаратные компоненты) на первом шаге применения методики представляется в виде имитационной модели.

На втором шаге с помощью средства внесения неисправностей в эту модель системы вносятся неисправности различных типов и распределяются между компонентами системы. Для моделирования состояния системы, в котором возникает отказ, надо каким-либо образом учитывать неисправности, возникающие в натурной системе. Существуют алгоритмы [8], позволяющие имитировать появление отказов в системе путём внесения неисправностей в модель системы.

На третьем шаге при помощи генетического алгоритма ищется набор механизмов отказоустойчивости. Для механизмов отказоустойчивости, которые мы хотим применять, известно какие неисправности они могут устранять, и известны наборы параметров этих средств. Разумно предположить, что если средство, позволяющее устранить неисправность, применяется к неисправному компоненту, то неисправность не приводит к отказу системы и таким образом ликвидируется. Из-за применения средств отказоустойчивости возрастает время выполнения задач в системе и они могут не укладываться в директивный срок.

Решение задачи выбора механизмов отказоустойчивости представляет собою закодированную битовую строку, имеющую структуру, показанную на рисунке 1:

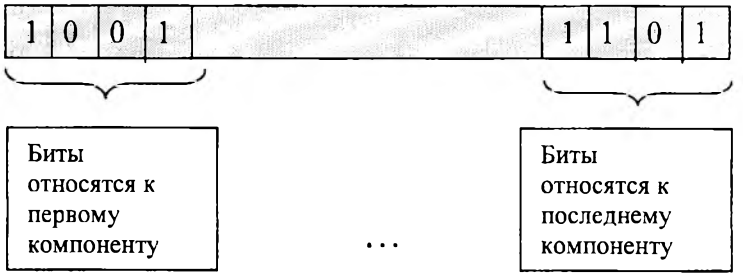


Рисунок 1. Структура закодированного решения

Битовая строка состоит из блоков, которые соответствуют компонентам вычислительной системы. Число битов в блоке соответствует числу заданных средств отказоустойчивости, и каждый бит определяет, применяется ли то или иное средство к текущему компоненту (1 – применяется, 0 – нет).

В качестве целевой функции выступает количество неустранённых неисправностей в системе после применения средств отказоустойчивости, указанных в решении. Эта функция минимизируется.

Этот шаг рассматриваемой методики можно разделить на следующие этапы:

I. Генерация начальной популяции решений, на основе информации о моделируемой системе и внесённых в неё неисправностях.
II. Вычисление значений целевой функции для членов популяции и фиксирование наилучшего на текущий момент решения. Для каждого заданного в популяции решения целевая функция определяет, качество данного решения.
III. Проверяется критерий останова, если он достигнут, то вычисления прекращаются. В качестве критерия останова используется условие выполнения алгоритмом заданного количества итераций без улучшения целевой функции.
IV. Выбор решения для следующей популяции. На данный момент алгоритм использует схему пропорциональной селекции для выбора элементов популяции.
V. Над элементами новой популяции производится операция скрещивания. Элементы для скрещивания выбираются произвольным образом. Используется схема универсального (вероятностного) скрещивания двух решений, в результате чего появляются два новых решения. При скрещивании отбрасываются решения, которые не удовлетворяют ограничениям на использование механизмов

отказоустойчивости. Например, механизм контрольных точек не применяется для аппаратных компонентов, поэтому в битовом представлении решения в соответствующих позициях для аппаратных будет стоять '0' и операции скрещивания выбраны так, что не будут его модифицировать.

VI. Над элементами новой популяции производится операция мутации. Уровень мутации элемента популяции зависит от параметра, задаваемого пользователем. При мутации также как и при скрещивании отбрасываются решения, которые не удовлетворяют ограничениям на использование механизмов отказоустойчивости.

VII. Переход к этапу II.

На четвёртом шаге методики по полученному на предыдущем шаге решению строится имитационная модель, в которой, если для конкретного компонента в полученном решении стоит '1' в качестве признака использования данного механизма отказоустойчивости, то данный механизм применяется к компоненту в имитационной модели. После запуска этой модели и проверки директивных сроков выполнения задач данное решение принимается как текущее, либо не принимается.

Затем возвращаемся к предыдущему шагу и повторяем процесс получения нового решения и его проверки, пока решение, удовлетворяющее ограничениям, не будет улучшаться при заданном числе итераций генетического алгоритма.

Данная методика без автоматической генерации модели с механизмами отказоустойчивости была реализована в виде программного средства.

3. Апробация методики

Проверка корректности методики проводилась путём сравнения результатов, полученных с использованием генетического алгоритма с результатами, полученными с помощью алгоритма полного перебора.

В качестве среды моделирования была выбрана система ДИАНА [9]. Для внесения неисправностей в модель вычислительной системы используется Средство автоматического внесения неисправностей [8]. Реализация генетического алгоритма, осуществлялась при поддержке Среды конструирования специализированных алгоритмов оптимизации[10].

Среда специализированных алгоритмов оптимизации предоставляет библиотеку классов на языке C++, реализующих проблемно-независимую часть алгоритмов. В рамках исследования

методики были реализованы классы на C++, унаследованные от стандартной библиотеки классов средства конструирования алгоритмов. Они описывали решение задачи, реализовывали целевую функцию и операции скрещивания и мутации.

В качестве исследуемой системы была рассмотрена часть реальной бортовой вычислительной системы. Исследуемая система состоит из процессора, на котором выполняются несколько функциональных задач и планировщика, распределяющего процессорное время между задачами, задачи могут обмениваться данными. Эта система моделировалась в среде ДИАНА. Функциональные задачи моделировались процессом *FunctionalTask*, а планировщик моделировался процессом *Scheduler*, наконец процесс *Timer* моделировал функции системных часов.

Схема модели приведена на рисунке 2.

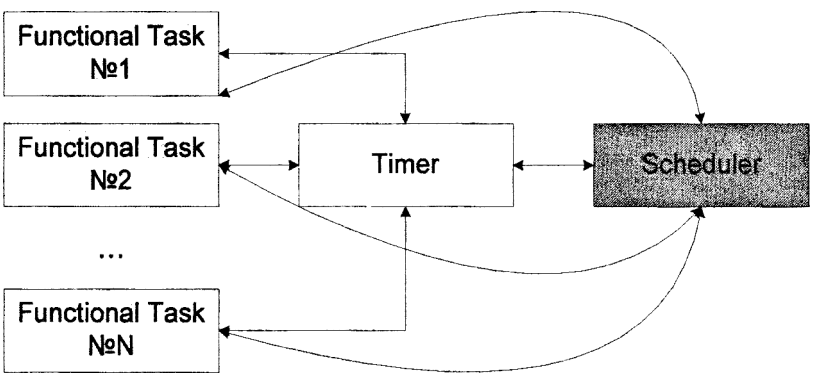


Рисунок 2. Схема исследуемой системы

В качестве механизмов отказоустойчивости применялись: резервирование (для таймера), многоверсионное программирование и метод контрольных точек (для функциональных задач и планировщика). В планировщик и в набор функциональных задач вносились неисправности, приводящие к отказу компонентов системы. В таймер и в набор функциональных задач вносились неисправности, приводящие к искажению данных внутри компонентов системы. Количество функциональных задач варьировалось в диапазоне от 3 до 10. Количество итераций без улучшения целевой функции было равно 25.

Как генетический алгоритм, так и переборный используют одну и ту же целевую функцию, так что сложность каждой итерации у них одинакова, если пренебречь временем на мутацию и скрещивание.

В результате экспериментов с данной моделью было обнаружено, что количество итераций, необходимое для получения приемлемого по качеству решения при помощи предложенной методики существенно меньше, чем при полном переборе всех вариантов. И при полном переборе и при использовании методики, достигался глобальный минимум, равный 0. Это соответствует исправлению всех неисправностей в системе. Результаты, полученные при экспериментах с различными наборами неисправностей, показаны в таблице 1. В первом столбце таблицы приведено количество функциональных задач в модели. Во втором столбце приведено среднее количество итераций генетического алгоритма для достижения минимума при различных наборах неисправностей. В третьем столбце приведена оценка количества итераций алгоритма полного перебора, которая не зависит от наборов неисправностей.

Количество функциональных задач	**Количество итераций генетического алгоритма**	**Оценка количества итераций алгоритма полного перебора**
3	70	8192
5	360	524288
10	920	17179869184

Таблица 1. Сравнение количества итераций алгоритмов для нахождения оптимального решения.

Заключение

Эксперименты показали, что предложенная в статье методика работает корректно. В дальнейшем предполагается:

- Реализовать часть методики, относящуюся к автоматической генерации отказоустойчивой модели и её прогона, для автоматизации проверки соблюдения директивных сроков задач. Это позволит проводить гораздо большее число экспериментов за одно и тоже время.
- Исследовать поддержку в популяции большего количества различных решений. Это ухудшит скорость сходимости генетического алгоритма. Зато позволит в случае нахождения серии решений, в которых задачи не удовлетворяют директивным срокам, быстрее получить решения, в которых задачи удовлетворяют директивным срокам.
- Провести эксперименты не только с генетическими алгоритмами, но и с алгоритмами имитации отжига. Здесь пока

неясно будет ли преимущество в скорости сходимости в случае использования алгоритмов имитации отжига.

- Оценить применимость методов многокритериальной оптимизации для нахождения компромисса между надёжностью и производительностью. На данный момент все характеристики системы, на которые не влияют механизмы отказоустойчивости, считаются фиксированными.

Литература

1. Stankovic J. A. Real-time Computing. // Byte Magazine. 1992. 17. N 8. P. 155-160.
2. Ammar H., Cukic B., Fuhrman C., Mili A. A Comparative Analysis of Hardware and Software Reliability Engineering. // Annals of Software Engineering. 2000. 10. N 1-4. P. 103-150.
3. Гладков Л.А., Курейчик В.В., Курейчик В.М. Генетические алгоритмы. / Под ред. В.М. Курейчика. – 2е изд., испр. и доп. – М.: ФИЗМАТЛИТ, 2006., 320 с.
4. Thompson, A. (1995). Evolving fault tolerant systems. In Proc. 1st IEE/IEEE Int. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA'95), pages 524-529. IEE Conf. Publication No. 414.
5. Hartmann,M. and Haddow, P. C. (2004). Evolution of fault-tolerant and noise-robust digital designs. IEE Proc. -Comput. Digit. Tech., 151(4):287-294.
6. Смелянский Р.Л. Анализ производительности распределенных микропроцессорных вычислительных систем на основе инварианта поведения программ. / Дисс. на соискание ученой степени доктора физико-математических наук. М.:МГУ, 1990
7. Shooman M.L. Reliability of computer systems and networks. John Willey & Sons Inc. 2002. 528 p.
8. Волканов Д.Ю., Шаров А.А. Программное средство автоматического внесения неисправностей для оценки надежности вычислительных систем реального времени с использованием имитационного моделирования // Методы и средства обработки информации. Труды второй Всероссийской научной конференции. - М.: Издательский отдел факультета вычислительной математики и кибернетики МГУ им. М.В. Ломоносова, 2005. - С.457-464.
9. A.Bakhmurov, A.Kapitonova, R.Smeliansky DYANA: An Environment for Embedded System Design and Analysis, in Proc. of 5-th International Conference TACAS'99, Amsterdam, The Netherlands, March 22-28, 1999. Springer (LNCS Vol.1579), pp.390-404

10. Калашников А.В., Костенко В.А., Маркин М.И. Средства конструирования итерационных алгоритмов для решения задач комбинаторной оптимизации // Искусственный интеллект, 2004., No 2, С.91-95.

Раздел IV
Сетевая обработка

Козлов Д. Д., Петухов А. А.

Методы обнаружения уязвимостей в web-приложениях

Введение

В современных информационных системах широкое распространение получило использование web-приложений. Web-приложения обладают такими важными достоинствами, как простота и привычность интерфейса, возможность удаленной работы через Интернет, быстрота разработки приложения. Вместе с этим web-приложения создают большое число проблем, связанных с обеспечением информационной безопасности, ведь их разработка часто выполняется в сжатые сроки, а приложение становится доступным через Интернет и для пользователей, и для злоумышленников. Уязвимости позволяют злоумышленникам похищать конфиденциальную информацию, проводить несанкционированные изменения данных, нарушать доступность приложения. В настоящее время проблема обеспечения безопасности web-приложений весьма актуальна: согласно [1] более 60% от всех обнаруживаемых уязвимостей относятся к web-приложениям.

Одним из широко распространенных методов обеспечения безопасности web-приложений является обнаружение уязвимостей web-приложения с целью последующего их устранения. В данной работе рассмотрены современные методы обнаружения уязвимостей в web-приложениях и проведен анализ их возможностей.

1. Методы обнаружения уязвимостей web-приложений

На сегодняшний день согласно исследованиям OWASP [8] наиболее эффективным способом обнаружения уязвимостей web-приложений является экспертный анализ исходных кодов web-приложения (code review). Этот способ весьма трудоемок, требует высокой квалификации эксперта и не защищен от ошибок эксперта. Поэтому активно развиваются методы автоматического обнаружения уязвимостей web-приложений.

Методы автоматического обнаружения уязвимостей web-приложений можно разделить на две основные группы: 1) методы, анализирующие работу развернутого на стенде web-приложения без обращения к исходным кодам web-приложения; 2) методы,

анализирующие исходные коды web-приложения и конфигурационные настройки.

Первая группа методов рассматривает web-приложение с точки зрения внешнего пользователя, то есть потенциального злоумышленника. В эту группу входят следующие методы:

- Метод получения идентифицирующей информации о web-приложении и выявления его уязвимостей с помощью бюллетеней безопасности (security advisory).
- Метод тестирования на проникновение.

Вторая группа состоит из следующих методов:

- Метод статического анализа исходных кодов web-приложения.
- Метод динамического анализа исходных кодов web-приложения.

Для разработки web-приложений применяются разнообразные инструментальные средства и языки программирования. Наиболее популярными среди них являются PHP, Perl, Python, Ruby, Java/JSP, .Net/ASP. Возможности, предоставляемые этими средствами, существенно различаются, ограничивая применение тех или иных методов анализа. В данной работе рассмотрены методы автоматического обнаружения уязвимостей и описаны области применимости и ограничения каждого из методов.

1.1 Метод получения идентифицирующей информации о web-приложении

Метод получения идентифицирующей информации о web-приложении основан на посылке от имени обычного пользователя web-приложению набора HTTP-запросов, ответы на которые позволят сделать вывод о том, на каком web-сервере работает приложение, с помощью какой технологии оно разработано, какие версии программного обеспечения использует, какие стандартные компоненты оно использует и т.д. Для определения типа и версии web-сервера используется техника fingerprint [2], которая основана на том, что каждый web-сервер по-своему обрабатывает HTTP-протокол, в результате чего можно с высокой степенью вероятности определить тип и даже версию web-сервера путем посылки серверу набора корректных и некорректных запросов и анализа соответствующих ответов. Для определения остальных парамстров используется анализ HTTP-ответов и HTML-страниц средствами поиска регулярных выражений. Шаблоны для поиска задаются экспертом. Например, использование расширения .jsp в URI может свидетельствовать о том, что web-приложение было разработано с использованием технологии JSP.

Возможность получения идентифицирующей информации пользователем приложения является потенциальной уязвимостью ввиду следующих причин: в сети Интернет накоплены огромные объёмы данных по уязвимостям программных продуктов с указанием версий программ, методов реализации атак, бюллетени безопасности Bugtraq [9] и Security Focus [10] ежедневно публикуют отчёты о новых найденных уязвимостях. Эта информация предназначена для администраторов и специалистов в области информационной безопасности, в тоже время она доступна через сеть Интернет всем желающим. Администраторы информационных систем часто не устанавливают обновления безопасности вовремя. В результате становится возможным взломать информационную систему путем использования всем известной уязвимости, которую администратор не закрыл установкой соответствующего обновления. В последние годы на хорошо известные и незакрытые уязвимости операционных систем было проведено большое количество атак, вызвавших широко известные эпидемии Интернет-червей.

Метод получения идентифицирующей информации о web-приложении и выявления его уязвимостей с помощью бюллетеней безопасности получил широкое практическое применение при проведении атак ввиду своей простоты и доступности. При этом сам метод не позволяет найти новые уязвимости web-приложения, а его полезность для обнаружения уязвимостей новых приложений состоит в том, что он позволяет указать на саму возможность утечки информации о web-приложении.

Данный метод требует от эксперта настройки шаблонов, по которым производится поиск идентифицирующей информации. Эти шаблоны в большинстве случаев специфичны для конкретной технологии разработки web-приложений.

1.2 Метод тестирования на проникновение

Метод тестирования на проникновение (penetration testing) рассматривает web-приложение с точки зрения внешнего пользователя, то есть потенциального злоумышленника. При этом считается, что злоумышленник обладает такими же возможностями, как и обычный пользователь, т.е. не имеет доступа к исходным кодам web-приложения, конфигурационным настройкам и т.п. Метод предусматривает тестирование работающего на стенде web-приложения путем посылки запросов которые эмулируют пользовательскую активность, включающую в том числе и некорректные запросы, соответствующие действиям злоумышленника.

При поиске уязвимостей в web-приложении методом тестирования на проникновение возникают три основные задачи [3]:

- Получение и анализ структуры web-приложения.
- Построение набора тестовых HTTP-запросов на основе построенной структуры web-приложения.
- Прогон тестового набора с анализом ответов web-приложения для выявления уязвимостей.

Задача получения и анализа структуры web-приложения состоит в том, чтобы построить полный список URI web-приложения, методов доступа к ним и списков их параметров, выделить URI, защищённые аутентификацией. Данная информация необходима для построения набора тестовых запросов к web-приложению. Для автоматического получения структуры web-приложения используются сетевые роботы (web crawlers) [5]. В случае статических HTML-страниц работа робота сводится к обходу всех доступных ссылок web-приложения, а в случае наличия форм и скриптов – к заполнению форм и извлечению гиперссылок из скриптов. Получение полной структуры web-приложения возможно не во всех случаях – основные проблемы возникают при заполнении web-форм и интерпретации скриптов [4, 5].

Web-формы делятся на две категории: формы, содержащие только элементы с ограниченными областями значений (select, option и т.д.) и формы, содержащие элементы с неограниченными областями значений (например, textarea). Обход страниц, связанных с посылкой форм первого типа осуществляется последовательным перебором всех комбинаций значений полей формы. Для переходов, связанных с заполнением текстовых полей, возможны два способа обхода: ручной или автоматизированный. При ручном способе управление процессом передаётся человеку, а при автоматизированном способе применяются эвристические методы заполнения текстовых полей, основанные на словарях, например, VeriWeb [4], HiWE [5]. Для форм, содержащих элементы с неограниченной областью значений, автоматизированный обход не гарантирует нахождения всех URI приложения.

Для автоматического обхода страниц, содержащих ссылки, формируемые средствами скриптовых языков, необходима возможность не только интерпретации скриптового языка, но и эмуляции действий пользователей для формирования тех или иных событий, например, onClick, onMove. Существующие средства интерпретации скриптовых языков не гарантируют выявления всех гиперссылок.

Задача построения тестового набора запросов к web-приложению состоит в том, чтобы по исходным данным (список URI приложения, методы доступа, принимаемые параметры) подобрать запросы так, чтобы было обнаружено как можно больше уязвимостей [7]. Существующие способы построения таких запросов рассмотрены ниже.

Построение запросов по базе ресурсов подразумевает, что существует база ресурсов, которые потенциально могут встретиться в структуре web-приложения. Наличие в web-приложении того или иного ресурса из базы свидетельствует об уязвимости, связанной с возможным доступом к этому ресурсу. Например, в web-приложении присутствуют имена известных уязвимых CGI-сценариев и имена конфигурационных файлов web-серверов. Поиск ресурсов происходит по всей структуре web-приложения. В каждом каталоге (из структуры web-приложения) по очереди запрашиваются все имена, представленные в базе ресурсов. Способ построение запросов по базе ресурсов полностью автоматический и опирается на накопленную информацию о характерных уязвимостях web-приложений. Таким образом, данный способ имеет ограничения, аналогичные рассмотренным выше в методе получения идентифицирующей информации, но в тоже время позволяет выявить новые уязвимости web-приложения, основанные на типовых ошибках разработчиков и администратора.

Генерация запросов по шаблону с типизированными параметрами состоит в том, что для каждого URI задаётся шаблон, параметры которого типизированы, после чего происходит автоматическая генерация запросов по заданному шаблону со случайным выбором значений конкретных параметров. Значения параметров могут задаваться регулярными выражениями. Данный способ используется для обнаружения ошибок проверки корректности введенных пользователем данных. Критерий наличия уязвимости вводит пользователь – если ответ web-сервера соответствует некоторому заданному регулярному выражению, то делается вывод об уязвимости приложения. Описанным способом также реализуются переборы паролей. Данный способ требует привлечения эксперта и тонкой настройки, так как необходимо для каждого ресурса web-приложения составить наборы значений параметров. При этом данный способ не зависит от технологии, на которой разработано web-приложение, так как работает только в терминах протокола HTTP.

Анализ настроек каталогов web-приложения заключается в том, что по структуре web-приложения проверяются типовые уязвимости, связанные с неправильным конфигурированием web-приложения и web-сервера. Сюда относятся проверка возможности автоматического построения индекса каталога, выполнения HTTP-методов PUT и DELETE, возможность обращения к ресурсам из областей аутентификации напрямую, возможность получения исходных кодов web-приложения.

Задача прогона тестового набора и анализа ответов сервера состоит с том, чтобы сделать правильный вывод о том, демонстрирует ли данный HTTP-запрос наличие уязвимости в web-приложении или

нет. Данная задача тесно связана с задачей построения тестового набора, а основная проблема состоит в определении критериев наличия уязвимости. В настоящее время для решения этой задачи применяется метод, в котором распознавание осуществляется регулярными выражениями, задаваемыми экспертом. Таким образом, данный метод также требует тонкой настройки экспертом, при этом метод работает в терминах протокола HTTP и не зависит от технологии, на которой разработано web-приложение.

Метод тестирования на проникновение существенно меньше зависит от технологии разработки web-приложения, чем другие методы. Данный метод позволяет накапливать знания эксперта в виде набора правил построения запросов и набора шаблонов для анализа HTTP-ответов. Этот метод получил широкое распространение для выявления уязвимостей разработанных web-приложений, когда необходимо оценить наличие хотя бы типовых ошибок при разработке и настройке web-приложения. Достоинством данного метода является то, что метод позволяет оценивать развернутое и настроенное web-приложение, выявляя не только ошибки кодирования, но и ошибки конфигурирования web-сервера и web-приложения.

1.3 Метод статического анализа

Метод статического анализа исходных кодов web-приложения не предусматривает реального выполнения web-приложения. Вместо этого производится построение графов управления и зависимостей по данным и обнаружение уязвимостей посредством анализа этих графов. Для обнаружения уязвимостей используется два основных подхода: анализ типов безопасности [7] и анализ потоков данных [6]. В каждом из подходов уязвимость определяется, как нарушение в программе свойства noninterference [11].

При анализе типов безопасности вводится набор типов безопасности *(t1, t2, ..., tn)*, над которыми вводится отношение частичного порядка $\leq$. Каждой переменной программы ставится в соответствие её тип безопасности. Существуют два способа определения типов безопасности всех переменных – ручной и автоматический. Ручной способ предполагает, что при каждом объявлении переменной программист должен явно задать её тип безопасности. Автоматический способ предполагает, что даны: разметка типами безопасности переменных, содержащих входные данные, разметка функций, осуществляющих пользовательский ввод правила вывода типов безопасности ещё неразмеченных переменных. Таким образом, при анализе программы последовательно от начала до конца все неразмеченные переменные получают свой тип безопасности

автоматически. Тип безопасности переменной постоянен на всё время жизни переменной.

Для выявления уязвимостей необходимо специфицировать, требуемые уровни безопасности параметров функций. В реализациях метода данная спецификация часто находится в отдельном конфигурационном файле и не зависит от конкретной программы (аннотируются стандартные библиотечные функции). Следующий пример иллюстрирует данный подход:

Типы безопасности: {untainted, tainted}
Начальная разметка: int main (int argc, tainted char *argv[])
Сама программа:

```
void LogMessage(char *fmt, ...){
        ...
        fprintf(fd, fmt, arg);
}

int main (int argc, tainted char *argv[]){
        char *arg2 = argv[2];
        ...
        char *Mess = new [strlen(arg2) + strlen("arg2 = ") + 1];
        strcat(strcpy(Mess, "arg2 ="), arg2);
        LogMessage(Mess);
}
```

Программа имеет уязвимость форматной строки. Методом анализа уровней безопасности данная уязвимость будет обнаружена, если функция *fprintf* будет аннотирована, как принимающая второй аргумент только уровня безопасности *untainted.*

В работе [12] было доказано, что в языках со строгой типизацией данный подход обнаруживает все уязвимости, связанные с некорректными потоками информации.

Анализ типов безопасности имеет существенный недостаток – постоянную привязку типа безопасности к переменной. В результате тип безопасности определяется без учета того, из какого источника данные попали в переменную, что приводит к большому числу ошибок первого рода. Для того чтобы избежать большого числа ошибок первого рода, было предложено привязывать тип безопасности не к переменной, а к её значению. В результате каждая переменная теоретически может получать любой тип безопасности на протяжении своей жизни. Этот подход получил название анализа потоков данных.

В рамках анализа потоков данных каждой конструкции языка программирования сопоставляются формальные правила вывода результирующего типа безопасности переменных, участвующих в этой конструкции. А для библиотечных функций создается разметка,

описывающая типы безопасности параметров и результата функции. На основе разметки и формальных правил вывода анализатор определяет в каждом участке кода тип безопасности конкретных данных. Сопоставляя аннотации библиотечных функций и типы безопасности их параметров, можно выявить несоответствия типа безопасности требованиям разметки, что указывает на наличие уязвимости в web-приложении.

В рамках данного подхода для части конструкций программы программист сам должен указывать тип безопасности переменных, например, при необходимости повысить тип безопасности после проверки корректности введенных данных.

Обнаружение уязвимостей web-приложений методом статического анализа предусматривает решение следующих основных задач:

- Определение конструкций языка и библиотечных функций, которые возвращают данные, потенциально контролируемые злоумышленником. Например, это параметры HTTP запросов GET и POST, пользовательские cookie и т.д. Вся информация, полученная из таких конструкций, помечается меткой tainted.
- Разработка аксиом распространения метки tainted между переменными приложения. Данные аксиомы должны поддерживать не только тривиальные присваивания, но и более сложные конструкции языка, такие, например, как присваивания через массивы, работу через ссылки и т.д. Данная система аксиом должна содержать правила, по которым информация, помеченная как tainted, может вновь стать безопасной, untainted. Полнота и точность метода обнаружения уязвимостей преимущественно зависит от качества модели распространения метки tainted по приложению.
- Определение конструкций языка, которые не должны принимать данные с типом безопасности tainted. В этот список должны входить функции для работы с системным окружением, с СУБД, с почтовыми сервисами и т.д.

Метод статического анализа исходных кодов web-приложения позволяет обнаруживать уязвимости, связанные с неустойчивостью web-приложения к некорректным входным данным. Другие классы уязвимостей данный метод обнаруживать не позволяет. Метод специфичен для каждой технологии создания web-приложений и требует от программиста аннотирования функций проверки корректности входных данных.

1.4 Метод динамического анализа

Метод динамического анализа исходных кодов web-приложения по своей сути аналогичен методу статического анализа исходных кодов за исключением того, что анализ производится в процессе выполнения web-приложения без построения графов программ. Вместо этого в процессе выполнения программы (т.е. неявно сформирован один из путей в графе управления) для каждой переменной производится вычисление типа безопасности, а для каждой размеченной функции сравниваются типы безопасности параметров со спецификацией, указанной в разметке.

Метод динамического анализа позволяет осуществлять обнаружение уязвимостей для широко применяемых при создании web-приложений динамических языков, как, например, Perl, PHP, Python, Ruby. В настоящее время для интерпретируемых языков web-приложений метод динамического анализа часто интегрируется в интерпретатор языка, например, Perl Tainted Mode для языка Perl, PHPrevent для языка PHP и Ruby's Tainted Mode для языка Ruby.

Основной проблемой метода динамического анализа является проблема снятия метки tainted с данных. Существует несколько способов решения данной проблемы: дополнительная разметка исходного кода, применение стандартных библиотеки «очищающих» функций из состава технологии, на которой разработано web-приложение, применение различного рода эвристик. Так, в технологии Perl любое регулярное выражение над tainted данными снимает с них метку tainted. Таким образом, Perl полностью доверяет программисту в проведённых очистках входной информации. В технологии Ruby программист должен явно указать после своих проверок изменение типа безопасности на untainted.

Метод динамического анализа, в отличие от статического анализа, позволяет осуществлять обнаружение уязвимостей в генерируемом на лету коде. В тоже время, метод динамического анализа анализирует лишь один из возможных графов выполнения программы и не позволяет сделать вывод об отсутствии уязвимостей во всей программе. Метод динамического анализа, аналогично статическому анализу специфичен для каждой технологии создания web-приложений.

Заключение

Для разработки web-приложений применяются разнообразные инструментальные средства и языки программирования. В частности широко распространены динамические языки программирования, такие как PHP, Perl, Python. Применение метода статического анализа к динамическим языкам имеет существенный недостаток: в приложениях,

где исходный код может генерироваться во время исполнения, метод статического анализа не может гарантировать обнаружения всех уязвимостей, связанных с нарушением свойства noninterference. Метод динамического анализа также не может гарантировать обнаружения всех уязвимостей, связанных с нарушением свойства noninterference, если тестовые наборы данных не покрывают все возможные пути выполнения программы. Метод тестирования на проникновение также не может гарантировать обнаружения всех уязвимостей. Таким образом, все рассмотренные методы обнаружения уязвимостей web-приложений не могут гарантировать обнаружения всех уязвимостей.

Как было указано выше, методы статического и динамического анализа позволяют обнаруживать только уязвимости, связанные с передачей web-приложению некорректных входных данных. В то время как метод тестирования на проникновение позволяет обнаруживать и другие уязвимости.

На основании проведенного анализа можно сделать вывод, что дальнейшее развитие методов автоматического обнаружения уязвимостей web-приложений пойдет по пути интеграции возможностей различных методов с тем, чтобы было возможно

- охватить максимально широкий класс уязвимостей;
- контролировать полноту обнаружения уязвимостей, чтобы по итогам автоматического анализа можно было бы (в идеале) дать гарантию отсутствия уязвимостей заданных классов.

Литература

1. Andrews M., The State of Web Security. IEEE Security & Privacy, vol. 4, no. 4, pp. 14-15, 2006.
2. Shah S., "An Introduction to HTTP fingerprinting", http://net-square.com/httprint/httprint_paper.html, 2004.
3. Auronen L., Tool-Based Approach to Assessing Web Application Security. Seminar on Network Security, 2002.
4. Benedikt M., Freire J., Godefroid P., VeriWeb: Automatically Testing Dynamic Web Sites. Proceesings of 11-th WWW Conference, 2002.
5. Raghavan S., Garcia-Molina H., Crawling the Hidden Web. Stanford University Technical Report, 2000.
6. Sabelfeld, A. Myers. Language-Based Information-Flow Security.
7. Yao-Wen Huang, Shih-Kun Huang Tsung-Po Lin Chung-Hung Tsai Web Application Security Assessment by Fault Injection and Behavior Monitoring. Proceedings of 12-th WWW Conference, 2003.
8. Curphey M., Wiesman A., Van der Stock A., Stirbei R. A Guide to Building Secure Web Applications and Web Services. OWASP, 2005.
9. Bugtraq. http://en.wikipedia.org/wiki/Bugtraq.
10. Security Focus. http://securityfocus.com.

11. J. A. Goguen and J. Meseguer Security Policies and Security Models. In Proc. 1982 IEEE Symposium on Security and Privacy, 1982.
12. D. Volpano, G. Smith A type-based approach to program security._In Proc. TAPSOFT'97, LNCS 1214, 1997.

Афанасьев М.К.

Экспериментальная проверка применимости ГА для нахождения близкого к оптимальному расположения блоков данных в вычислительной сети.

Современное состояние сетей и подсетей Интернет позволяет создавать глобальные распределенные хранилища данных и использовать их для решения большого числа прикладных задач, в частности для сбора данных научных экспериментов, обмена медиа-данными и решения задач электронной коммерции.

В подобных хранилищах из-за неэффективности доступа к данным часто при передаче информации образуются значительные задержки. В связи с этим разработка программных средств для повышения скорости передачи информации между узлами гетерогенных распределенных систем с большим количеством отдельных хранилищ данных является актуальной проблемой [1].

В [2,3] предлагается генетический алгоритм (ГА) нахождения близкого к оптимальному расположения блоков данных, позволяющий учитывать как внешние, так и вызываемые перераспределением блоков данных, изменения параметров сети. Формально задача имеет вид:

$$\sigma^* = \arg\min_{\sigma \in \Sigma} F(\sigma, \alpha)$$

$$\textit{при условии} \quad \alpha = L(\sigma)$$

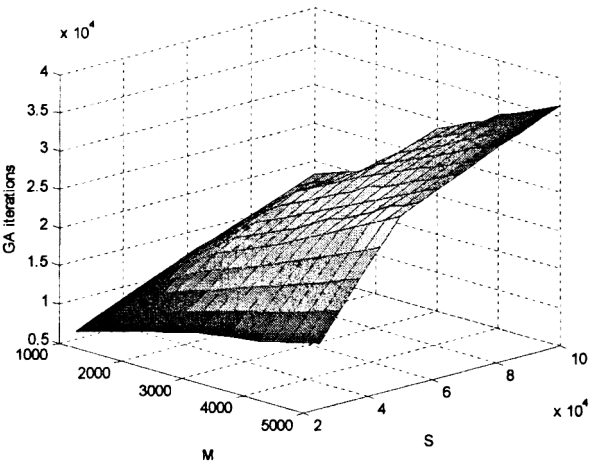
где α - матрица эффективных скоростей передачи данных между узлами, а σ задает расположение блоков данных по узлам сети. Так как расположение блоков данных по сети влияет на скорости передачи, $L(\sigma)$ для каждого расположения данных σ задает соответствующие эффективные скорости передачи α. $F(\sigma,\alpha)$ есть мера эффективности работы сети. В частном случае, $F(\sigma,\alpha)$ является суммарной средней задержкой передачи данных в сети.

Задача была рассмотрена в двух постановках – без поддержки и с поддержкой дублирования блоков данных. Ситуация отсутствия поддержки дублирования типична для финансовых и конфиденциальных данных. Дублирование поддерживается при передаче видео и аудио информации. При кодировании хромосом ГА для случая без дублирования информации, гены соответствуют блокам данных, и их значения – номерам узлов на которых они хранятся. Для случая с дублированием, гены соответствуют узлам, а их значения –

набору блоков данных, которые хранятся на узле. Скрещивание представляет собой одноточечный кроссовер и мутация – случайную замену значений ген при ограничении на внестимость узлов. Функция качества есть мера эффективности работы сети F. Использование ГА позволяет в динамике модифицировать функцию качества и допустимость решений с целью учета изменений параметров сети.

Для оценки работоспособности подхода было проведено 3 типа модельных экспериментов:

- данные с небольшим числом узлов и блоков данных. При небольшом числе узлов и блоков данных возможно полным перебором найти точное решение, и целью данного эксперимента являлась проверка того, насколько близко получаемое алгоритмом решение к точному. Было проведено около 20 наборов параметров, и на всех полученное решение совпадало с точным.
- модельные данные максимально приближенных к данным по CDN-сетям. Целью данного эксперимента являлась оценка масштабируемости алгоритма. CDN-сети (Content Distribution Networks) имеют до 6000 серверов и может быть выделено до 100000 отдельно хранимых крупных блоков данных. Были проведены тестовые эксперименты с числом серверов от 1000 до 5000 и с числом блоков данных от 25000 до 100000 и была построена зависимость количества итераций ГА от этих параметров (скорость выполнения одной итерации зависит от используемой аппаратной и программной среды. Умножение количества итераций на скорость дает суммарное время выполнения алгоритма для данной среды). Во всех экспериментах использовалась мягкая схема естественного отбора с размером популяции 100, количеством мутаций на каждой итерации 65, и количеством операций скрещивания 35. Данные о задаче (α,β, V_{total},W) генерировались случайным образом. Проверка показала, что алгоритм хорошо масштабируется – количетво итераций увеличивается с ростом числа узлов (M) и блоков (S) практически линейно. Картинка построена для случая отсутствия дублирования блоков данных



- динамически меняющиеся данные о сети. Целью данного эксперимента являлась проверка того, насколько быстро алгоритм адаптируется к изменениям состояния сети. При этом добавлялся новый узел и оценивалось количество итераций для нахождения оптимума новой задачи в случаях:
  - в начальной популяции присутствует регенерированный оптимум предыдущей задачи
  - начальная популяция сгенерирована случайным образом

Было показано, что при использовании регенерированного оптимума в качестве начальной популяции, число итераций ГА уменьшается по сравнению со случаем, когда начальная популяция сгенерирована случайным образом

Как показали модельные эксперименты, система является работоспособной и целесообразно ее применение в сетях оперирующих крупными "стабильными" блоками данных, таких как CDN-сети и локальные сети с большим количеством баз данных.

В качестве экспериментально-практической реализации алгоритма, была рассмотрена его работа в локальной сети одного из крупных университетов США. Сеть включала порядка 600 узлов, из них 300-400 соотетствовали индивидуальным компьютерам. Остальные соответствовали беспроводным роутерам, устройствам IP телефонии, телевизионным устройствам типа TiVo и т.д. На большинстве узлов была установлена та или иная версия Windows (как правило, Windows XP Home или Windows XP Professional). Вторыми по популярности системами являлись различные версии Macs. Количество иных операционных систем и трафик инициируемый ими были

незначительны. Сеть включала 7 серверов, хранящих научные данные, аудио и видео информацию, а также временные данные из сети Интернет. Сеть состояла из нескольких подсетей, выделенных в зависимости от используемых оборудования и каналов связи, географического местоположения и т.д. Пропускная способность большинства подсетей была 100 Megabit. Основными пользователями сети являлись студенты graduate level (магистры и аспиранты), в связи с чем значительная доля трафика соответствовала запросам на получение научной/предметной информации. По числу школ/факультетов можно было выделить порядка 10 различных категорий пользователей (инженеры, бизнес-студенты, медики и т.д.). Студенты одного факультета, как правило, пользуются данными из своей предметной области (или даже подобласти). В связи с этим возникла гипотеза, что перераспределение данных может сократить задержки в сети. Было принято решение использовать предложенный алгоритм.

Для доступа в Интернет, каждый пользователь обязан себя идентифицировать и зарегистрировать MAC адрес своего компьютера. При этом компьютеру выделяется собственный статический IP адрес в зависимости от того, где живет студент. Выделенный IP адрес и то, какому студенту он был выделен, заносятся в специальную базу данных. Поскольку стандартный жилищный контракт подписывался на 12 месяцев (начиная с сентября), большинство студентов меняют жилье (и IP адрес) лишь раз год.

На 7 серверах с данными были установлены программы, которые в течении нескольких недель по IP адресам отслеживали частоту средние запросов пользователей и скорость передачи от сервера к отдельному узлу. На основании полученной информации удалось выделить порядка 500 блоков данных, для каждого пользователя – среднюю частоту обращения к тому или иному блоку данных, а также скорости передачи данных от пользователей к серверам.

Полученные данные были переданы в ГА. Время работы ГА на Pentium 4, 3.4 ГГц (вычислительный сервер одного из факультетов) составило менее 30 минут. На основании полученных результатов, данные были перераспределены между серверами. Как и ожидалось, перераспределение данных привело к изменению скоростей доступа пользователей к серверам. На основании собранной новой статистики было произведено повторное перераспределение данных. Подобная процедура была проведена 3 раза. Все перемещения данных производились ночью, когда сети были минимально загружены. На четвертый раз алгоритм показал, что имеющееся распределение близко к оптимальному. По сравнению с изначальным распределением было перемещено 30% блоков данных и суммарная скорость работы сети возрасла на 15%.

Модельные эксперименты и практическая реализация продемонстрировали работоспособность предлагаемого подхода и его применимость для ряда сетей, в частности, для CDN сетей.

Литература

[1] L.W. Dowdy, D.V. Foster Comparative Models of the File Assignment Problem, Computing Surveys, Vol. 14, № 2, 1982, pp 287-313

[2] Афанасьев М.К. Алгоритм динамического распределения данных на основании анализа запросов узлов // Программные системы и инструменты. Тематический сборник № 6, Издательский отдел факультета ВмиК МГУ, Москва, 2005

[3] Афанасьев М.К. Динамическое распределение данных по сети хранения на основании анализа запросов узлов. // Материалы VI Международной научно-технической конференции Новые информационные технологии и системы, Информационно-издательский центр ПГУ, Пенза, 2004, стр 182-185

Раздел V
Инструментальные средства и сообщения

Веселов Н. А., Головин С. И.

Об одной системе автоматического построения твердотельной модели объекта по трем ортографическим проекциям

Введение

В последнее время широкое распространение получили системы автоматизированного проектирования (САПР), которые позволяют существенно ускорить процесс создания деталей и дают возможность проводить их всесторонний анализ, оперируя с трехмерными твердотельными моделями объектов. Однако переход к использованию таких систем затрудняет то, что многие организации работают с инженерными чертежами и не могут себе позволить отказаться от накопленных баз данных. Ручной перевод чертежей в трехмерный вид очень трудоемок и требует высокой квалификации персонала, поэтому в настоящий момент проблема автоматизированного перевода стоит достаточно остро.

Впервые проблему построения трехмерных твердотельных моделей объектов по инженерным чертежам поднял Idesawa в 1973 году [1]. С тех пор было предложено множество подходов к её решению. Наибольшее распространение получили методы, которые в качестве входной информации используют три ортографические проекции (эти проекции принято называть верхним, фронтальным и боковым видом). Эти методы можно разделить на каркасно ориентированные (B-Rep: boundary representation) и объемно-ориентированные (CSG: constructive solid geometry), рассмотрим их более подробно.

Суть каркасно ориентированного подхода заключается в построении, так называемой, каркасной модели, которая включает в себя только трехмерные ребра, последующем нахождении граней модели и составлении из них самой твердотельной модели. Выделяют следующие шаги: построение каркасной модели, построение граней и построение твердотельной модели.

Объемно ориентированный подход основывается на предположении, что объект можно построить из конечного числа примитивов заданной формы, используя булевы операции.

В настоящей статье рассматривается каркасно-ориентированная система построения твердотельной модели объекта по

трем ортографическим проекциям. В ней реализован новый метод создания каркасной модели, учитывающий экстремальные точки без обработки центральных линий [2, 3], которые далеко не всегда присутствуют на чертежах, и содержит ряд нововведений, которые ускоряют построение каркасной модели. Так же в статье представлено концептуальное описание архитектуры разработанной системы и проведено обоснование выбора используемых программных средств.

1. Применяемые алгоритмы и методы решения

Настоящий раздел посвящен рассмотрению алгоритмов и методов, реализованных в системе. Как было отмечено выше, эти алгоритмы относятся к классу каркасно-ориентированных подходов к решению задачи построения твердотельной модели. Данное рассмотрение необходимо вследствие того, что существует огромное количество методов и их вариаций, позволяющих решить обозначенную задачу даже в указанном классе [1 - 5]. Кроме того, при описании применяемых в системе алгоритмов будет дано обоснование сделанного нами выбора.

1.1. Построение каркасной модели

Все каркасно-ориентированные методы включают в себя следующие этапы построения каркасной модели: 1) построение трехмерных вершин; 2) нахождение связанных трехмерных вершин; 3) построение трехмерных ребер.

Нахождение связанных трехмерных вершин заключается в получении набора ребер на каждом ортографическом виде, каждый набор состоит из ребер с одинаковыми уравнениями и крайние вершины этих наборов имеют одинаковые проекции на общие оси.

При таком подходе происходит перебор всех двумерных вершин на ортографических видах на первом этапе, и нахождение путей между вершинами на втором этапе. Кроме того, в существующих работах либо не применяется метод экстремальных точек, либо требуется дополнительная информация в виде центральных линий. В рассматриваемой системе реализован подход, который устраняет этот недостаток и не содержит этапа построения трехмерных вершин. Он состоит из следующих шагов: 1) построение связей между ребрами; 2) нахождение связанных троек ребер; 3) построение трехмерных ребер. Опишем их более подробно.

1.1.1. Построение связей между ребрами

Два ребра называются связанными, если одна из их проекций на общую ось содержит другую. На этом этапе определяются все связи всех ребер на каждом из видов. Алгоритм состоит из 3 шагов.

Шаг 1: Если есть нерассмотренная пара видов, то пометить её как рассмотренную текущую и первый вид обозначить V_1, второй V_2, общую ось обозначить **axe**, перейти на шаг 2. В противном случае все связи между элементами ортографических видов построены. Окончание алгоритма.

Шаг 2: Спроектировать все ребра с текущей пары видов на ось axe. Полученные множества проекций обозначить $E(V_1)$ и $E(V_2)$, и упорядочить в них элементы по координате общей оси, перейти на шаг 3.

Шаг 3: Для каждого элемента из $E(V_1)$ выполнить: если этот элемент имеет общие точки с элементом из $E(V_2)$, то их прообразы связываются. Перейти на шаг 1.

1.1.2. Нахождение связанных троек ребер

После того, как все связи построены, можно определить связанные тройки ребер. Такие тройки состоят из ребер, каждое из которых принадлежит своему виду, и ребра попарно связаны. Каждая такая тройка определят три проекции некоторого трехмерного ребра на плоскости. Далее по трем проекциям восстанавливается само трехмерное ребро, используя метод, рассмотренный в предыдущем пункте. Алгоритм состоит из 2 шагов:

Шаг 1: Обозначим три вида как OXY, OXZ, OYZ. Для каждого элемента множества $E(OXY)$, обозначим его $curXY$, найти связанные с ним элементы на видах OXZ и OYZ, назовем их $E(OXY-OXZ)$ и $E(OXY-OYZ)$. Перейти на шаг 2.

После того, как перебрали все элементы $E(OXY)$, каркасная модель построена.

Шаг 2: Если среди $E(OXY-OXZ)$ и $E(OXY-OYZ)$ есть связанная пара, то построить трехмерное ребро по трем проекциям (связанная пара + $curXY$) и добавить его в каркасную модель.

На этом шаге можно столкнуться с ситуацией, когда проекции накладываются не полностью. В этом случае все проекции обрезаются таким образом, чтобы взаимное наложение было полным, и уже по ним строится трехмерное ребро.

1.2. Выделение граней

Каркасная модель в явном виде не содержит информации о гранях, из которых будет состоять итоговая модель, данный этап призван выделить её. Для решения этой задачи в системе был

реализован метод MIA, впервые предложенный в работе [4]. Его суть состоит в нахождении пространственных циклов ребер каркасной модели. Если уравнения всех ребер цикла удовлетворяют уравнению некой поверхности второго порядка, то такой цикл задает грань, контуром которой он является.

1.3. Построение твердотельной модели

После того как сформирован набор граней, из которых будет состоять итоговая модель, нужно определить их ориентацию и, возможно, удалить некоторые грани, которые были ошибочно построены на предыдущем этапе, что было вызвано недостаточностью использования лишь трех ортографических проекций для однозначного восстановления произвольной модели.

В основу реализованного метода положен алгоритм, описанный в работе [5], к которому был добавлен набор эвристических правил, призванный ускорить его. Алгоритм состоит из следующих шагов:

1) нахождение минимальных объемов, 2)нахождение ложных объемов и 3) построение твердотельной модели.

1.3.1. Нахождение минимальных объемов

Под минимальным объемом понимается область пространства, ограниченная неким замкнутым контуром, составленным из граней и не содержащим внутри себя другие грани.

Целью данного этапа является нахождение всех минимальных объемов, из которых состоит модель. Для их нахождения используется следующая идея: берется произвольная грань из набора, полученного на предыдущем этапе, и в ней фиксируется направление нормали. Назовем эту грань базовой. По этой нормали определяется направление обхода ребер грани по правилу буравчика. Далее перебираются все ребра и для каждого ишутся грани, которые образуют минимальный угол с базовой гранью и пересекаются с ней по этому ребру.

1.3.2. Нахождение ложных объемов

Ложным называется объем, которого нет в модели, но который был построен на предыдущем этапе, другие объемы будем называть истинными. Для их нахождения используется метод, предложенный в работе [5]. Под итоговой моделью понимается некое множество построенных на предыдущем этапе элементарных объемов. Это множество не единственно, так как по трем ортографическим проекциям не всегда возможно восстановить твердотельную модель, но предложенный подход гарантирует то, что будут построены все модели, соответствующие чертежам. Не единственность решений вызвана тем,

что не используется информация о скрытых линиях (линии, которые лежат внутри объекта), не обрабатываются разрезы и сечения.

1.3.3.Построение твердотельной модели

После предыдущего этапа имеется множество наборов объемов, помеченных либо как истинные, либо как ложные, каждый такой набор отвечает некоторой итоговой модели. Для построения твердотельных моделей используются булевы операции. Сначала все истинные объемы добавляются в строящуюся твердотельную модель, а потом из них вырезаются все ложные объемы. Для выполнения этих операций использовались функции библиотек OpenCascade [6].

2. Концептуальное описание системы

При разработке системы была поставлена задача разработать программное средство, которое бы позволяло загружать чертежи, представленные в одном из промышленных стандартов, строить по ним твердотельные модели и использовать их в современных системах трехмерного моделирования. Можно выделить три программных модуля: 1) модуль загрузки и работы с dxf-форматом, 2) модуль построения твердотельной модели, 3) модуль графического интерфейса и визуализации.

Первый и третий модули были реализованы на основе уже существующих программных решений. Одним из основных факторов при их выборе была доступность исходных кодов. Модуль построения твердотельной модели является оригинальным.

2.1. Модуль загрузки и работы с dxf-форматом

Задачей этого модуля является загрузка dxf-файла во внутрениий формат. В его основу положена свободно распространяемая библиотека dxf-lib компании Ribbonsoft [7]. Она позволяет загружать dxf-файлы в некоторый внутренний формат и проводить различные манипуляции с полученными сущностями. Библиотека позволяет работать со следующими геометрическими объектами: точки, векторы, дуги окружностей и эллипсов. Этот набор объектов определяет класс ортографических проекций, которые могут быть обработаны системой.

2.2. Модуль построения твердотельной модели

Этот модуль занимается непосредственно построением каркасной и, далее, твердотельной модели объекта, результат сохраняется во внутреннем формате, унаследованном от dxf-lib. Модуль делится на три части: 1) предобработка информации; 2) построение каркасной модели; 3) построение твердотельной модели.

Предполагается, что на вход системе подается dxf-файл, в котором содержатся три ортографические проекции, разделенные двумя перпендикулярными линиями. Первой задачей этого этапа является определение принадлежности сущности тому или иному виду, кроме того, если на чертеже содержатся примитивы, отличные от точек, векторов и дуг окружностей и эллипсов, то они игнорируются.

Далее на каждом ортографическом виде определяются экстремальные точки дуг окружностей и эллипсов и разбиваются ими на элементырные дуги, которые добавляются в ортографический вид, при этом базовые дуги удаляются. Кроме того, все сущности аналогичным образом разбиваются точками пересечения. Такая обработка информации позволяет системе не зависеть от способа задания чертежа и дает возможность применить метод экстремальных точек.

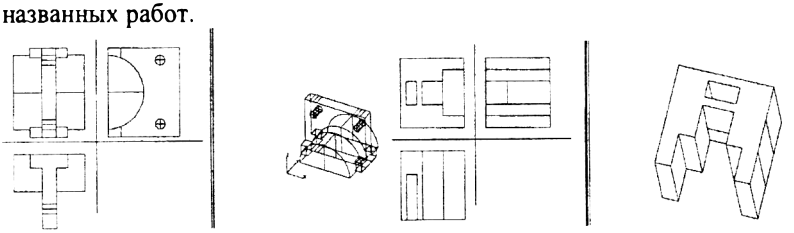
После этого по обработанным ортографическим видам строится каркасная модель и по ней итоговая твердотельная.

2.3. Модуль графического интерфейса и визуализации

Твердотельные модели, полученные на предыдущем этапе, хранятся во внутреннем формате. Для того, чтобы их визуализировать и перевести в один из промышленных стандартов (IGES, STEP), применяется данный модуль. Он основан на системе OpenCascade. Эта система с открытыми исходными текстами предназначена для визуализации и обработки различных графических объектов как на плоскости, так и в пространстве. Кроме того, в ней содержится ряд библиотек, которые прозволяют работать с основными форматами хранения графической информации и осуществлять перевод из одного формата в другой.

3. Тестирование системы

В ходе экспериментов системой были построены все модели по содержащимся в статьях [1-5] чертежам. Такой выбор тестовых данных обусловлен тем, что положенные в основу системы алгоритмы взяты из названных работ.



4. Заключение

В настоящей работе была рассмотрена система автоматического построения твердотельной модели объекта по трем ортографическим

проекциям. Она состоит из трех модулей: модуль загрузки и работы с dxf-форматом, модуль построения твердотельной модели, модуль графического интерфейса и визуализации. В основу первого модуля положена свободно распространяемая библиотека dxf-lib компании Ribbonsoft. Второй модуль является оригинальным. В нем реализован каркасно-ориентированный метод построения твердотельной модели объекта, содержащий новый подход к построению каркасной модели, позволяющий избежать этапа построения трехмерных вершин и использовать метод экстремальных точек без требования присутствия на ортографических видах центральных линий дуг окружностей и эллипсов. Третий модуль основан на системе OpenCascade и предназначен для визуализации построенных моделей и их экспортированию в отраслевые стандарты хранения твердотельных моделей IGES и STEP.

Разработанная система позволяет осуществлять автоматический перевод чертежей, представленных в формате dxf, в твердотельные модели, которые могут быть сохранены в форматах IGES и STEP. Перечисленные форматы являются промышленными стандартами и поддерживаются всеми современными системами моделирования.

Литература

1. Idesawa Masanori. A system to generate a solid figure from three views // Bull JSME. 1973.
2. Shi-Xia Liu, Shi-Min Hu, Jia-Guang Sun. A Matrix-Based Approach to Reconstruction of 3D Objects from Three Orthographic Views // Proceedings of the 8th Pacific Conference on Computer Graphics and Applications. 2000.
3. Shi-Xia Liu, Shi-Min Hu, Yu-Jian Chen, Jia-Guang Sun. Reconstruction of curved solids from engineering drawings // Computer-Aided Design. 2001. 33. P. 1059-1072.
4. Kuo M H. Reconstruction of quadric surface solids from three-view engineering drawings // Computer-Aided Design. 1998. 30. P. 517-527.
5. Shin B, Shin Y. Fast 3D solid model reconstruction from orthographic vies // Computer Aided Design. 1998. P. 171-179.
6. Open CASCADE, simulation integrator [Электронный ресурс] - http://opencascade.com
7. RibbonSoft, open source C++ library [Электронный ресурс] - http://ribbonsoft.com/

Аннотации

Махнычев В.С., Ильичев А.Б. Распараллеливание задачи о составлении вузовского расписания с использованием распределенного обхода дерева перебора // Программные системы и инструменты. Тематический сборник № 7, М.: Изд-во факультета ВМиК МГУ, 2006.

Решение задачи о составлении расписания затруднено огромным количеством различных вариантов, из которых предстоит сделать выбор. Поэтому для решения задачи применяют в основном эвристические алгоритмы с небольшими элементами перебора. В данной работе предпринята попытка улучшить качество результатов, увеличив возможности перебора за счет использования распределенной системы. Примененная схема параллельного обхода дерева перебора была ранее испытана на программе игры в шахматы.

Для демонстрации эффективности описанных методов использовались различные тестовые наборы, приближенные к реальной задаче для отдельно взятого факультета.

Библиогр.: 2 назв.

Истомин Т.Е., Пыптев С.А. Технология конфигурирования программных систем // Программные системы и инструменты. Тематический сборник № 7, М.: Изд-во факультета ВМиК МГУ, 2006.

В статье описывается технология конфигурирования программных систем перед компиляцией исходных кодов. Ее возможности показаны на примерах из опыта разработки операционной системы для микроконтроллеров.

Библиогр.: 2 назв.

Смирнов А.А. Методы автоматической расстановки консистентных контрольных точек. // Программные системы и инструменты. Тематический сборник № 7, М.: Изд-во факультета ВМиК МГУ, 2006.

Данная статья посвящена вопросам анализа параллельных программ с целью расстановки консистентных контрольных точек. Рассматриваются основные подходы к реализации систем КТ, приводится описание одной из реализаций таких систем, библиотеки libcheckpoint. Описываются два различных метода поиска консистентных контрольных точек: статический анализ трассы параллельной программы и динамический алгоритм поиска мест создания КТ во время работы параллельной программы, приводится сравнение данных методов.

Библиогр.: 7 назв.

Горицкая В.Ю. Алгоритмы планирования в многопроцессорных системах // Программные системы и инструменты. Тематический сборник № 7, М.: Изд-во факультета ВМиК МГУ, 2006.

Для эффективного распределения ресурсов многопроцессорных вычислительных систем (процессорных мощностей, времени) обычно используются системы управления заданиями (такие, как LoadLeveler, Sun Grid Engine и т. д.). Из-за появления новых аппаратных платформ (многоядерные системы), а так же в результате того, что архитектуры вычислительных систем становятся все более сложными, неоднородными (кластеры), насчитывающими сотни и тысячи процессорных элементов, которые могут быть распределены географически (grid-системы), необходима разработка новых подходов к планированию пользовательских задач.

В данной работе приводится обзор алгоритмов планирования процессов; рассматривается, как эти алгоритмы используются в реальных системах, управляющих очередьми задач на различных вычислителях.

Ил.: 4

Библиогр.: 17 назв.

Герасимов С.В., Качуровский М.А., Киреев А.А. Технология создания предметно-ориентированных пользовательских интерфейсов // Программные системы и инструменты. Тематический сборник № 7, М.: Изд-во факультета ВМиК МГУ, 2006.

В статье рассматриваются проблемы проектирования предметно-ориентированных пользовательских интерфейсов информационных систем. Предлагается технология автоматизированной разработки пользовательских интерфейсов систем класса электронная история болезни (ЭИБ). Особенностями систем ЭИБ являются большой объем и разнородность обрабатываемых данных и, как следствие, необходимость создания значительного числа стандартных экранных форм и проектирования специализированных пользовательских интерфейсов. На базе предложенной технологии создан пользовательский интерфейс ЭИБ клиники неотложной кардиологии НИИ СП им. Н.В. Склифосовского.

Ил.: 6

Библиогр.: 4 назв.

Попова Е.А. Методы построения ансамблей деревьев решений // Программные системы и инструменты. Тематический сборник № 7, М.: Изд-во факультета ВМиК МГУ, 2006.

Одним из широко распространенных методов статистической обработки данных является метод деревьев решений и ансамблей деревьев решений.

В данной работе проведен обзор методов и алгоритмов построения ансамблей деревьев решений, анализ их применения к решению прикладных задач, выделены основные проблемы существующих на данный момент подходов.

Библиогр.: 31 назв.

Глазкова В. В., Петровский М.И. Дообучаемый метод классификации многотемных документов для анализа и фильтрации интернет информации // Программные системы и инструменты. Тематический сборник № 7, М.: Изд-во факультета ВМиК МГУ, 2006.

В настоящей статье рассматривается задача классификации многотемных документов в рамках проблемы анализа и фильтрации интернет информации. Авторами был предложен метод классификации многотемных документов с возможностью дообучения с использованием только новых обучающих документов. До настоящего момента метода классификации многотемных документов с такой возможностью дообучения не существовало. Эксперименты с разработанным методом на эталонном тестовом наборе показали высокую точность классификации многотемных документов.

Ил.: 1

Табл.: 1

Библиогр.: 13 назв.

Курынин Р.В., Машечкин И.В., Петровский М.И. Об одном методе нечетких деревьев решений в задаче анализа и прогнозирования качества продукции в производственном процессе // Программные системы и инструменты. Тематический сборник № 7, М.: Изд-во факультета ВМиК МГУ, 2006.

В настоящей статье рассматривается задача анализа и прогнозирования качества продукции в производственном процессе. Авторами была предложена модификация метода нечетких деревьев решений, позволяющая при построении нечеткого дерева решений проводить разбиение как по дискретным, так и по непрерывным атрибутам, а также обрабатывать пропущенные значения. Эксперименты с предложенным методом показали высокую точность по сравнению с результатами тестирования других популярных методов.

Ил.: 1

Табл.: 3

Библиогр.: 11 назв.

Машечкин И.В., Петровский М.И., Трошин С.В. Система мониторинга и анализа поведения пользователей компьютерной системы // Программные системы и инструменты. Тематический сборник № 7, М.: Изд-во факультета ВМиК МГУ, 2006.

В статье рассматриваются вопросы построения эффективных программных систем защиты от внутренних вторжений, основанных на несигнатурных методах и обладающих свойствами автономности, адаптируемости и самообучаемости. Отдельно рассмотрены проблемы консолидации исходных данных из журналов регистрации и протоколов

ОС, методы промежуточного представления, передачи данных и хранения собранных данных. Предложена архитектура системы консолидации и рабочего места аналитика безопасности. Предложены методы применения технологии OLAP для анализа собранных данных об активности пользователей, а также алгоритмы интеллектуального анализа данных (Data Mining) для построения модели поведения пользователя на основе ассоциативных правил. Построенная модель поведения может быть использована для визуального представления аналитику безопасности в виде сети зависимостей, а также для автоматического поиска аномалий в поведении пользователей и оценки степени потенциальной угрозы, исходящего от каждого пользователя. Реализована экспериментальная пилотная версия такой системы, которая была верифицирована по методике DARAP Intrusion Detection Evaluation Program, с использованием эталонных наборов данных. Результаты экспериментальной верификации приведены в статье.

Ил.: 10

Таб.: 2

Библиогр.: 11 назв.

Курынин Р.В., Машечкин И.В., Петровский М.И. О некоторых методах интеллектуального анализа данных для мониторинга технологических процессов // Программные системы и инструменты. Тематический сборник № 7, М.: Изд-во факультета ВМиК МГУ, 2006.

Целью настоящей работы является проведение исследований по применению методов Data Mining в прикладной задаче мониторинга ТП и выявления нештатных ситуаций. Для решения данной задачи было реализовано экспериментальное программное средство, предназначенное для выявления аномалий в ТП, имеющее возможность определять различные модели выявления аномалий и «подключать» новые алгоритмы обнаружения аномалий, чтобы в дальнейшем при эксплуатации в реальных условиях можно было выбирать и настраивать наиболее подходящий алгоритм. Прототип реализован для работы в реальной промышленной среде управления и мониторинга ТП в среде WinCC.

Ил.: 5

Табл.: 9

Библиогр.: 12 назв.

Смелянский Р.Л., Шалимов А.В. Метод оценки частот выполнения фрагментов кода последовательной программы // Программные системы и инструменты. Тематический сборник № 7, М.: Изд-во факультета ВМиК МГУ, 2006.

В статье предложен метод анализа частотных характеристик поведения программы на основе функций распределения ее входных

параметров. Дан обзор основных и предложены новые методы определения редко выполняемых частей кода программы.

Библиогр.: 8 назв.

Балашов В.В., Ларионов А.А. Верификация имитационных моделей в среде ДИАНА с применением средства SPIN // Программные системы и инструменты. Тематический сборник № 7, М.: Изд-во факультета ВМиК МГУ, 2006.

В статье представлены результаты работ по разработке ядра системы верификации в среде имитационного моделирования ДИАНА на основе программного средства SPIN. Разработанное ядро существенно повышает производительность системы верификации и позволяет применять его на практических задачах верификации имитационных моделей. Обоснована корректность предложенной схемы трансляции исходного текста имитационной модели во входной язык SPIN. Приведены результаты экспериментального исследования производительности системы верификации.

Ил.: 2

Табл.: 3

Библиогр.: 11 назв.

Волканов Д.Ю., Григорян М.В. Методика поиска оптимального набора механизмов отказоустойчивости для бортовых вычислительных систем // Программные системы и инструменты. Тематический сборник № 7, М.: Изд-во факультета ВМиК МГУ, 2006.

В статье предложена методика поиска оптимального набора механизмов отказоустойчивости для бортовых вычислительных систем, и приведён пример использования программного средства, реализующего эту методику.

Ил.: 2

Табл.: 1

Библиогр.: 10 назв.

Козлов Д.Д., Петухов А.А. Методы обнаружения уязвимостей в web-приложениях // Программные системы и инструменты. Тематический сборник № 7, М.: Изд-во факультета ВМиК МГУ, 2006.

В статье рассматривается задача автоматического поиска уязвимостей в web-приложениях. В статье рассматриваются основные подходы, применяемые для решения задачи, анализируются достоинства и недостатки этих подходов.

Библиогр.: 12 назв.

Афанасьев М.К. Экспериментальная проверка применимости ГА для нахождения близкого к оптимальному расположения блоков данных в вычислительной сети // Программные системы и инструменты. Тематический сборник № 7, М.: Изд-во факультета ВМиК МГУ, 2006.

В современных хранилищах данных часто возникает ситуация при которой информация оказывается распределена по узлам сети неэффективно. Как результат, при обращении к данным конечными пользователями задержки оказываются очень большими. В итоге возникает задача перераспределения данных по узлам сети с целью минимизации задержек. В работе проводится проверка применимости генетических алгоритмов для решения данной задачи на трех типах модельных экспериментов и на практической реализации в локальной университетской сети.

Ил.: 1

Библиогр.: 3 назв.

Веселов Н.А., Головин С.И. Об одной системе автоматического построения твердотельной модели объекта по трем ортографическим проекциям // Программные системы и инструменты. Тематический сборник № 7, М.: Изд-во факультета ВМиК МГУ, 2006.

В настоящей работе описана система построения твердотельной модели объекта по трем ортографическим проекциям, состоящим из прямых линий и дуг окружностей и эллипсов. Данная задача возникает при переходе от плоских чертежей к пространственным моделям объектов в системах трехмерного твердотельного моделирования (Solid Works, Solid Edge и др.). Несмотря на большое количество подобных систем, в них до сих пор не реализовано автоматическое решение описанной задачи. Это связано с тем, что методы решения, существующие в настоящий момент, являются несовершенными и требуют дальнейшей разработки. Кроме того, при описании систем авторы уделяли внимание лишь используемым подходам и алгоритмам, и опускали такие аспекты как формат и способ загрузки чертежей, методы визуализации результатов и возможности их использования в существующих системах трехмерного моделирования. В этой статье авторы попытались исправить этот недостаток.

В данной работе производится описание разработанной системы автоматического построения твердотельной модели объекта по трем ортографическим проекциям, применяемых в ней алгоритмов и методов.

Ил.: 2

Библиогр.: 7 назв.