ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г. DOI 10.17587/issn.2220-3397 ISSN 2220-3397

Редакционный совет
Садовничий В.А., акад. РАН (*председатель*)
Бетелин В.Б., акад. РАН
Васильев В.Н., чл.-корр. РАН
Жижченко А.Б., акад. РАН
Макаров В.Л., акад. РАН
Панченко В.Я., акад. РАН
Стемпковский А.Л., акад. РАН
Ухлинов Л.М., д.т.н.
Федоров И.Б., акад. РАН
Четверушкин Б.Н., акад. РАН

Главный редактор
Васенин В.А., д.ф.-м.н., проф.

Редколлегия
Антонов Б.И.
Афонин С.А., к.ф.-м.н.
Бурдонов И.Б., д.ф.-м.н., проф.
Борзовс Ю., проф. (Латвия)
Гаврилов А.В., к.т.н.
Галатенко А.В., к.ф.-м.н.
Корнеев В.В., д.т.н., проф.
Костюхин К.А., к.ф.-м.н.
Махортов С.Д., д.ф.-м.н., доц.
Манцивода А.В., д.ф.-м.н., доц.
Назиров Р.Р., д.т.н., проф.
Нечаев В.В., д.т.н., проф.
Новиков Б.А., д.ф.-м.н., проф.
Павлов В.Л. (США)
Пальчунов Д.Е., д.ф.-м.н., доц.
Петренко А.К., д.ф.-м.н., проф.
Позднеев Б.М., д.т.н., проф.
Позин Б.А., д.т.н., проф.
Серебряков В.А., д.ф.-м.н., проф.
Сорокин А.В., к.т.н., доц.
Терехов А.Н., д.ф.-м.н., проф.
Филимонов Н.Б., д.т.н., проф.
Шапченко К.А., к.ф.-м.н.
Шундеев А.С., к.ф.-м.н.
Щур Л.Н., д.ф.-м.н., проф.
Язов Ю.К., д.т.н., проф.
Якобсон И., проф. (Швейцария)

Редакция
Лысенко А.В., Чугунова А.В.

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э. Баумана

СОДЕРЖАНИЕ

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в любом почтовом отделении (индексы: по каталогу агентства "Роспечать" — **22765**, по Объединенному каталогу "Пресса России" — **39795**) или непосредственно в редакции.
Тел.: (499) 269-53-97. Факс: (499) 269-55-10.
Http://novtex.ru/prin/rus E-mail: prin@novtex.ru
Журнал включен в систему Российского индекса научного цитирования.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

© Издательство "Новые технологии", "Программная инженерия", 2016

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

Vol. 7
N 10
2016

Published since September 2010 **DOI 10.17587/issn.2220-3397** **ISSN 2220-3397**

Editorial Council:

SADOVNICHY V. A., Dr. Sci. (Phys.-Math.), Acad. RAS (*Head*)
BETELIN V. B., Dr. Sci. (Phys.-Math.), Acad. RAS
VASIL'EV V. N., Dr. Sci. (Tech.), Cor.-Mem. RAS
ZHIZHCHENKO A. B., Dr. Sci. (Phys.-Math.), Acad. RAS
MAKAROV V. L., Dr. Sci. (Phys.-Math.), Acad. RAS
PANCHENKO V. YA., Dr. Sci. (Phys.-Math.), Acad. RAS
STEMPKOVSKY A. L., Dr. Sci. (Tech.), Acad. RAS
UKHLINOV L. M., Dr. Sci. (Tech.)
FEDOROV I. B., Dr. Sci. (Tech.), Acad. RAS
CHETVERTUSHKIN B. N., Dr. Sci. (Phys.-Math.), Acad. RAS

Editor-in-Chief:

VASENIN V. A., Dr. Sci. (Phys.-Math.)

Editorial Board:

ANTONOV B.I.
AFONIN S.A., Cand. Sci. (Phys.-Math)
BURDONOV I.B., Dr. Sci. (Phys.-Math)
BORZOVS JURIS, Dr. Sci. (Comp. Sci), Latvia
GALATENKO A.V., Cand. Sci. (Phys.-Math)
GAVRILOV A.V., Cand. Sci. (Tech)
JACOBSON IVAR, Dr. Sci. (Philos., Comp. Sci.), Switzerland
KORNEEV V.V., Dr. Sci. (Tech)
KOSTYUKHIN K.A., Cand. Sci. (Phys.-Math)
MAKHORTOV S.D., Dr. Sci. (Phys.-Math)
MANCIVODA A.V., Dr. Sci. (Phys.-Math)
NAZIROV R.R. , Dr. Sci. (Tech)
NECHAEV V.V., Cand. Sci. (Tech)
NOVIKOV B.A., Dr. Sci. (Phys.-Math)
PAVLOV V.L., USA
PAL'CHUNOV D.E., Dr. Sci. (Phys.-Math)
PETRENKO A.K., Dr. Sci. (Phys.-Math)
POZDNEEV B.M., Dr. Sci. (Tech)
POZIN B.A., Dr. Sci. (Tech)
SEREBRJAKOV V.A., Dr. Sci. (Phys.-Math)
SOROKIN A.V., Cand. Sci. (Tech)
TEREKHOV A.N., Dr. Sci. (Phys.-Math)
FILIMONOV N.B., Dr. Sci. (Tech)
SHAPCHENKO K.A., Cand. Sci. (Phys.-Math)
SHUNDEEV A.S., Cand. Sci. (Phys.-Math)
SHCHUR L.N., Dr. Sci. (Phys.-Math)
YAZOV Yu. K., Dr. Sci. (Tech)

Editors: LYSENKO A.V., CHUGUNOVA A.V.

CONTENTS

Information about the jornal is available online at:
http://novtex.ru/prin/eng e-mail:prin@novtex.ru

УДК 004.4'416 DOI: 10.17587/prin.7.435-445

Н. И. Вьюкова, ст. науч. сотр., e-mail: niva@niisi.ras.ru,
В. А. Галатенко, д-р физ.-мат. наук, ст. науч. сотр., e-mail: galat@niisi.ras.ru,
С. В. Самборский, ст. науч. сотр., e-mail: sambor@niisi.ras.ru,
Федеральное государственное учреждение "Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук", Москва

Директивная и автоматическая векторизации циклов

Рассмотрены средства автоматической и директивной векторизаций программ. Возможности автоматической векторизации представлены на примере современных версий компилятора GCC. В качестве средств директивной векторизации использованы возможности стандарта OpenMP 4.0, реализованные в GCC, а также программа Scout, осуществляющая векторизацию программ на уровне исходного кода. Дана оценка преимуществ и ограничений отмеченных средств, предложен подход к их совместному применению.

Ключевые слова: *векторное расширение, Single Instruction Multiple Data (SIMD), векторизация, явное векторное программирование, Scout, OpenMP*

Введение

Векторные расширения присутствуют практически во всех современных микропроцессорных архитектурах. Сфера их применения постоянно расширяется. В настоящее время она включает приложения мультимедиа, различные области научно-технический вычислений, в том числе сегмент высокопроизводительных встраиваемых вычислений для оборонных, аэрокосмических и государственных проектов (MAG HPEC), а также приложения баз данных.

Использование векторных операций вместо скалярных представляет потенциальную возможность ускорения вычислений в N раз, где N — число элементов вектора. Однако выпускаемые в настоящее время процессоры не способны автоматически трансформировать скалярные вычисления в векторные. Предполагается, что применение векторных операций в приложениях должно обеспечиваться с помощью тех или иных программных средств. В работе [1] представлен краткий обзор средств и методов получения векторизованного кода и рассмотрены некоторые из них. В их числе:

- использование векторизованных библиотек;
- ручное векторное программирование на языке ассемблера;
- ручное векторное программирование на языках высокого уровня с использованием встроенных функций;
- автоматическая векторизация программ средствами компилятора.

Каждый из перечисленных выше подходов имеет свои сильные и слабые стороны. Использование высококачественных библиотек позволяет добиться высокой производительности при небольших трудозатратах, однако подходящая библиотека не всегда доступна. Очевидными недостатками ручного программирования являются высокая трудоемкость и сложность портирования кода. Использование векторизующего компилятора практически не требует усилий от разработчиков программного обеспечения. Однако как показано в работах [2, 3], результативность автоматической векторизации в современных компиляторах остается невысокой. Это обстоятельство связано, с одной стороны, с недостаточной развитостью методов анализа и трансформации программ, применяемых в современных компиляторах, с другой стороны, с различными ограничениями языковых стандартов, в рамках которых вынуждены действовать компиляторы.

В качестве решения компанией Intel была предложена концепция явного векторного программирования [4], нашедшая свое выражение в спецификациях CilkPlus [5], которые были реализованы в компиляторе icc для языков C/C++. Спецификации CilkPlus включают директивы векторизации, директивы описания элементных функций (имеющих скалярную и векторную версии), а также языковые расширения для оперирования с вырезками массивов. Эти возможности (за исключением операций с вырезками массивов) с некоторыми изменениями позднее были включены в стандарт OpenMP [6] версий 4.0 и выше. Поддержка CilkPlus и OpenMP 4.0 реализована в компиляторах icc, а также для некоторых процессоров в Clang/LLVM и GCC. Директивы позволяют программисту: явно указать, какие участки кода (циклы, функции) должны быть векторизованы, возможно

даже в обход языковых стандартов; сообщить компилятору дополнительную информацию, в частности, о выравнивании данных и о зависимостях.

Отдельный класс средств директивной векторизации составляют инструментальные средства векторизации, осуществляющие трансформацию программ на уровне входного языка: Scout [7, 8], SWARP [9] и др.

Дальнейшие разделы статьи построены по следующему плану. В разд. 1 обсуждены преимущества и ограничения инструментальных средств векторизации на уровне исходного кода. Разд. 2 содержит анализ и сравнение возможностей автоматической и директивной векторизации. Рассмотрены примеры циклов с различными особенностями. Приведено сравнение результатов их автоматической векторизации с помощью современных версий компилятора GCC и директивной векторизации на основе стандарта OpenMP, а также с помощью программы Scout. В заключении подведены итоги проведенного анализа, предложена возможная методика совместного использования указанных средств векторизации.

1. Инструментальные средства векторизации на уровне исходного кода

Средства автоматической или директивной векторизации могут быть встроены в компилятор или реализованы в виде отдельного инструментария. В последнем случае векторизация выполняется как трансформация на уровне исходного текста (*source-to-source*). Исходный текст на языке высокого уровня трансформируется в векторизованный текст, чаще всего на том же самом языке. Иногда инструментальные средства векторизации называют векторизующими препроцессорами, хотя это название вряд ли можно считать оправданным. Под препроцессированием обычно подразумевается только текстовая обработка, тогда как векторизация требует не только синтаксического, но и глубокого семантического анализа программы.

Векторизованный текст отличается от исходного тем, что скалярные вычисления в циклах заменяются векторными с использованием переменных векторного типа и вызовов встроенных функций компилятора, оперирующих с векторными значениями. Как правило, эти встроенные функции соответствуют инструкциям векторного расширения процессора. Следовательно, применение инструментальных средств векторизации требует поддержки со стороны компилятора. Компилятор должен поддерживать векторные типы данных и встроенные функции, соответствующие инструкциям векторного расширения.

Использование внешних инструментальных средств векторизации имеет два важных преимущества по сравнению с ручной векторизацией. Во-первых, директивная векторизация требует значительно меньших усилий от прикладного программиста, чем ручное программирование векторных операций. Во-вторых, внешние инструментальные средства векторизации предпочтительнее ручного программирования с точки зрения портируемости исходного кода. Указывая различные ключи при запуске инструментария векторизации, можно преобразовывать исходный невекторизованный код в векторизованный с использованием разных векторных расширений. Специфика векторного расширения выносится из исходного кода либо в инструментальные средства векторизации, либо, что лучше, в отдельное описание векторного расширения, задающее настройки инструментальных средств векторизации.

Тем не менее директивная векторизация не обеспечивает идеальной портируемости, поскольку директивы, задающие наилучшую векторизацию программы (в большей степени — оптимальные параметры директив, например, кратность развертки), могут отличаться для разных векторных расширений.

Важный частный случай портирования заключается в переносе программ на целевую платформу без векторных расширений. В случае использования директив векторизации портирование тривиально, так как компиляторы обычно просто игнорируют "чужие" директивы `#pragma`. Если же программа векторизована вручную, преобразование ее в невекторизованную может породить код менее эффективный, чем изначально невекторизованный.

Очевидным минусом инструментальных средств векторизации является тот факт, что они зачастую не позволяют добиться производительности, которая достигается при ручном векторном программировании. Отчасти это объясняется тем, что существующие методы автоматической и директивной векторизаций способны использовать лишь ограниченный набор векторных инструкций, в основном команды, являющиеся векторными аналогами скалярных операций. Программист же может обеспечить высокую производительность за счет использования специализированных команд, ориентированных на определенные классы алгоритмов.

На первый взгляд, нет большой разницы, интегрирована ли векторизация в компилятор или реализована в виде отдельного программного средства. Однако есть несколько существенных отличий.

Во-первых, успешная векторизация может требовать дополнительной трансформации программы. Даже в самом простом случае векторизуемый цикл, как правило, дополняется невекторизованным циклом, в который выносятся последние итерации исходного цикла, если число итераций не кратно длине вектора (*остаточный цикл*). Кроме того, для хорошей векторизации может возникать необходимость избавиться от условных операторов в цикле, выполнить inline-подстановку вызываемых в цикле функций, преобразовать гнездо вложенных циклов.

В компиляторах эти трансформации обычно уже присутствуют независимо от поддержки векторизации, но во внешнем инструментарии векторизации их приходится реализовывать для того чтобы обеспечить возможность векторизации. Тем самым отчасти размывается граница между автоматическими

и директивными инструментальными средствами, так как директивные средства векторизации могут автоматически выполнять вспомогательные преобразования, а автоматические, наоборот, потребовать явной директивы для выполнения неочевидного преобразования, способствующего эффективности последующей векторизации. Кроме того, необходимость реализовывать в инструменте векторизации вспомогательные преобразования программы приводит к тому, что он становится универсальным инструментарием, выполняющим преобразования на уровне исходного кода, и на его основе могут выполняться и другие оптимизации (например, распараллеливание на несколько процессоров).

Во-вторых, векторизация, реализуемая в компиляторе, имеет преимущество по сравнению с векторизацией на уровне исходного кода, поскольку компилятор, преобразуя программу до исполняемого кода, может точнее оценить выигрыш от векторизации. Причем очень важно, что компилятор может оценить взаимодействие процесса векторизации с другими подходами к оптимизациям.

В-третьих, преобразование на уровне исходного кода, по сравнению с векторизацией в компиляторе, дает ряд дополнительных технологических возможностей, к числу которых относятся перечисленные далее.

- Изучение векторизованной программы с целью оценить успешность векторизации и то, какие векторные инструкции удалось задействовать.
- Отладка в терминах векторизованного текста. Отладка векторизованной программы в терминах исходного (невекторизованного) текста может оказаться совершенно лишена смысла.
- Профилирование векторизованной программы. Например, профилирование может потребоваться для оценки соотношения времени выполнения векторизованного цикла и остаточного цикла.
- Дальнейшая ручная оптимизация векторизованной программы.
- Ручное аннотирование векторизованной программы для эффективной отладки и профилирования.

Все это возможно только при условии, что выходной текст "читабелен" и понятен для программиста, а не только для компилятора. Результат преобразования на уровне исходного кода должен быть максимально подобен исходному тексту. Для этого необходимы:

- сохранение имен переменных и функций;
- "хорошие" имена для новых переменных, на основе имен соответствующих переменных в исходной программе;
- максимальное сохранение исходного форматирования (отступы, переносы строк, пробелы);
- сохранение комментариев;
- порождение комментариев при применениях преобразований, в частности, должны снабжаться комментариями порожденные циклы (остаточные циклы, циклы редукции, циклы, сгенерированные в результате вынесении ветвления, разбиения или слияния циклов во входной программе).

Определенные сложности связаны с использованием макросов. С одной стороны, если вычислительные операции находятся в макросах, то без выполнения макроподстановок нельзя получить код, пригодный для векторизации. С другой стороны, полностью выполнив макроподстановки можно получить текст, уже плохо пригодный для изучения и модификации программистом. Неочевидно, как следует отличать те макроподстановки, которые необходимо выполнить до векторизации.

Практический вывод заключается в том, что желательно минимально использовать макросы и другие директивы препроцессора в тексте, который предполагается векторизовать. Желательно также эти части исходного текста выносить в отдельные файлы.

В целом восстановление из трансформированного внутреннего представления текста на языке высокого уровня, максимально близкого к исходному, это задача непростая, но необходимая.

В качестве примера внешнего инструментария векторизации авторы использовали программу Scout, разработанную в Дрезденском техническом университете, которая была доступна в исходных текстах. Программа Scout реализована на основе компиляторной инфраструктуры Clang/LLVM и снабжена лицензией, позволяющей использовать ее в исходном или измененном видах. Одной из важных целей при разработке программы Scout было обеспечение максимальной гибкости. Такое требование относится как к ее способности векторизовать разнообразные циклические конструкции, так и к возможности настраиваться на различные современные SIMD-архитектуры. В настоящее время в Scout реализована поддержка нескольких вариантов векторных расширений SSE и AVX.

Программа Scout осуществляет векторизацию циклов на уровне исходного кода на языке C. Она трансформирует входную программу в векторизованную также на языке C, руководствуясь при этом заданными программистом директивами. В отличие от большинства современных компиляторов, программа Scout способна векторизовать не только внутренние, но и объемлющие (не внутренние) циклы в гнездах циклов. Программа Scout использует низкоуровневый метод векторизации, основанный на развертке и сжатии (*unroll-and-jam*), являющийся упрощенной версией метода SLP [10], применяемого в современных компиляторах.

В векторизованных циклах скалярные переменные и операции заменяются на векторные. Векторные операции могут быть представлены в виде арифметических операторов или вызовов функций. Функции, реализующие векторные операции, являются встроенными функциями компилятора, которые соответствуют командам процессора, либо inline-функциями, использующими встроенные функции.

Программа Scout учитывает, что число итераций цикла может быть не кратно длине вектора (числу элементов в векторе), и генерирует остаточный цикл, который реализует несколько последних итераций. Если известно, что остаточный цикл не нужен, в директиве

векторизации можно задать параметр, отменяющий его генерацию. Поддерживаются также параметры, задающие гарантированное выравнивание данных для массивов и указателей.

Программа Scout способна векторизовать циклы редукции, такие как суммирование или вычисление максимума элементов вектора. В векторизованной версии цикла вычисляется вектор частичных значений (сумм, максимумов и т. д.), а затем выполняется редукция этого вектора.

В некоторых случаях Scout может векторизовать циклы, в теле которых содержатся ветвления. Если условие ветвления инвариантно относительно цикла, то ветвление выносится из цикла, и формируются два цикла, для then-части и else-части. Если условие не инвариантно, делается попытка выполнить if-конверсию и сгенерировать код с использованием предикатных операций или операций условного копирования. Если цикл содержит невекторизуемые вычисления, Scout выполняет его частичную векторизацию.

Помимо собственно векторизации, Scout поддерживает директивы для развертки циклов в заданное число итераций, а также inline-подстановку функций. Подстановка функций, если это возможно, выполняется также автоматически перед векторизацией циклов.

Программа Scout использовалась в основном в Аэрокосмическом Центре Германии и позволила добиться существенного увеличения производительности на задачах газовой динамики [8]. При этом исходный код приложений изменялся только путем добавления директив Scout. Проект имел также важное исследовательское значение, поскольку некоторые идеи, опробованные в его реализации, были затем применены для развития средств векторизации в компиляторах. Версия 1.6 была выпущена в феврале 2014 г., и с тех пор эта программа, по-видимому, не развивалась. Тем не менее авторы использовали ее как характерный пример внешнего инструментария директивной векторизации, имеющего ряд интересных возможностей.

2. Примеры

В этом разделе продемонстрируем специфику директивной и автоматической векторизаций на нескольких примерах циклов с различными особенностями. Все примеры реализованы на языке C. Использовался компилятор GCC версий 5.3 и 6.1. Директивная векторизация выполнялась с помощью программы Scout, а также с помощью компилятора GCC с применением директив стандарта OpenMP 4.0. Эксперименты проводили с векторными расширениями AVX и SSE4. Большинство примеров с некоторыми изменениями взяты из работ [7, 11].

2.1. Простые циклы с арифметическими вычислениями

Рассмотрим пример функции сложения одномерных массивов (листинг 1).

```
#define SIZE (1L << 16)
void test1(double *a, double *b)
{
 int i;
 for (i = 0; i < SIZE; i++)
 {
       a[i] += b[i];
 }
}
```

Листинг 1. Пример цикла с арифметическими вычислениями

Скомпилируем его с ключами

```
gcc -O3 -c -mavx -fopt-info-vec-optimized
          test1.c –save-temps
```

Ключ `-O3` включает автоматическую векторизацию, ключ `-fopt-info-vec-optimized` обеспечивает вывод сообщений о векторизованных циклах в стандартный протокол. В данном случае получаем сообщение

```
test1.c:5:2: note: loop vectorized
test1.c:5:2: note: loop versioned for
vectorization because of possible aliasing
test1.c:5:2: note: loop peeled for
vectorization to enhance alignment
```

Сообщение констатирует, что (1) цикл был векторизован; (2) помимо векторизованной версии цикла сгенерирована также невекторизованная версия на тот случай, если массивы `a`, `b` перекрываются в памяти, создавая зависимости, препятствующие векторизации (подробнее о зависимостях см. в работе [12]); (3) цикл был частично раскатан (`peeled`) для того чтобы обеспечить выровненный доступ к массиву `a`. В результате для этой простой программы компилятор сгенерировал довольно обширный и непростой для понимания код.

Покажем теперь, как можно обеспечить более благоприятные условия для автоматической векторизации цикла из листинга 1. Если массивы `a`, `b` в функции `test1` не могут перекрываться и для них обеспечено достаточное выравнивание (16 для SSE4 и 32 для AVX), то можно помочь компилятору сгенерировать более эффективный и компактный код, используя спецификатор `restrict` и встроенную функцию `__builtin_assume_aligned`, задающую выравнивание указателя (листинг 2).

```
#define SIZE      (1L << 16)
void test1a(double *restrict a, double *restrict b)
{
      int i;
      double *x = __builtin_assume_aligned(a, 32);
      double *y = __builtin_assume_aligned(b, 32);
      for (i = 0; i < SIZE; i++)
      {
      x[i]+ = y[i];
      }
}
```

Листинг 2. Использование спецификаторов restrict и встроенной функции `__builtin_assume_aligned`

Для этой программы компилятор генерирует векторизованный цикл без версий и пилинга, используя операции выровненного чтения/записи. Заметим, что в последующих примерах вызовы __builtin _ assume _ aligned не используются в целях сокращения и упрощения записи.

Аналогичный результат можно получить с помощью директивной векторизации на основе стандарта OpenMP. Для этого достаточно вставить перед циклом из листинга 1 директиву

```
#pragma omp simd aligned(a,b:4*sizeof(double))
```

и добавить ключ компиляции -fopenmp-simd.

Нужно учитывать, что директива omp simd подразумевает отсутствие зависимостей по данным в следующем за ней цикле. Для учета межитерационных зависимостей, таких как a[i + k] = a[i] + c, предусмотрен параметр safelen(*k*), где *k* — максимальная длина вектора, при которой векторизация не нарушит зависимостей.

Рассмотрим теперь векторизацию этого цикла с помощью Scout. Для применения Scout перед заголовком цикла в листинге 1 нужно вставить директиву #pragma scout loop vectorize. Запустим Scout, указав AVX в качестве векторного расширения, которое следует использовать:

```
scout _ cli -scout:configuration = avx.cpp
-scout:extension = vect test1.c > test
```

Выдача Scout

```
This is Scout
version 1.6.0
Boost version 1.58.0
clang version 3.4 (tags/RELEASE _ 34/final 237279)
warning: load configuration avx.cpp
warning: start processing
test1.c:7:2: note: loop vectorized {tgt:0:0}
```

показывает, что цикл был векторизован (note: loop vectorized). Результат векторизации сохранен в файле test1.vect (листинг 3). Это программа на языке C с расширениями, включающими векторные типы и встроенные функции, соответствующие векторным командам процессора. В данном случае используются переменные векторного типа __ m256d. Переменная i в цикле увеличивается на 4, поскольку в каждой итерации обрабатываются векторы из 4 double-значений. Комментарий перед заголовком цикла (/* 2 packed loads, 1 packed stores, 1 packed ops */) сообщает, сколько векторных команд чтения, записи и вычислительных операций удалось задействовать в теле сгенерированного цикла.

Для того чтобы получить объектный модуль, файл test1.vect необходимо скомпилировать с помощью компилятора, включив заголовочный файл с описаниями используемых Scout векторных типов и функций.

Рассмотрим содержательные отличия кода, сгенерированного Scout, от результата векторизации компилятором GCC. Во-первых, программа Scout после векторизованного цикла сгенерировала остаточный цикл, необходимый для реализации завершающих итераций исходного цикла, если число итераций не кратно числу элементов вектора. Программа Scout не распознает ситуации, когда этот цикл избыточен.

```
#define SIZE      (1L << 16)
void test1(double *a, double *b)
{
  __m256d art_vect0, art_vect1, a_i_vect2;
  int i;
  /* 2 packed loads, 1 packed stores, 1 packed ops */
  for (i = 0; i < SIZE - 3; i += 4)  {
    art_vect0 = _mm256_loadu_pd(a + i);
    art_vect1 = _mm256_loadu_pd(b + i);
    a_i_vect2 = _mm256_add_pd(art_vect0, art_vect1);
    _mm256_storeu_pd(a + i, a_i_vect2);
  }
  for (; i < SIZE; i++)  {
    a[i] = a[i] + b[i];
  }
}
```

Листинг 3. Результат векторизации цикла с помощью Scout

Можно отменить генерацию остаточного цикла, добавив параметр noremainder в директиву векторизации: #pragma scout loop vectorize noremainder.

Во-воторых, программа Scout полагается на отсутствие зависимостей по данным в цикле, подлежащем векторизации. В отличие от компилятора, она не генерирует невекторизованную версию цикла на случай, если массивы a и b перекрываются с дистанцией меньшей, чем размер вектора. Даже зависимость, явно следующая из исходного кода цикла, такая как a[i + 1] = a[i] + c;, не препятствует векторизации. Таким образом, используя директиву #pragma scout loop vectorize, программист должен гарантировать отсутствие подобных зависимостей. В противном случае векторизованный код может оказаться неэквивалентным исходной программе.

В-третьих, еще одно отличие Scout в сравнении с компилятором заключается в генерации кода для невыровненных данных. Если нет информации о выравнивании данных, Scout генерирует операции невыровненного чтения и записи (_ mm256 _ loadu _ pd, _ mm256 _ storeu _ pd). Компилятор же применяет пилинг для того, чтобы обеспечить более эффективный выровненный доступ к массиву a (поскольку в этом цикле a используется чаще, чем b).

Программе Scout также можно сообщить информацию о выравнивании массивов с помощью параметра aligned:

```
#pragma scout loop vectorize noremainder aligned(a,b)
```

В этом случае будет сгенерирован код с использованием операций выровненного чтения и записи (листинг 4).

```
void test1(double *a, double *b)
{
  __m256d art_vect0, art_vect1, a_i_vect2;
  int i;
  /* 2 packed loads, 1 packed stores, 1 packed ops */
  for (i = 0; i < SIZE; i += 4)  {
    art_vect0 = _mm256_load_pd(a + i);
    art_vect1 = _mm256_load_pd(b + i);
    a_i_vect2 = _mm256_add_pd(art_vect0, art_vect1);
    _mm256_store_pd(a + i, a_i_vect2);
  }
}
```

Листинг 4. Результат Scout-векторизации цикла из листинга 1 с параметрами aligned и noremainder

Указывая параметр `aligned(a,b)`, гарантируем, что все обращения по указателям `a, b` выровнены. Если выровнены не все обращения, то нужно явно указать, какие именно адресные выражения выровнены, например: `aligned(a[2*i], b[k + 2])`.

2.2. Циклы с условными вычислениями

Стратегия векторизации для циклов с условными вычислениями зависит от характера условий. Рассмотрим пример цикла на листинге 5. Здесь условие инвариантно относительно цикла. Как компилятор, так и Scout выполняют вынесение условия из цикла и формируют два новых цикла (листинг 6), каждый из которых в дальнейшем векторизуется.

```
void test27_4(int mode, double * restrict a,
double * restrict c, double b)
{
#pragma scout loop vectorize aligned(a,c)
 for (int i = 0; i < 100; i++)
 {
  c[i] = mode == 0 ? a[i] + b : a[i] - b;
 }
}
```

Листинг 5. Цикл с инвариантным условием

```
  if (mode == 0)
    for (i = 0; i < 100; i++)  {
      c[i] = a[i] + b;
    }
  else
    for (i = 0; i < 100; i++)  {
      c[i] = a[i] - b;
    }
```

Листинг 6. Результат вынесения условия из цикла, представленного на листинге 5

Если условие зависит от переменных цикла, векторизация возможна с использованием команд векторного сравнения и условных пересылок или предикатных вычислений, если они поддерживаются процессором. Рассмотрим цикл, представленный на листинге 7.

```
#define SIZE      (1L << 16)
void test8(double* restrict x, double* restrict y)
{
  int i;
  for (i = 0; i < SIZE; i++) {
      if (y[i] > x[i]) x[i] = y[i];
  }
}
```

Листинг 7. Цикл с условным вычислением

При компиляции в режиме автоматической векторизации для AVX получаем сообщение о том, что цикл векторизован. Однако исследование ассемблерного кода показывает, что фактически векторизация не выполнена (это известная ошибка, которая, вероятно, будет исправлена в следующих версиях компилятора GCC). Заметим, что при использовании директивы `#pragma omp simd aligned(x,y:4*sizeof(double))` и ключа `-fopenmp-simd` данный цикл успешно векторизуется.

Применение Scout к циклу из листинга 7 с добавлением директивы `#pragma scout loop vectorize noremainder aligned(x, y)` также дает неожиданные результаты. Хотя на экран выводится сообщение о том, что цикл векторизован, фактически сгенерированный код (листинг 8) не содержит векторных команд.

```
void test8(double *x, double *y)
{
  __m256d art_vect3;
  unsigned int art_vect1, art_vect2;
  int art_vect0[4];
  int i;
  art_vect1 = 0;
  /* 0 packed loads, 0 packed stores, 0 packed ops */
  for (i = 0; i < SIZE; i++)  {
    if (y[i] > x[i]) {
      art_vect0[art_vect1++] = i;
      if (art_vect1 == 4U) {
        art_vect3 = _mm256_set_pd(y[art_vect0[3]], y[art_vect0[2]],
                                  y[art_vect0[1]], y[art_vect0[0]]);
        x[art_vect0[0]] = _mm256_extract_pd(art_vect3, 0);
        x[art_vect0[1]] = _mm256_extract_pd(art_vect3, 1);
        x[art_vect0[2]] = _mm256_extract_pd(art_vect3, 2);
        x[art_vect0[3]] = _mm256_extract_pd(art_vect3, 3);
        art_vect1 = 0U;
      }
    }
  }
  for (art_vect2 = 0U; art_vect2 < art_vect1; ++art_vect2)
    x[art_vect0[art_vect2]] = y[art_vect0[art_vect2]];
}
```

Листинг 8. Результат Scout-векторизации цикла из листинга 7

Из комментария перед телом цикла видно, что ни одной векторной операции в цикле не задействовано. Формально полученная программа эквивалентна исходной. Смысл преобразования, выполненного Scout, в том, что в четырехэлементном целочисленном массиве (не векторе) `art_vect0` накапливаются значения индекса `i`, для которых выполняется условие `y[i] > x[i]`. Как только накапливается четыре таких значения, проводится считывание четырех элементов массива `y` в вектор, а затем запись элементов этого вектора в массив `x`. И чтение, и запись проводятся "не подряд". Но AVX не поддерживает подобные "индексные" операции (*gather/scatter*) в полном объеме, а кроме того, Scout, по-видимому, не умеет их использовать, поэтому и для чтения, и для записи сгенерирован скалярный код. Впрочем, в данном примере даже использование индексных операций чтения и записи векторов не спасло бы ситуацию, так как считывание и сравнение `x[i]` и `y[i]` проводится с помощью скалярных операций.

Аналогичные результаты дает Scout и для рассмотренного далее более сложного цикла из листинга 9, в котором представлен чуть более сложный цикл, использующий три массива. Этот цикл успешно векторизуется компилятором в режиме автоматической векторизации, а также с помощью директивы `#pragma omp simd`.

Заметим, что для SSE4 примеры из листингов 7, 9 не векторизуются ни автоматически, ни

```
#define SIZE      (1L << 16)
void test19(double *restrict x, double *restrict y,
double *restrict z)
{
      int i;
#pragma omp simd aligned(x,y,z:4*sizeof(double))
#pragma scout loop vectorize noremainder aligned(x, y, z)
      for (i = 0; i < SIZE; i++) {
            if (y[i] <= z[i]) x[i] = z[i];
      }
}
```

Листинг 9. Более сложный цикл с условным вычислением

с помощью директивы `#pragma openmp simd`. Это связано с отсутствием в SSE4 инструкции условной записи в память. Без нее векторизация невозможна, так как стандарт языка запрещает генерацию записей, не предусмотренных исходным кодом программы, в память.

Примеры этого раздела показывают, что векторизация циклов с условными вычислениями связана со значительными сложностями и не всегда осуществляется на практике, даже если она в принципе возможна.

2.3. Циклы с вызовами функций

Как компилятор, так и программа Scout выполняют inline-подстановки функций перед векторизацией циклов. Таким образом, цикл, содержащий вызов функции, может быть векторизован, если определение функции находится в пределах той же единицы компиляции и содержащиеся в ней вычисления векторизуемы.

Если определение функции недоступно в файле, где находится использующий ее цикл, то можно применить средства оптимизации на стадии компоновки (*Link-Time Optimization* — LTO), поддерживаемые многими современными компиляторами, включая GCC. В таком случае подстановка функций и векторизация циклов могут проводиться во время оптимизаций на стадии компоновки.

Стандарт OpenMP предусматривает описания и использование так называемых элементных (*elemental*) функций. Для элементных функций компилятор генерирует как скалярный, так и векторный варианты, что позволяет векторизовать циклы, содержащие вызовы таких функций (см. примеры в работе [13]).

Особый случай представляют собой вызовы стандартных математических функций языка C. Компилятор, имея информацию о свойствах и семантике стандартных функций, может применять различные способы оптимизации их вызовов и векторизации циклов, содержащих такие вызовы. Например, компилятор icc использует вызовы векторных версий математических функций, предоставляемые библиотекой. Для константного выражения, включающего математическую функцию, может быть выполнена свертка. Наконец, если целевой процессор имеет векторную команду для вычисления математической функции (например, `sqrt`, `abs`), то компилятор может в определенных условиях использовать ее при векторизации цикла. Рассмотрим пример из листинга 10.

```
#include <math.h>
void test26_3(double x[100], double y[100], double t)
{
      int i;
      for (i = 0; i < 100; i++) {
        y[i] = x[i] >= 0.0 ? sqrt(x[i]) : t;
      }
}
```

Листинг 10. Пример цикла с вызовом функции `sqrt`

Хотя AVX предоставляет векторную команду вычисления квадратного корня, автоматическая векторизация со стандартным набором ключей, показанным в подразд. 2.1, не выполнилась. Причина в том, что функция `sqrt()` должна устанавливать значение переменной `errno` в случае некорректного аргумента. Поэтому результат команды извлечения корня проверяется, и, если это NaN ("не число"), то для установки `errno` осуществляется вызов библиотечной функции `sqrt()`, что делает векторизацию невозможной. При добавлении ключа `-fno-math-errno` векторизация этого цикла выполняется успешно с использованием векторных команд чтения и записи, сравнения, извлечения квадратного корня и условного копирования.

Рассмотрим теперь векторизацию цикла из листинга 10 с помощью Scout. Для этого добавим перед заголовком цикла директиву `#pragma scout loop vectorize noremainder aligned(x, y)`. В командной строке нужно указать ключом `-I` маршрут, по которому доступен заголовочный файл `math.h` (либо описать функцию `sqrt` непосредственно в C-файле).

Результат применения Scout дан в листинге 11. Видно, что цикл векторизован максимально эффективно, все операции в нем векторные. Следует учитывать, что Scout векторизует подобные циклы в обход требования стандарта об установке переменной `errno`. В данном случае это допустимо, поскольку корректность входных аргументов `sqrt` в программе гарантирована.

```
#include <math.h>
void test26_3(double x[100], double y[100], double t)
{
  __m256d art_vect1;
  __m256d art_vect0, art_vect2, art_vect3, y_i_vect4;
  int i;
  art_vect1 = _mm256_set1_pd(0.);
  art_vect3 = _mm256_broadcast_sd(&t);
  /* 1 packed loads, 1 packed stores, 2 packed ops */
  for (i = 0; i < 100; i += 4)  {
    art_vect0 = _mm256_load_pd(x + i);
    art_vect2 = _mm256_sqrt_pd(art_vect0);
    y_i_vect4 = _mm256_blendv_pd(art_vect3, art_vect2,
      _mm256_cmp_pd(art_vect0, art_vect1, _CMP_GE_OQ));
    _mm256_store_pd(y + i, y_i_vect4);
  }
}
```

Листинг 11. Результат Scout-векторизации цикла из листинга 10

2.4. Циклы редукции

К циклам редукции относят циклы, вычисляющие значения, которые зависят от всех элементов входного массива (или массивов), такие как сумма, максимум или минимум элементов вектора, ска-

```
#define SIZE      (1L << 16)
double test25(double *x, double *y) {
  int i;
  double r=0;
  for (i = 0; i < SIZE; i++) {
    r += x[i] * y[i];
  }
  return r;
}
```

Листинг 12. Цикл вычисления скалярного произведения

лярное произведение векторов и т. п. Рассмотрим в качестве примера цикл вычисления скалярного произведения (листинг 12).

Данный цикл может быть векторизован с помощью трансформации исходного цикла в цикл для вычисления вектора частичных сумм. Вслед за таким циклом добавляется код для суммирования элементов вектора частичных сумм. Такой подход требует изменения порядка операций, заданных во входной программе, что противоречит стандарту языка C. Поэтому автоматическая векторизация подобных циклов компилятором возможна только с ключом `-funsafe-math-optimizations`. Использование этого ключа, отменяющего некоторые из требований стандарта вещественной арифметики IEEE754, может быть нежелательно для всей программы. Поэтому для подобных циклов предпочтительно применение директивной векторизации Scout или OpenMP. В директиве OpenMP при этом необходимо явно указать параметр редукции (reduction): `#pragma omp simd aligned(x,y:32) reduction(+:r)`.

Программа Scout генерирует для этого цикла код, представленный на листинге 13. Основной цикл векторизован эффективно, но последующий код для сложения частичных сумм реализован не лучшим образом: в нем не использованы горизонтальные команды для сложения элементов вектора. Впрочем, при обработке больших массивов этот недостаток не может существенно ухудшить общее время выполнения.

```
double test25(double *x, double *y)
{
  __m256d r_vect0, art_vect1, art_vect2, art_vect3;
  unsigned int art_vect4;
  int i;
  double r;
  r = 0;
  r_vect0 = _mm256_set1_pd(0);
  /* 2 packed loads, 0 packed stores, 2 packed ops */
  for (i = 0; i < SIZE; i += 4)  {
    art_vect1 = _mm256_load_pd(x + i);
    art_vect2 = _mm256_load_pd(y + i);
    art_vect3 = _mm256_mul_pd(art_vect1, art_vect2);
    r_vect0 = _mm256_add_pd(r_vect0, art_vect3);
  }
  for (art_vect4 = 0U; art_vect4 < 4U; ++art_vect4)  {
    r = r + _mm256_extract_pd(r_vect0, art_vect4);
  }
  return r;
}
```

Листинг 13. Результат Scout-векторизации цикла вычисления скалярного произведения векторов

2.5. Циклы с непоследовательным доступом к элементам массивов

Особого подхода требует векторизация циклов, в которых доступ к элементам массивов осуществляется не строго последовательно. Компилятор GCC распознает некоторые шаблоны непоследовательного доступа к данным и генерирует векторный код с применением команд, реализующих перестановку элементов вектора. Рассмотрим в качестве примера цикл, представленный на листинге 14.

```
#define SIZE      (1L << 15)
void test_prm1 (double *restrict a, double *restrict b){
      int i;
      double *x = __builtin_assume_aligned(a, 32);
      double *y = __builtin_assume_aligned(b, 32);
      for (i = 0; i < SIZE; i+=2) {
            x[i] = y[i+1];
            x[i+1] = y[i];
      }
}
```

Листинг 14. Цикл с непоследовательным доступом к элементам массива

Для SSE4 компилятор генерирует эффективный векторный код, используя команду `palignr` для перестановки элементов вектора с элементами (`y[i], y[i + 1]`). Векторизация выполняется как в автоматическом режиме, так и с применением директивы `#pragma omp simd aligned(x,y:4*sizeof(double))`. Для AVX, где число элементов вектора равно четырем, векторный код с перестановками оказывается менее эффективным.

Результат применения Scout к этой программе для SSE4 с использованием директивы `#pragma scout loop vectorize noremainder aligned(x[i], y[i])` показан на листинге 15. Хотя в выдаче на терминал сообщается, что цикл векторизован, фактически сгенерирован некоторый вариант развертки цикла, в котором нет ни одной векторной операции. Она может оказаться несколько эффективней скалярного варианта за счет снижения накладных расходов на организацию цикла. Но более вероятен проигрыш в производительности. Это зависит от того, насколько эффективно реализованы функции `_mm_set_pd`, `_mm_extract_pd`. Данный пример показывает, что в Scout, по-видимому, не реализованы механизмы применения перестановок в векторизуемых циклах.

```
#define SIZE    (1L << 15)
void test_prm1 (double *restrict x, double *restrict y){
  __m128d x_i_vect0, x_i_1_vect1;
  int i;
  /* 0 packed loads, 0 packed stores, 0 packed ops */
  for (i = 0; i < SIZE; i += 2 * 2)  {
    x_i_vect0 = _mm_set_pd(y[(i + 1 + 2)], y[i + 1]);
    x[i] = _mm_extract_pd(x_i_vect0, 0);
    x[(i + 2)] = _mm_extract_pd(x_i_vect0, 1);
    x_i_1_vect1 = _mm_set_pd(y[(i + 2)], y[i]);
    x[i + 1] = _mm_extract_pd(x_i_1_vect1, 0);
    x[(i + 1 + 2)] = _mm_extract_pd(x_i_1_vect1, 1);
  }
]
```

Листинг 15. Результат Scout-векторизации цикла из листинга 14 для SSE4

2.6. Вложенные циклы

Применение средств векторизации к гнездам циклов может представлять дополнительные сложности. Иногда для получения эффективного кода выгоднее векторизовать не внутренний цикл в гнезде циклов, а один из объемлющих. Рассмотрим пример, представленный на листинге 16.

```
void test28_5(double a[100], double b[1000],
              double c, int k, double d[100])
{
      int i, j;
      for (i = 0; i < 100; i++)
      {
        d[i] = a[i];
        for (j = 0; j < k; j++) {
            d[i] += b[j*k + i];
        }
        d[i] *=c;
      }
}
```

Листинг 16. Пример гнезда циклов

Рассмотрим результаты, которые удалось получить с помощью Scout. Попробуем сначала применить Scout для векторизации внутреннего цикла, используя директиву `#pragma scout loop vectorize noremainder aligned(d)`. Результат показан на листинге 17.

Очевидно, что результат никуда не годится: единственная векторизованная операция — сложение. Ни чтение из памяти, ни запись не векторизованы, и, кроме того, так как внутренний цикл представляет собой редукцию, то добавлен еще цикл суммирования элементов вектора, а значит, при малых значениях `k` эффективность может быть ниже, чем у скалярной версии. Параметр `aligned(d)` здесь бесполезен, так как во внутреннем цикле массив `d` представлен только скалярным значением `d[i]`. Попробуем теперь векторизовать внешний цикл. Результат показан на листинге 18.

```
void test28_5(double a[100], double b[1000],
              double c, int k, double d[100])
{
  __m256d d_i_vect0, art_vect1;
  unsigned int art_vect2;
  int i, j;
  for (i = 0; i < 100; i++)  {
    d[i] = a[i];
    d_i_vect0 = _mm256_set1_pd(0);
    /* 0 packed loads, 0 packed stores, 1 packed ops */
    for (j = 0; j < k; j += 4)  {
      art_vect1 = _mm256_set_pd(b[(j * k + i + (k * 3))], b[(j * k + i + (k *
2))], b[(j * k + i + k)], b[j * k + i]);
      d_i_vect0 = _mm256_add_pd(d_i_vect0, art_vect1);
    }
    for (art_vect2 = 0U; art_vect2 < 4U; ++art_vect2)  {
      d[i] = d[i] + _mm256_extract_pd(d_i_vect0, art_vect2);
    }
    d[i] *= c;
  }
}
```

Листинг 17. Результат Scout-векторизации внутреннего цикла из листинга 16

```
void test28_5(double a[100], double b[1000],
              double c, int k, double d[100])
{
  __m256d d_i_vect0, art_vect1, art_vect2;
  int i, j;
  art_vect2 = _mm256_broadcast_sd(&c);
  /* 2 packed loads, 1 packed stores, 2 packed ops */
  for (i = 0; i < 100; i += 4)  {
    d_i_vect0 = _mm256_load_pd(a + i);
    for (j = 0; j < k; j++)  {
      art_vect1 = _mm256_loadu_pd(b + (j * k + i));
      d_i_vect0 = _mm256_add_pd(d_i_vect0, art_vect1);
    }
    d_i_vect0 = _mm256_mul_pd(d_i_vect0, art_vect2);
    _mm256_store_pd(d + i, d_i_vect0);
  }
```

Листинг 18. Результат Scout-векторизации внешнего цикла из листинга 16

Теперь во внутреннем цикле векторизованы все операции. Так как внешний цикл не является циклом редукции, нет необходимости в цикле суммирования элементов вектора. Единственное, что может снизить эффективность, это невыровненное чтение элементов `b` во внутреннем цикле. Такие обращения к памяти действительно невыровнены, поэтому в данном случае все зависит от того, насколько эффективно аппаратура выполняет невыровненное чтение. Тем не менее, даже в худшем случае, когда `_mm256_loadu_pd` выполняется как четыре скалярных считывания из памяти, можно рассчитывать, что программа будет работать не медленнее, чем исходная невекторизованная. Для чтения массива `a` и записи в массив `d` сгенерированы выровненные операции, так как указан параметр `aligned(d, a)`.

Что касается компилятора, то в данном примере он не выполняет векторизацию ни для внутреннего, ни для внешнего циклов, ни в автоматическом режиме, ни с применением директив `openmp simd`. В некоторых руководствах по оптимизации можно встретить рекомендацию оформлять тело внешнего цикла как элементную функцию. Однако авторам не удалось таким способом добиться эффективной векторизации внешних циклов. Разумеется, остается возможность инвертировать гнездо циклов вручную.

Заключение

Представленные в разд. 2 примеры показывают, что ни одно из рассмотренных средств не обеспечивает эффективной векторизации для всех разновидностей циклов. Возможный подход состоит в совместном использовании различных инструментальных средств векторизации.

В первую очередь, необходимо выявить набор программных циклов (и гнезд циклов), критичных с точки зрения производительности, например, с помощью

профилирования. Циклы, успешно векторизованные компилятором, можно попытаться дополнительно "окультурить", как описано в подразд. 2.1 (листинг 2), либо применив к ним директиву `#pragma omp simd aligned(...)`. Для внутренних циклов, которые не удалось векторизовать автоматически, можно попытаться применить директивы OpenMP. При этом необходимо учитывать специфику директивной векторизации, касающуюся трактовки зависимостей по данным, а также возможные нарушения требований языковых стандартов. Если целесообразна векторизация одного из объемлющих циклов в гнезде вложенных циклов, полезно может быть инструментальное средство, подобное Scout, поскольку в современных компиляторах эта функциональная возможность не поддерживается.

Для циклов, которые не удалось эффективно векторизовать ни одним из инструментальных средств, следует проанализировать причины неудачи. В этом может помочь изучение дампов компилятора, хотя это требует определенной квалификации. Неуспех векторизации может быть связан с ограничениями либо ошибками в реализации инструментального средства. Сообщая разработчикам о подобных ситуациях, вы можете способствовать тому, что в очередную версию будут внесены соответствующие исправления. В некоторых случаях может помочь трансформация программного кода, снимающая препятствия к векторизации. Полезные рекомендации по написанию хорошо векторизуемых циклов можно найти, например, в работе [12].

Отдельного рассмотрения заслуживает вопрос о том, как определить, был ли векторизирован цикл и насколько эффективно это было сделано. Приведенные выше примеры показывают, что наличие диагностики *"loop vectorized"* не дает гарантии эффективной векторизации. Необходимо дополнительно исследовать результат с помощью профилирования либо путем изучения сгенерированного кода. С точки зрения удобства анализа векторизированной программы несомненным преимуществом обладают средства векторизации на уровне исходного кода. Вместе с тем при их использовании нужно учитывать, что производительность может зависеть от того, насколько эффективно реализованы встроенные функции, используемые в векторизованных циклах.

Анализ ассемблерного кода, сгенерированного компилятором, может быть затруднителен, так как не всегда легко выявить фрагменты, относящиеся к конкретному циклу. Для упрощения этой задачи полезно вынесение цикла в отдельную функцию, а также использование ключей компилятора, помогающих сопоставить ассемблерные команды с участками исходной программы.

Таким образом, векторизация унаследованных приложений остается на данный момент сложной и трудоемкой задачей. Не существует инструментального средства (или набора таких средств) векторизации, который можно было бы использовать просто как элемент технологического конвейера в процессе разработки и сборки приложений. Имеющиеся средства требуют квалифицированного подхода с индивидуальным рассмотрением каждого "горячего" цикла и ручным анализом результатов векторизации.

Что касается создания нового программного обеспечения, то ставка в настоящее время делается на развитие, стандартизацию и широкое применение средств явного векторного программирования. Однако эти средства доступны пока не во всех компиляторах и не для всех векторных расширений, а их реализации небезупречны. Поэтому актуальным остается также использование подходящих библиотек, рекомендаций по написанию хорошо векторизуемого кода, а также ручного векторного программирования.

Список литературы

1. **Вьюкова Н. И., Галатенко В. А., Самборский С. В.** Использование векторных расширений современных процессоров // Программная инженерия. 2016. № 4. С. 147—157.

2. **Maleki S., Gao Y., Garzaran M. J., Wong T., Padua D. A.** An Evaluation of Vectorizing Compilers. URL: http://polaris.cs.uiuc.edu/~garzaran/doc/pact11.pdf.

3. **Satish N., Kim C., Chhugani J., Saito H., Krishnaiyer R., Smelyanskiy M., Girkar M., Dubey P.** Can Traditional Programming Bridge the Ninja Performance Gap for Parallel Computing Applications? URL: http://web.eecs.umich.edu/~msmelyan/papers/isca-2012-paper.pdf

4. **Explicit** Vector Programming. URL: https://software.intel.com/en-us/code-samples/intel-c-compiler/explicit-vector-programming

5. **Intel** Cilk^{TM} Plus. URL: https://www.cilkplus.org

6. **OpenMP** Application Program Interface. Version 4.5. November 2015. URL: http://www.openmp.org/mp-documents/openmp-4.5.pdf

7. **An Introduction** to Scout, a Vectorizing Source-to-Source Transformator. URL: http://tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zih /forschung/projekte/scout/Scout_ACCU.pdf

8. **Krzikalla O., Feldhoff K., Müller-Pfefferkorn R., Nagel W. E.** Auto-Vectorization Techniques for Modern SIMD Architectures. 16th Workshop on Compilers for Parallel Computing, January 11—13, 2012, Padova, Italy. URL: https://tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zih/forschung/projekte/scout/Scout_CPC2012.pdf

9. **Pokam G., Bihan S., Simmonet J., Bodin F.** SWARP — A Retargetable Preprocessor for Multimedia Instructions // Concurrency and Computation: Practice and Experience. 2004. Vol. 16, N. 2—3. P. 303—318.

10. **Larsen S., Amarasinghe S.** Exploiting Superword Parallelism with Multimedia Instruction Sets. // Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation (PLDI 2000). Vancouver, British Columbia, Canada. 2000. P. 145—156.

11. **Auto-vectorization** with gcc 4.7. Lockless, Inc. URL: http://locklessinc.com/articles/vectorize

12. **A Guide to Vectorization** with Intel® C++ Compilers. URL: https://software.intel.com/sites/default/files/m/4/8/8/2/a/31848-CompilerAutovectorizationGuide.pdf

13. **Xinmin Tian, Bronis R. de Supinski.** Explicit Vector Programming with OpenMP 4.0 SIMD Extensions. URL: http://primeurmagazine.com/repository/PrimeurMagazine-AE-PR-12-14-32.pdf

Directive and Automatic Loop Vectorization

N. I. Vyukova, niva@niisi.ras.ru, **V. A. Galatenko,** galat@niisi.ras.ru,
S. V. Samborskij, sambor@niisi.ras.ru, Federal State Institution "Scientific Research Institute of System Analysis of the Russian Academy of Science", Moscow, 117218, Russian Federation

Corresponding author:

Galatenko Vladimir A., Head of Sector, Federal State Institution "Scientific Research Institute of System Analysis of the Russian Academy of Science", Moscow, 117218, Russian Federation,
e-mail: galat@niisi.ras.ru

Received on July 26, 2016
Accepted on July 29, 2016

Nearly all today's computer platforms include Single Instruction Multiple Data (SIMD) extensions. The SIMD model allows for exploiting the data level parallelism present in computationally intensive applications. The auto-vectorization techniques became common in modern compilers. However the compiler's ability for auto-vectorization is limited with certain requirements of language standards, lack of information about data dependencies and alignment and other factors. A programmer can assist the compiler by providing explicit vectorization directives with parameters specifying the properties of data and operations essential for the vectorization process. This opens the opportunities for effective vectorization of a much wider spectrum of loops.

The purpose of this paper is to evaluate and compare some available implementations of automatic and directive vectorization. The capabilities of automatic vectorization are presented with recent versions of GCC. The facilities of directive vectorization are exemplified with the features of the OpenMP 4.0 standard implemented in GCC and the source-to source vectorizing preprocessor named Scout. We analyze the results of applying the above tools to a number of program loops with various specific features (presence of conditional computations, function calls, non-contiguous memory access, nested loops) and evaluate the advantages and limitations of each tool. In the conclusion an approach to their conjoint usage is proposed.

Keywords: *vector extensions, SIMD (Single Instruction Multiple Data), vectorization, explicit vector programming, Scout, OpenMP*

For citation:

Vyukova N. I., Galatenko V. A., Samborskij S. V. Directive and Automatic Loop Vectorization, *Programmnaya Ingeneria***,** 2016, vol. 7, no. 10, pp. 435—445.

DOI: 10.17587/prin.7.435-445

References

1. **Vyukova N. I., Galatenko V. A., Samborskij S. V.** Ispol'zovanie vektornyh rasshirenij sovremennyh processorov (Exploiting Vector Extensions of Modern Processors), *Programmnaya Ingeneria*, 2016, vol. 7, no. 4, pp. 147 — 157 (in Russian).

2. **Maleki S., Gao Y., Garzaran M. J., Wong T., Padua D. A.** An Evaluation of Vectorizing Compilers, available at: http://polaris.cs.uiuc.edu/~garzaran/doc/pact11.pdf

3. **Satish N., Kim C., Chhugani J., Saito H., Krishnaiyer R., Smelyanskiy M., Girkar M., Dubey P.** Can Traditional Programming Bridge the Ninja Performance Gap for Parallel Computing Applications? available at: http://web.eecs.umich.edu/~msmelyan/papers/isca-2012-paper.pdf

4. **Explicit** Vector Programming, available at: https://software.intel.com/en-us/code-samples/intel-c-compiler/explicit-vector-programming

5. **Intel** Cilk^{TM} Plus, available at: https://www.cilkplus.org/

6. **OpenMP** Application Program Interface. Version 4.5. November 2015, available at: http://www.openmp.org/mp-documents/openmp-4.5.pdf

7. **An Introduction** to Scout, a Vectorizing Source-to-Source Transformator, available at: http://tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zih/forschung/projekte/scout/Scout_ACCU.pdf

8. **Krzikalla O., Feldhoff K., Muller-Pfefferkorn R., Nagel W. E.** Auto-Vectorization Techniques for Modern SIMD Architectures, *16th Workshop on Compilers for Parallel Computing*, January 11—13, 2012, Padova, Italy, available at: https://tu-dresden.de/die_tu_dresden/zentrale_einrichtungen/zih/forschung/projekte/scout/Scout_CPC2012.pdf

9. **Pokam G., Bihan S., Simmonet J., Bodin F.** SWARP — A Retargetable Preprocessor for Multimedia Instructions, *Concurrency and Computation: Practice and Experience,* 2004, vol. 16, no. 2—3, pp. 303—318.

10. **Larsen S., Amarasinghe S.** Exploiting Superword Parallelism with Multimedia Instruction Sets, *Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation* (PLDI 2000), Vancouver, British Columbia, Canada, 2000, pp. 145—156.

11. **Auto-vectorization** with gcc 4.7. Lockless, Inc., available at: http://locklessinc.com/articles/vectorize/

12. **A Guide** to Vectorization with Intel® C++ Compilers, available at: https://software.intel.com/sites/default/files/8c/a9/CompilerAutovectorizationGuide.pdf

13. **Xinmin Tian, Bronis R. de Supinski**. Explicit Vector Programming with OpenMP 4.0 SIMD Extensions, available at: http://primeurmagazine.com/repository/PrimeurMagazine-AE-PR-12-14-32.pdf

УДК 004.021 DOI: 10.17587/prin.7.446-455

М. С. Голосовский, руководитель проектов, e-mail: golosovskiy@yandex.ru,
Московский Университет имени С. Ю. Витте, г. Москва

Модель расчета оценок трудоемкости и срока разработки информационных систем на начальном этапе жизненного цикла проекта

Предложена модель расчета оценок трудоемкости и срока разработки информационных систем на начальных стадиях жизненного цикла проекта с использованием в качестве исходных данных описаний требований и рисков, представленных в уставе проекта. Исходные данные модели приводятся к нечетким значениям, которые впоследствии используют в вычислениях, выполняемых на основе схемы нечеткого логического вывода Сугено, модифицированной для возможности использования нечетких входных и выходных переменных.

Ключевые слова: *система Сугено, система нечеткого логического вывода, лингвистическая переменная, фаззификация, COCOMO II, устав проекта, экспертные оценки, оценка трудоемкости, оценка длительности, модель расчета трудоемкости, COSYSMO*

Введение

Вопросу априорного оценивания трудоемкости и сроков разработки информационных систем посвящено большое число публикаций [1—19]. Среди наиболее известных алгоритмических методов оценивания выделяют методы COCOMO II (Constructive Cost Model) [1—4], SLIM-модель (модель Путнэма, Putnam model) [1], COSYMO (Constructive Systems Engineering Cost Model) [5]. Основой для расчета в этих моделях служит экспертная оценка о будущем размере информационной системы, выраженная в числе строк кода или функциональных точек (которые в итоге тоже приводятся к числу строк кода). При этом основная идея, на которой базируются эти модели, заключается в нелинейности возрастания трудоемкости разработки системы в зависимости от ее размера [6—8].

К основным недостаткам оценок трудоемкости и сроков разработки информационных систем, которые получаются с помощью этих моделей, можно отнести следующие.

- В результате работы этих моделей получается точечная оценка, которая может сильно изменяться при незначительном изменении входных характеристик.
- Получение оценки размера разрабатываемой системы, выраженной в строках кода или функциональных точках, осуществляется с привлечением экспертов и может содержать большую ошибку на ранних этапах жизненного цикла проекта.
- Использование в моделях большого числа качественных оценок как процесса разработки информационных систем, так и размеров проекта. Как пример, при оценке пользователи склонны завышать квалификацию персонала или эффективность применяемых методов разработки программного обеспечения (ПО).

Гибкие процессы разработки информационных систем призваны разрешить неопределенность в оценке сроков и трудоемкости разработки информационных систем посредством разбиения проекта на мелкие этапы (инкременты/итерации), при этом оценка сроков и трудоемкости разработки информационных систем проводится командой проекта только для отдельно взятого этапа перед его выполнением. В работе [15] показано, что проекты, использующие agile-методологии, также подвержены недооценке трудоемкости и сроков. Еще одним недостатком agile-проектов является отсутствие оценки числа итераций, необходимых для завершения проекта, так как согласно agile-методологии в конце каждой итерации получается продукт, готовый к использованию. Однако время реализации проекта теоретически может быть бесконечным.

В работе С. Макконела [2] выделены перечисленные далее три критерия качественной оценки.

1. Использование расчетных данных в приоритете перед экспертными.

2. Ретроспективные данные конкретной компании приоритетнее отраслевых данных.

3. Оценка не должна быть точечной, а должна задаваться интервалом — значением верхней и нижней границ. При этом оценка считается успешной, если фактические трудозатраты и срок проекта попадают внутрь интервала.

Целью исследования, результаты которого представлены в статье, является синтез математической

модели, позволяющей рассчитать интервальные оценки сроков и трудоемкости разработки информационных систем без использования в качестве исходных данных числа строк кода, а также уменьшение влияния качественных оценок на результаты расчета путем использования аппарата теории нечетких множеств.

Модель расчета оценки сроков и трудоемкости разработки информационных систем

В качестве исходных данных для расчета прогнозируемых сроков и трудоемкости разработки информационных систем с использованием целевой модели будем использовать следующие данные, которые можно получить из устава проекта, разработанного по методологии Института управления проектами PMI [16]:

— описание требований *Rq* с указанием размера в виде лингвистической переменной и планируемого типа системы, в рамках которой планируется реализовывать требование;

— описание рисков проекта *Ri* с указанием лингвистического значения вероятности его наступления и степени влияния, а также типа риска (влияющего на трудоемкость реализации отдельного требования, на трудоемкость реализации всей информационной системы, на срок выполнения проекта);

— активы процессов организации (в части ретроспективных данных по предыдущим проектам, возможной численности команды, описание процессов разработки и др.).

В работе [17] приведены результаты исследования, показывающего, что при экспертной оценке трудоемкости, выраженной в человеко-днях, результаты оказываются точнее, чем при оценке, выраженной в человеко-часах. Поэтому в предлагаемой модели показатели трудоемкости будут выражены в человеко-днях, а показатели длительности — в днях.

Для выполнения вычислений воспользуемся лингвистическими переменными, представленными асимметричными гауссовыми функциями принадлежности, которые рассчитывают по следующей формуле:

$$\begin{cases} \mu(x) = \mathbf{e}^{-\frac{(x-b)^2}{2a_1^2}}, & x < b \\ \mu(x) = \mathbf{e}^{-\frac{(x-b)^2}{2a_2^2}}, & x \geqslant b \end{cases}, \qquad (1)$$

где a_1 — ширина левой части функции принадлежности; b — ядро функции принадлежности (абсцисса, в которой $\mu(x) = 1$); a_2 — ширина правой части функции принадлежности.

В общем виде функцию принадлежности можно задать тремя параметрами, используемыми в формуле (1) (в литературе часто встречается название "L-R-представление нечеткого числа"):

$$A = [a_1, b, a_2].$$

Использование несимметричных функций принадлежности связано с тем обстоятельством, что эксперты чаще дают оптимистичные оценки трудоемкости и сроков разработки информационных систем, чем пессимистичные [1, 2]. Этот момент будем учитывать при определении арифметических операций с нечеткими переменными.

Над нечеткими числами возможно выполнение стандартных арифметических операций: сложение, вычитания, умножения, деления. Более подробно описание арифметических операций приведено в работе [22]. В статье будем использовать упрощенные формулы сложения, разности, умножения, деления для L-R-представления нечетких чисел [21, 22]:

$$(A_1 + A_2) = \left(a_{1A_1} + a_{1A_2},\ b_{A_1} + b_{A_2},\ a_{2A_1} + a_{2A_2}\right), \qquad (2)$$

$$(A_1 - A_2) = \left(a_{1A_1} - a_{1A_2},\ b_{A_1} - b_{A_2},\ a_{2A_1} - a_{2A_2}\right), \qquad (3)$$

$$(A_1 \cdot A_2) = \left(a_{1A_1} \cdot a_{1A_2},\ b_{A_1} \cdot b_{A_2},\ a_{2A_1} \cdot a_{2A_2}\right), \qquad (4)$$

$$(A_1 / A_2) = \left(a_{1A_1} / a_{1A_2},\ b_{A_1} / b_{A_2},\ a_{2A_1} / a_{2A_2}\right), \qquad (5)$$

где A_1, A_2 — нечеткие числа, над которыми проводятся операции; a_{1A_1}, b_{A_1}, a_{2A_1} — параметры L-R-представления для нечеткого числа A_1; a_{1A_2}, b_{A_2}, a_{2A_2} — параметры L-R-представления для нечеткого числа A_2.

На рис. 1 представлена модель расчета оценки трудоемкости разработки системы на стадии пред-

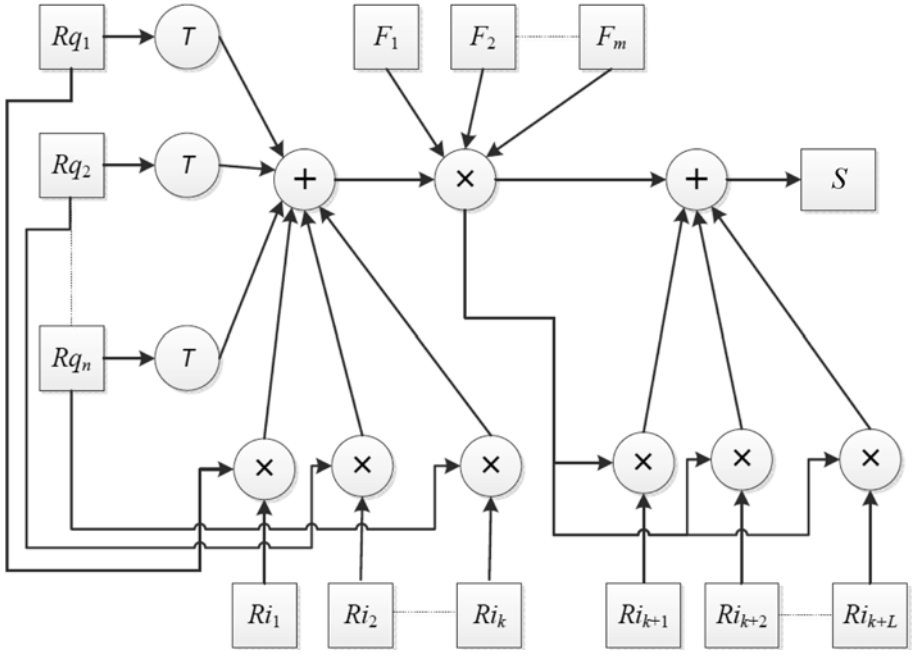

Рис. 1. Модель расчета оценки трудоемкости разработки информационной системы

проектных исследований. Элементами модели являются: Rq_i —требования к системе, представленные в проектной документации; T — модификатор типа системы; Ri_i — проектные риски; F_i — факторы, влияющие на разработку информационной системы; S — оценка трудоемкости разработки информационной системы; k — число рисков, влияющих на реализацию отдельных требований; L — число рисков, влияющих на выполнение проекта в целом; × — оператор умножения, + — оператор сложения. Расчет по этой модели состоит из перечисленных далее этапов.

1. Каждому требованию Rq_i присваивают значение трудоемкости в терминах лингвистических переменных. Эксперт не указывает точное значение трудоемкости в человеко-днях, а выбирает из множества значений одно из значений лингвистической переменной "Трудоемкость требования" {очень маленькая, маленькая, средняя, большая, очень большая}, функции принадлежности которой представлены на рис. 2 (см. третью сторону обложки). По аналогии с техникой покер-планирования [18—20], трудоемкость для каждого ядра функции принадлежности лингвистической переменной возрастает не линейно, а в последовательности, когда каждое последующее значение равно удвоенному значению предыдущего. Это же правило соблюдается для ширины функций принадлежности. В общем случае оценки трудоемкости выполнения требований должны удовлетворять ограничениям конуса неопределенности на этапе инициации проекта, а также, в частности, содержать интервал [0,25x, 4x]. Обоснование такого распределения оценок дано в работах [1, 2].

2. После получения оценки для каждого требования Rq_i его умножают на коэффициент типа системы T, в результате чего получают типизированную оценку требования. Коэффициент типа системы может быть рассчитан на основе ретроспективных или отраслевых данных. Его суть заключается в учете различий между одинаковым лингвистическим значением трудоемкости и ее L-R-представлением. Так, для информационных систем разного типа, например, мобильных приложений, веб-приложений или системных встраиваемых приложений, одинаковое лингвистическое значение трудоемкости будет иметь различное L-R-представление.

3. Если для каждого требования существует риск увеличения трудоемкости реализации требования, связанный непосредственно с выбранным требованием, то рассчитывают влияние риска $\tilde{R}q^r$ по формуле

$$\tilde{R}q^r = \tilde{R}qPrTr, \tag{6}$$

где $\tilde{R}q$ — нечеткое значение трудоемкости требования; Pr — нечеткое значение вероятности риска; Tr — нечеткое значение величины влияния риска. Для эксперта на раннем этапе проекта без количественного и качественного анализа рисков, как правило, трудно установить значение вероятности возникновения риска и степень влияния риска на изменение трудоемкости. В связи с этим обстоятельством по аналогии с представлением переменной трудоемкости эти значения задают в виде лингвистических переменных, с учетом того факта, что функции принадлежности термов лингвистических переменных являются симметричными и располагаются на области определения равномерно.

4. Базовая трудоемкость $\tilde{S}_b$ разработки информационной системы без учета факторов и рисков, влияющих на проект в целом, рассчитывают как сумму типизированных значений трудоемкостей для каждого требования и оценки влияния рисков для каждого требования по формуле

$$\tilde{S}_b = \sum_{i=1}^{N} \tilde{R}q_i^T + \sum_{i=1}^{N} \tilde{R}q_i^r, \tag{7}$$

где $\tilde{R}q_i^T$ — типизированная оценка трудоемкости i-го требования; $\tilde{R}q^r$ — значение влияния риска на i-е требование; N — число требований.

5. Оценку трудоемкости разработки информационной системы с учетом влияния факторов $\tilde{S}_F$ формируют по аналогии с аналогичной оценкой, получаемой методом COCOMO II. При этом можно использовать те же факторы (если они применимы и известны на ранней стадии оценки проекта). Отличие заключается в том, что в качестве значения влияния фактора применяют нечеткое число. Расчет этой оценки осуществляют по формуле

$$\tilde{S}_F = \tilde{S}_b \prod_{i=1}^{M} F_i, \tag{8}$$

где F_i — значение i-го фактора; $\tilde{S}_b$ — значение базовой оценки трудоемкости, рассчитанной по формуле (7); M — число факторов.

6. Финальную оценку трудоемкости $\tilde{S}$ получают за счет учета влияния общепроектных рисков на общую трудоемкость разработки информационной системы с учетом влияния факторов $\tilde{S}_F$, полученную на предыдущем шаге, по формуле:

$$\tilde{S} = \tilde{S}_F + \sum_{j=1}^{L} \tilde{S}_F Pr_j Tr_j, \tag{9}$$

где Pr_j — L-R-представление значения вероятности наступления j-го риска; Tr_j — L-R-представление значения величины влияния риска; L — число рисков, влияющих на выполнение проекта в целом.

После получения значения трудоемкости $\tilde{S}$ в виде L-R-представления нечеткого числа необходимо оценить срок завершения, а также возможность завершения проекта к заданному сроку с помощью разработанного алгоритма. Для этого будем использовать систему нечеткого вывода, работающую по схеме Такаги—Сугено—Канга (часто используют сокращенное название Сугено), в связи с тем, что в ней, по сравнению с другими системами, в качестве заключения правил используются функции от входных переменных, которые можно задать в явном виде. В классической системе Сугено используется

следующий алгоритм вычисления выходного значения системы.

1. Этап фаззификации. Четкие значения входных переменных x приводят расчет к нечетким значениям посредством вычисления степени истинности продукционного правила "x_i есть A_i^j" как расчет соответствующего значения $\mu_A(x_i)$.

2. Этап нечеткого вывода. С использованием полученных на этапе 1 значений степеней истинности вычисляют результирующие значения для каждого j-го правила из набора в m правил:

$$R_j : \text{ЕСЛИ}\left(x_1 \text{ есть } A_1^j\right),\ldots, \\ \text{И}\left(x_n \text{ есть } A_n^j\right)\text{ТО } y = f_j\left(x_1, x_2, \ldots, x_n\right), \tag{10}$$

где $f_j(x_1, x_2,..., x_n)$ — заданная пользователем функция от n входных переменных; A_n^j — терм множества входной переменной; x_n — четкое значение входной переменной.

Истинность продукционного правила α вычисляют по формуле

$$\alpha_j = \bigcap_{i=1}^{n} \mu_{A_i^j}(x_i),$$

где $\cap$ — операция пересечения нечетких множеств (t-норма). В разрабатываемой модели в качестве оператора t-нормы будем использовать, например, оператор умножения (оператор PROD).

3. Этап дефаззификации. Получение четкого значения выхода системы нечеткого логического вывода y выполняется по формуле

$$y = \frac{\sum_{j=1}^{m} \alpha_j f_j\left(x_1, x_2, \ldots, x_n\right)}{\sum_{j=1}^{m} \alpha_j},$$

где α_j — степень истинности j-го правила; $f_j(x_1, x_2, ..., x_n)$ — функция от n входных переменных, используемая в качестве консеквентна j-го правила; m — число правил логического вывода в системе.

Представленный выше алгоритм не предназначен для работы с нечеткими значениями входных переменных, в связи с чем модифицируем алгоритм Сугено посредством переопределения этапов фаззификации и дефаззификации. На этапе фаззификации для определения истинности нечеткой входной переменной будем использовать меру сходства между функциями принадлежности. Наиболее часто в качестве меры сходства выбирают операцию нечеткой конъюнкции (s-норму) [21] в виде оператора MIN или алгебраического произведения PROD. Однако указанные меры сходства не учитывают ширину функций принадлежности. При использовании этих операторов функции принадлежности будут считаться равными (мера истинности равна 1) в случае, если совпадает ядро функций принадлежности. Для учета ширины функций принадлежности в качестве меры сходства возьмем отношение площади пересечения под функциями принадлежности к суммарной площади. Расчет меры сходства двух функций принадлежности $\mu(\tilde{x})$ будем проводить по формуле

$$\mu(\tilde{x}) = \frac{\int_{-\infty}^{+\infty} \min\left(\mu_A(x), \mu_{\tilde{x}}(x)\right) dx}{\int_{-\infty}^{+\infty} \max\left(\mu_A(x), \mu_{\tilde{x}}(x)\right) dx}, \tag{11}$$

где $\mu_A(x)$ — функция принадлежности входного терма лингвистической переменной A; $\mu_{\tilde{x}}(x)$ — функция принадлежности входной переменной. При использовании формулы (11) $\mu(\tilde{x})$ будет равно 1 только в том случае, если ширина функций принадлежности и их ядра совпадают.

Недостатком формулы (11) является отсутствие возможности применения открытых функций принадлежности (таких как сигмоидные) для описания лингвистических переменных на их границах. В этом случае целесообразно использовать отношение площади пересечения под функциями принадлежности к площади под функцией, для которой определяется степень принадлежности. Другими словами, определяется, какая часть рассматриваемого множества совпадает с входным множеством. В дальнейшем расчет будем вести по формуле

$$\mu(\tilde{x}) = \frac{\int_{-\infty}^{+\infty} \min\left(\mu_A(x), \mu_{\tilde{x}}(x)\right) dx}{\int_{-\infty}^{+\infty} \mu_{\tilde{x}}(x)\, dx},$$

где $\mu_A(x)$ — функция принадлежности входного терма лингвистической переменной A; $\mu_{\tilde{x}}(x)$ — функция принадлежности нечеткой входной переменной.

Правила нечеткого вывода формируются также по формуле (6), за исключением того факта, что в качестве аргументов функций, используемых в заключениях правил, могут использоваться нечеткие переменные. В этом случае результатом функции будет нечеткое число в виде функции принадлежности.

В связи с тем обстоятельством, что для выполнения арифметических операций с применением формул (2)—(5), в качестве операндов должны использоваться нечеткие числа, а в качестве входных переменных для модифицированной системы нечеткого логического вывода могут использоваться четкие числа, введем правило формирования нечеткого числа $\tilde{x}$ в виде L-R-представления на основе четкого значения числа x:

$$\tilde{x} = G(x) = \left(g_1(x),\ x,\ g_2(x)\right). \tag{12}$$

Здесь $G(x)$ — общая запись функции формирования нечеткого числа $\tilde{x}$ на основе четкого значе-

ния x; $g_1(x)$ — функция вычисления ширины левой части функции принадлежности; $g_2(x)$ — функция вычисления ширины правой части функции принадлежности. Согласно этой формуле ядром нечеткого представления числа будет значение x, функции $g_1(x)$ и $g_2(x)$ обеспечивают фаззификацию (могут быть как константой, так и некоторыми функциями от четкого значения переменной). Формулу расчета результирующего значения модифицированной системы нечеткого логического вывода для нечетких значений представим в следующем виде:

$$\tilde{y} = \frac{\sum_{j=1}^{m} G_{j1}(\alpha_j) f_j\left(G_{j2}(x_1), G_{j2}(x_2), \ ..., \ G_{jm}(x_{m-1}), \ ..., \tilde{x}_n\right)}{\sum_{j=1}^{m} G_{j1}(\alpha_j)}, \quad (13)$$

где $\tilde{y}$ — нечеткое значение выходной переменной системы нечеткого логического вывода; $f_j(\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_n)$ — функция от n нечетких входных переменных; $G(x)$ — функция приведения четкого значения переменной к нечеткому, вычисляемая по формуле (12); α_j — оценка истинности j-го правила.

Кроме рисков, непосредственно влияющих на трудоемкость разработки, есть категория рисков, влияющая только на срок разработки, но не влияющая на трудоемкость. Как пример: задержки в поставке образцов оборудования или задержки в передаче технической документации по интеграции со смежными информационными системами. В случае возникновения таких рисков трудоемкость разработки информационной системы существенно не изменяется, а увеличивается только срок разработки. Для учета рисков такого типа итоговую оценку срока разработки рассчитывают по формуле

$$\tilde{y}_f = \tilde{y} + \sum_{i=1}^{N} \tilde{y} Pr_i Tr_i, \quad (14)$$

где N — число выявленных руководителем проектов или членами команды рисков; Pr_i — вероятность возникновения риска; Tr_i — величина влияния риска, заданная в виде L-R-представления нечетких чисел. На основе формулы (14) получим формулу расчета оценки возможности завершения проекта к заданному сроку x:

$$p(x) = \frac{\int_{-\infty}^{x} \mu_y(x) d(x)}{\int_{-\infty}^{+\infty} \mu_y(x) d(x)}. \quad (15)$$

Здесь $\mu_y(x)$ — функция принадлежности, полученная по формуле (1), за счет подстановки параметров L-R-представления нечеткого числа, полученных по формуле (14).

Алгоритм калибровки модели

В качестве исходных данных для калибровки предложенной модели необходимо брать фактические значения сроков и трудозатрат разработки информационной системы, полученных по завершению проекта. Однако для более точной настройки желательно сохранять и использовать в корректировке оценки (по числу требований, их трудоемкости, возможным рискам), полученные на начальном этапе проекта.

В этом случае алгоритм подстройки для каждого проекта, по которому есть хронологические данные, будет выглядеть следующим образом.

Шаг 1. Фиксирование фактического срока завершения проекта Y. Если есть сработавшие риски (из использовавшихся на этапе оценки), то срок уменьшается на значение сработавшего риска, и получается срок проекта за вычетом сработавших рисков Y^r. Если в процессе проекта сработали новые риски, оценка которых не проводилась, то в библиотеку добавляют новые риски, но их влияние на оценку срока не учитывают.

Шаг 2. Вычисление результирующего срока проекта за вычетом времени, затраченного по причине возникновения рисков, связанных со сроками разработки. На основе его формирование значения ошибки оценки срока выполнения проекта:

$$E = \breve{y}_{n-1} - G(Y^r),$$

где Y^r — фактическое значение срока выполнения проекта. Корректировка параметров модели нечеткого логического вывода осуществляется методом градиентного спуска [22] по формуле

$$\Delta\tilde{f}(k) = \tilde{\alpha}\frac{d\tilde{e}^2(k-1)}{d\tilde{f}(k-1)} + \beta\Delta\tilde{f}(k-1),$$

где $\tilde{e}$ — ошибка выходного значения оценки срока; $\tilde{f}$ — значение, полученное на выходе системы нечеткого логического вывода; k — шаг, на котором проводится корректировка параметров системы; $\tilde{\alpha}$ — коэффициент скорости подстройки значений системы; β — коэффициент поправки, его значение лежит в интервале [0, 1], при этом, чаще всего его полагают равным 0,9.

Шаг 3. Фиксирование фактической трудоемкости проекта. Из списка требований, на основе реализации которых рассчитана трудоемкость, отбирают требования, которые были изначально учтены при оценке трудоемкости разработки, берут их фактическую трудоемкость и делят на коэффициент типа системы T. После этого для изначально выбранного лингвистического значения требования калибруют базовую шкалу трудоемкости требования по формуле

$$Rq_i^j = \begin{cases} a_{1(n)} = a_{1(n-1)} + 0,3\dfrac{b_{n-1} - r^*}{b_{n-1}} \\ b_n = \dfrac{b_{n-1} - r^*}{b_{n-1}} \\ a_{2(n)} = a_{2(n-1)} + 0,5\dfrac{b_{n-1} - r^*}{b_{n-1}}, \end{cases}$$

где r^* — фактическое значение трудоемкости для выбранного j-го лингвистического значения i-го требования Rq.

Сравнение результатов применения предложенной модели с моделью COCOCMO II

Сравним целевую (синтезированную) модель с моделью COCOCMO II, для чего рассмотрим проект по разработке информационной системы обеспечения грузоперевозок. В качестве исходных данных примем основные требования к системе и их оценку (табл. 1). Для расчета по модели COCOMO II прогнозируемый размер требований выражен в тысячах строк кода (KLOC — kilo lines of code). В табл. 2 представлены идентифицированные на начальном этапе риски, в табл. 3 перечислены факторы, влияющие на трудоемкость разработки системы, выделенные на начальном этапе проекта.

Для расчета оценки срока выполнения будем использовать модифицированную систему нечеткого логического вывода с двумя входными лингвистическими переменными, лингвистические значения и L-R-представления функций принадлежности термов которых представлены в табл. 4. Знак ∞ обозначает, что будет сохраняться значение функции принадлежности, равное 1 на интервале $[b, +\infty)$ для правой части функции принадлежности ($a_2 = \infty$) и на интервале $(-\infty, b]$ для левой части функции принадлежности ($a_1 = \infty$). Функции, выполняющие операцию размытия четкого значения переменной, для

Таблица 1

Основные требования к системе и их оценки

Обозначение	Описание	Тип системы	Коэффициент типа системы T	Лингвистическая оценка трудоемкости	Экспертная оценка для модели COCOMO II, KLOC
Rq_1	Отображение положения грузовиков на карте в WEB-интерфейсе и их статус	WEB	0.9, 1.1, 1,3	Средняя	15
Rq_2	Прием заявок на транспортировку груза со склада, формирование заказа из системы складского учета			Большая	40
Rq_3	Управление заявками: отслеживание жизненного цикла заявки, поиск заявок.			Большая	35
Rq_4	Сбор информации на мобильном терминале, отслеживание статуса, считывание меток, передача информации о местоположении	Мобильное приложение	1, 1,7, 1,4	Маленькая	5

Таблица 2

Описание рисков

Обозначение	Описание	Тип	Вероятность возникновения	Степень влияния
Ri_1	Необходимость доработки интерфейса со стороны системы складского учета в рамках реализации требования Rq_2	Риск требования	Низкая 0,05, 0,2, 0,05	Большая 0,1, 0,85, 0,1
Ri_2	В процессе разработки может потребоваться смена системы управления базами данных	Общий риск	Средняя 0,1, 0,5, 0,1	Средняя 0,1, 0,5, 0,1
Ri_3	Согласование модели устройства для реализации может занять много времени	Риск длительности	Большая 0,1, 0,85, 0,1	Низкая 0,05, 0,2, 0,05

Таблица 3

Факторы, влияющие на трудоемкость разработки

Обозначение	Описание	Влияние фактора
F_1	Предполагаемая сложность продукта	0,8, 1,2, 1,1
F_2	Использование специализируемого оборудования	1,1, 1,3, 1,1

Таблица 4

Лингвистические значения и L-R-представления функций принадлежности термов входных переменных

Переменная 1		Переменная 2	
Размер команды		Трудоемкость выполнения	
Лингвистическое значение входного терма	Параметры функции принадлежности	Лингвистическое значение входного терма	Параметры функции принадлежности входного терма
Очень маленький	10,62, 1, 10,62	Очень маленькая	∞, 100, 133
Маленький	10,62, 25, 10,62	Маленькая	83,3, 412,7, 266,7
Средний	10,62, 50, 10,62	Средняя	166,7, 931,3, 533,5
Большой	10,62, 75, 10,62	Большая	333,3, 1943, 1067
Очень большой	10,62, 100, 10,62	Очень большая	667,5, 4017, ∞

формул (12), (13), будут иметь следующий вид: $g_1(x) = 0{,}3x$, $g_2(x) = 0{,}6x$. Правила нечеткого вывода и значения параметров для функций заключения модели представлены в табл. 5.

При указанных настройках системы нечеткого вывода, для фиксированных значений лингвистической переменной "Трудоемкость", а именно — "Очень маленькая" и "Очень большая", зависимости срока выполнения проекта от размера команды проекта представлены на рис. 3 (см. третью сторону обложки).

На этом рисунке a_1, b, a_2 — параметры функции принадлежности выходной переменной, описываемой по формуле (1). Согласно настроенным параметрам для очень маленьких проектов оптимальным является размер команды разработчиков не более десяти человек. При увеличении размера команды разработчиков возрастают накладные расходы на управление и увеличивается время на организацию работы команды, что приводит к увеличению срока разработки вместо ожидаемого сокращения.

Для исходных значений по формулам (6)—(9) модели получим оценку трудоемкости разработки проекта в виде L-R-представления $\tilde{S} = [73{,}5, 1051{,}2, 416{,}7]$.

Нечеткая оценка длительности разработки, рассчитанная с использованием модернизированной

Таблица 5

Правила нечеткого вывода и значения параметров для функции заключения

№ правила	Размер команды	Трудоемкость	Параметр выходной функции	№ правила	Размер команды	Трудоемкость	Параметр выходной функции
1	Очень маленький	Очень маленькая	0,81	14	Средний	Большая	0,75
2	Очень маленький	Маленькая	0,84	15	Средний	Очень большая	0,88
3	Очень маленький	Средняя	0,86	16	Большой	Очень маленькая	0,23
4	Очень маленький	Большая	0,88	17	Большой	Маленькая	0,24
5	Очень маленький	Очень большая	0,91	18	Большой	Средняя	0,65
6	Маленький	Очень маленькая	0,25	19	Большой	Большая	0,68
7	Маленький	Маленькая	0,33	20	Большой	Очень большая	0,85
8	Маленький	Средняя	0,68	21	Очень большая	Очень маленькая	0,22
9	Маленький	Большая	0,78	22	Очень большой	Маленькая	0,23
10	Маленький	Очень большая	0,9	23	Очень большой	Средняя	0,5
11	Средний	Очень маленькая	0,23	24	Очень большой	Большая	0,54
12	Средний	Маленькая	0,32	25	Очень большой	Очень большая	0,75
13	Средний	Средняя	0,68				

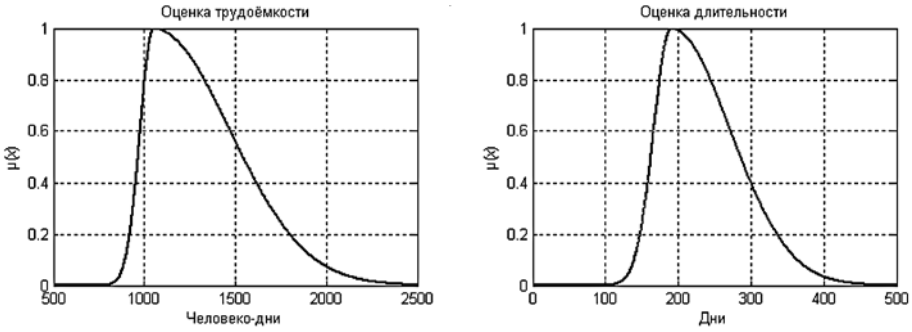

Рис. 4. Графики функций принадлежности нечетких переменных оценок трудоемкости и длительности, полученных в результате расчета

системы нечеткого вывода, получилась равной $\tilde{y}$ = [23,7, 162,7, 80,5] С учетом рисков увеличения длительности нечеткая оценка длительности равна $\tilde{y}_f$ = [23,8, 190,3, 80,9]. Графики функций принадлежности результирующих оценок трудоемкости и длительности представлены на рис. 4. По формуле (15) проведем расчет и построим зависимость оценки трудоемкости и длительности выполнения проекта для его завершения к заданному сроку.

Результаты расчетов зависимости срока проекта и возможности завершения проекта к заданному сроку представлены на рис. 5. Согласно данным, представленным на рис. 5, завершить выполнение проекта со степенью возможности 0,2 можно за 186 рабочих дней, со степенью возможности 0,5 — за 227 рабочих дней, со степенью возможности 0,8 — за 281 рабочий день.

Проведем расчет трудоемкости и номинальной длительности проекта по методу COCOMO II [3]. Номинальную трудоемкость разработки информационной системы рассчитывают по формуле

$$S = A(Size)^B, \qquad (16)$$

где A = 2,94, B = 0,91 — калибровочные константы; $Size$ — предполагаемый размер системы в KLOC.

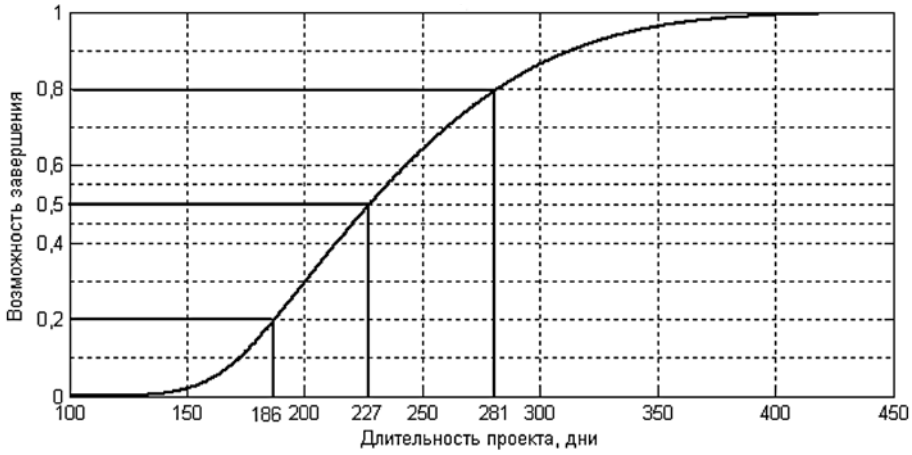

Рис. 5. Рассчитанная зависимость срока проекта и возможности завершения проекта к заданному сроку

Номинальную длительность проекта рассчитывают по формуле [3]

$$D = CS^E, \qquad (17)$$

где C = 3,67, E = 0,28 — калибровочные константы. На основе исходных данных, представленных в табл. 1, рассчитанные значения по формулам (16) и (17) будут S = 185 человеко-месяцев = 4078 человеко-дней (в расчете принималось 22 рабочих дня в человеко-месяце); D = 15,8 месяцев.

Сравнение результатов оценки срока выполнения проекта, вычисленной с использованием предложенной модели и модели COCOMO II, приведены в табл. 6. В общем случае оценки срока и трудоемкости, полученные с использованием предложенной модели, меньше по сравнению с классической моделью COCOMO. Такой результат объясняется тем, что параметры предложенной модели подстроены на основе пяти проектов среднего размера в одной организации, тогда как для модели COCOMO использовались характеристики около 60 крупных проектов разных организаций. В связи с этим обстоятельством для повышения качества результатов, полученных с использованием текущей модели, необходима ее подстройка на основе большего числа проектов.

Таблица 6

Сравнение результатов оценки по двум моделям

Параметр	Предложенная модель	COCOMO II
Трудоемкость, человеко-дни	830...2500	4078
Длительность, дни	135...400	474

Заключение

Разработанная модель расчета трудоемкости и сроков разработки информационных систем позволяет получить значения оценки срока и трудоемкости разработки меньшие, чем с использованием модели COCOMO II. Однако следует отметить, что предложенный подход позволяет дать более объективное обоснование полученной оценке. Рассмотренные в статье параметры модели получены на основе небольшой выборки проектов одной организации. Для получения более точных оценок (в рамках отрасли) и более точной калибровки

параметров модели в целях ее широкого использования необходимо дополнительное исследование. Основными достоинствами предложенной модели являются интуитивно более высокий уровень понимания (не требуется угадывать число строк кода или функциональных точек будущего проекта на этапе инициации проекта), а также зависимость только от данных, имеющихся на начальном этапе проекта.

Недостатком разработанной модели является отсутствие возможности изменения параметров L-R-представления функций принадлежности входной переменной "Размер команды" относительно значения лингвистической переменной "Трудоемкость выполнения". Например, для больших проектов команда исполнителей в 30 человек может быть относительно небольшой, тогда как для малых проектов команда того же размера является большой или даже очень большой. Последнее обстоятельство влияет на срабатывание соответствующих правил в системе нечеткого логического вывода и исключает ее более тонкую подстройку. В качестве цели дальнейшего исследования можно отметить поиск механизмов подстройки функций принадлежностей входных термов одной входной лингвистической переменной относительно зафиксированного параметра другой входной переменной для реализованной системы нечеткого логического вывода.

Работа выполнена при финансовой поддержке РФФИ (грант 16-06-00486).

Список литературы

1. **Макконел С.** Сколько стоит программный проект. СПб.: Питер, 2007. 297 с.
2. **McConnell S.** Software Estimation: Demystifying the Black Art (Developer Best Practices). Microsoft Press, 2006. 319 p.
3. **Boehm B. W., Abts C., Brown A. W.** et. al. Software Cost Estimation with COCOMO II 1st Edition. Prentice Hall, 2000. 368 p.
4. **Manalif E.** Fuzzy Expert-COCOMO Risk Assessment and Effort Contingency Model in Software Project Management: Ph. D. dissertation Canada, 2013. 401 p.
5. **Valerdi R.** The constructive systems engineering cost model (COSYSMO): Ph.D. dissertation USA, 2005. 152 p.
6. **Этингоф Е. В.** Оценка затрат на информационные системы // Управление экономическими системами: электронный научный журнал. 2013. № 12 (60). С. 33—45.
7. **Брукс Ф.** Мифический человеко-месяц, или Как создаются программные системы. СПб.: Символ-Плюс, 2010. 304 с.
8. **Гольфанд И. Я., Хлебутин П. С.** Оценка трудозатрат разработки программной компоненты // Труды ИСА РАН. 2005. Т. 15. С. 125—135.
9. **Максимов И. Б., Столяр В. П., Богомолов А. В.** Прикладная теория информационного обеспечения медико-биологических исследований. М.: Бином, 2013. 311 с.
10. **Щеглов И. Н., Богомолов А. В., Печатнов Ю. А.** Исследование влияния репрезентативности обучающей выборки на качество работы методов распознавания образов // Нейрокомпьютеры: разработка, применение. 2002. № 9—10.
11. **Миронов С. В., Куликов Г. В.** Технологии контроля безопасности автоматизированных систем на основе структурного и поведенческого тестирования программного обеспечения // Кибернетика и программирование. 2015. № 5. С. 158—172.
12. **Голосовский М. С.** Модель оценивания погрешностей прогнозирования сроков разработки программного обеспечения // Программные системы и вычислительные методы. 2015. № 3. С. 311—322.
13. **Голосовский М. С.** Информационно-логическая модель процесса разработки программного обеспечения // Программные системы и вычислительные методы. 2015. № 1. С. 59—68.
14. **Basha S., Dhavachelvan P.** Analysis of Empirical Software Effort Estimation Models // International Journal of Computer Science and Information Security. 2010. Vol. 7, No. 3. P. 68—77.
15. **Cao L.** Estimating Agile Software Project Effort: An Empirical Study. AMCIS, 2008, 401 p.
16. **Руководство** к своду знаний по управлению проектами (Руководство PMBOK®) — Пятое издание. Project Management Institute, 2013. 614 с.
17. **Jørgensen M**. Unit effects in software project effort estimation: Work-hours gives lower effort estimates than workdays // Journal of Systems and Software. 2016. Vol. 117. P. 274—281.
18. **Molokken-Ostvold K., Haugen N.** Combining Estimates with Planning Poker — An Empirical Study // Software Engineering Conference ASWEC, 2007. P. 349—358.
19. **Cohn M.** User Stories Applied. For Agile Software Development. Addison Wesley, 2004, 304 p.
20. **Cohn M.** Agile Estimating and Planning. Prentice Hall, 2005. 368 p.
21. **Пегат А.** Нечеткое моделирование и управление. 2-е издание. М.: БИНОМ. Лаборатория знаний, 2013. 798 с.
22. **Борисов В. В., Круглов В. В., Федулов А. С.** Нечеткие модели и сети. М.: Горячая линия — Телеком, 2007. С. 284.

Model for Estimation Software Development Effort and Duration on Initiation Project Phase

M. S. Golosovskiy, golosovskiy@yandex.ru, Moscow Witte University, Moscow, 115432, Russian Federation

Corresponding author:

Golosovskiy Mikhail S., Project Manager**,** Moscow Witte University, Moscow, 115432, Russian Federation, e-mail: golosovskiy@yandex.ru

Received on July 04, 2016
Accepted on July 21, 2016

A number of models exist for estimating software development and duration. The most popular are COCOMO II, COSYSMO, and SLIM. Most of them use a line of codes as input data and do not give interval estimation results. In this article, a model for estimating software development effort and duration on the initiation project phase is

proposed. The result which can be reasonably anticipated is based on the criteria outlined in Steven C. McConnell's research. The input data for the model, taken from the project charter, contains a description of high level requirements and risks. The input data for the model use fuzzy linguistic variable forms represented by Gaussian combination membership functions. Calculations are based on operations with fuzzy numbers. For the estimation of project duration, a modified Takagi-Sugeno (TSK) fuzzy inference model has been used. TSK models usually have a crisp input and output values. For the modified model, a method for calculating the degree to which fuzzy input belong to fuzzy sets via membership functions has been suggested. In addition, proposed a method for calculating fuzzy output of TSK model. The fuzzy output from the modified TSK model has been used for calculating the cumulative distribution function of the project duration. Finally, a comparison has been made between the proposed model and COCOMO II. It shows that the proposed model is more optimistic in effort estimation than COCOMO II.

Keywords: *Sugeno system, fuzzy system, fuzzy variable, fuzzification, COCOMO II, project charter, expert judgments, software development effort, estimate project duration, COSYSMO*

Acknowledgements: *This work was supported by the Russian Foundation for Basic Research, project nos. 16-06-00486.*

For citation:

Golosovskiy M. S. Model For Estimation Software Development Effort and Duration on Initiation Project Phase, *Programmnaya Ingeneria,* 2016, vol. 7, no. 10, pp. 446—455.

DOI: 10.17587/prin.7.446-455

References

1. **Makkonel S**. *Skol'ko stoit programmnyj proekt* (How much software project costs), Saint Petersburg, Piter, 2007, 297 p. (in Russian).

2. **McConnell S.** *Software Estimation: Demystifying the Black Art* (Developer Best Practices), Microsoft Press, 2006, 319 p.

3. **Boehm B. W., Abts C., Brown A. W., Chulani S., Clark B. K., Horowitz E., Madachy R., Reifer D. J., Steece B.** *Software Cost Estimation with COCOMO II*. 1st Edition, Prentice Hall, 2000, 368 p.

4. **Manalif E.** Fuzzy Expert-COCOMO Risk Assessment and Effort Contingency Model in Software Project Management: Ph. D. dissertation Canada, 2013, 401 p.

5. **Valerdi R**. The constructive systems engineering cost model (COSYSMO): Ph. D. dissertation USA, 2005. 152 p.

6. **Jetingof E. V.** Ocenka zatrat na informacionnye sistemy (Cost estimation of information systems), *Upravlenie jekonomicheskimi sistemami: jelektronnyj nauchnyj zhurnal,* 2013, № 12 (60), pp. 33—45. (in Russian).

7. **Bruks F.** *Mificheskij cheloveko-mesjac, ili kak sozdajutsja programmnye sistemy* (The Mythical Man-Month: Essays on Software Engineering), Saint Petersburg, Simvol-Pljus, 2010, 304 p. (in Russian).

8. **Gol'fand I. Ja., Hlebutin P. S.** Ocenka trudozatrat razrabotki programmnoj komponenty, (Estimate software component development effort), *Trudy ISA RAN,* 2005, vol. 15, pp. 125—135 (in Russian).

9. **Maksimov I. B., Stoljar V. P., Bogomolov A. V.** *Prikladnaja teorija informacionnogo obespechenija mediko-biologicheskih issledovanij* (Applied theory of information support of biomedical research), Moscow, Binom, 2013, 311 p. (in Russian).

10. **Shheglov I. N., Bogomolov A. V., Pechatnov Ju. A.** Issledovanie vlijanija reprezentativnosti obuchajushhej vyborki na kachestvo raboty metodov raspoznavanija obrazov (Study of the influence of the representativeness of the training sample on the quality of the pattern recognition methods), *Nejrokomp'jutery: razrabotka, primenenie,* 2002, no. 9—10 (in Russian).

11. **Mironov S. V., Kulikov G. V.** Tehnologii kontrolja bezopasnosti avtomatizirovannyh sistem na osnove strukturnogo i povedencheskogo testirovanija programmnogo obespechenija (Control technology security of automated systems based on structural and behavioral software testing), *Kibernetika i programmirovanie,* 2015, no. 5, pp. 158—172 (in Russian).

12. **Golosovskiy M. S.** Model' ocenivanija pogreshnostej prognozirovanija srokov razrabotki programmnogo obespechenija (Model error estimates forecast the duration of software development), *Programmnye sistemy i vychislitel'nye metody*, 2015, no. 3, pp. 311—322 (in Russian).

13. **Golosovskiy M. S.** Informacionno-logicheskaja model' processa razrabotki programmnogo obespechenija (Information-logical model of software development process), *Programmnye sistemy i vychislitel'nye metody*, 2015, no. 1, pp. 59—68 (in Russian).

14. **Basha S., Dhavachelvan P.** Analysis of Empirical Software Effort Estimation Models, *International Journal of Computer Science and Information Security*, 2010, vol. 7, no. 3, pp. 68—77.

15. **Cao L.** *Estimating Agile Software Project Effort: An Empirical Study*, AMCIS, 2008, 401 p.

16. **Project** management body of knowledge (PMBOK®) Fifth edition, Project Management Institute, 2013, 614 p.

17. **Jorgensen M.** Unit effects in software project effort estimation: Work-hours gives lower effort estimates than workdays, *Journal of Systems and Software*, 2016, vol. 117, pp. 274—281.

18. **Molokken-Ostvold K., Haugen N.** Combining Estimates with Planning Poker — An Empirical Study, *Software Engineering Conference ASWEC*, 2007, pp. 349—358.

19. **Cohn M.** *User Stories Applied. For Agile Software Development*, Addison Wesley, 2004, 304 p.

20. **Cohn M.** *Agile Estimating and Planning,* Prentice Hall, 2005, 368 pp.

21. **Pegat A.** *Nechjotkoe modelirovanie i upravlenie* (Fuzzy modeling and control), Second edition, Moscow, BINOM. Laboratorija znanij, 2013, 798 p. (in Russian).

22. **Borisov V. V., Kruglov V. V., Fedulov A. S.** *Nechjotkie modeli i seti* (Fuzzy models and networks), Moscow, Gorjachaja linija Telekom, 2007, 284 p. (in Russian).

УДК 004.056.57 DOI: 10.17587/prin.7.456-463

С. В. Жернаков, проф., д-р техн. наук, зав. кафедрой, e-mail: zhsviit@mail.ru,
Г. Н. Гаврилов, аспирант, e-mail: grigorijgavrilov@mail.ru,
Уфимский государственный авиационный технический университет

Методика обнаружения вредоносных программ в операционных системах для мобильных устройств (на примере операционной системы Android)

Представлены результаты анализа защищенности операционной системы Android для мобильных устройств и формализации образцов вредоносных программ в целях выявления признаков, присущих их поведению. На основе полученной информации разработана экспериментальная выборка, описывающая поведение двух типов программ: вредоносные и безопасные, осуществлен выбор наилучшего метода классификации данной выборки путем проведения экспериментов с применением различных методов классификации (классических, нейросетевых и машины опорных векторов). Поставленная задача повышения эффективности обнаружения вредоносных программ решена с использованием разработанной методики на основе машины опорных векторов и нечеткой логики. Данная методика реализована в виде исследовательского прототипа системы обнаружения вредоносных программ.

Ключевые слова: *Android, вредоносная программа, нейронные сети, машина опорных векторов, нечетная логика, методика, операционная система*

Введение

Операционная система (ОС) для мобильных устройств Android нацелена на охват широкого круга пользователей. В настоящее время она занимает 75...80 % рынка всех мобильных устройств. Эта ОС имеет широкие возможности за счет минимального числа ограничений по отношению к перечню доступных функций, что отрицательно влияет на ее защищенность в целом. Данная ОС имеет открытый исходный код, что позволяет модифицировать ее (встраивать вредоносные программы) [1—3], а также большую свободу действий в отношении доступных функциональных возможностей. Перечисленные факторы обусловливают удобство заражения вредоносными программами и их распространения. Популярность и развитие современной сети Интернет позволяет с большой скоростью распространить вредоносные программы на множество мобильных устройств. Таким образом, угроза со стороны вредоносных программ является актуальной для мобильных устройств, работающих под управлением ОС Android. Встроенная модель безопасности ОС Android имеет надежные механизмы защиты, но обладает рядом недостатков по отношению к вредоносным программам [4]. Сторонние антивирусные программы, работающие по классическому методу, основанному на сигнатурном анализе, обладают недостатком, согласно которому новая или модифицированная вредоносная программа при отсутствии в антивирусной базе сигнатур не будет обнаружена. Изложенные выше соображения послужили поводом для исследований в области обнаружения вредоносных программ на основе их поведенческого характера. Анализ отечественных и зарубежных публикаций [5—10] по данной тематике показывает, что работы в данной области активно ведутся. Однако в результатах этих работ отсутствуют практические рекомендации, а также качественные и количественные характеристики разработанных программных проектов для систем комплексной защиты средств мобильной связи ОС Android. Цель работы, результаты которой приведены в статье, — повышение эффективности обнаружения вредоносных программ в ОС Android для мобильных устройств путем разработки методики обнаружения вредоносных программ на основе анализа их поведенческого характера.

Анализ поведенческого характера вредоносных программ и описание разработанной экспериментальной выборки

В целях выявления признаков, необходимых для описания поведения вредоносных программ, выполнен анализ перечня разрешений системы раз-

	1	2	3	4	5	6	7	8	9	10	...	162	163	164	165
1	0	0	0	0	0	0	0	0	0	0	...	1	40	70	ok
2	0	0	0	0	0	0	0	0	0	0	...	0	0	11	ok
3	1	0	0	0	0	0	0	0	0	0	...	0	95	10	virus
4	1	1	1	1	1	1	1	1	1	1	...	1	41	0	virus
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	1	1	0	0	0	0	1	1	0	0	...	0	42	12	virus
96	1	0	0	0	0	1	1	0	1	0	...	0	41	11	virus
97	0	0	0	0	0	1	1	1	0	0	...	0	41	10	ok
98	0	0	1	0	0	0	0	0	0	0	...	1	39	9	ok
99	0	0	0	0	0	0	0	0	0	0	...	1	38	8	ok
100	0	0	1	0	0	0	0	0	0	0	...	0	36	7	ok

Рис. 1. Фрагмент экспериментальной выборки

решений, системных вызовов и формализация кода образцов вредоносных программ ОС для мобильных устройств [11]. Согласно полученным данным разработана экспериментальная выборка, описывающая поведение как вредоносных, так безопасных программ. Фрагмент выборки представлен на рис. 1. Она представляет собой экспериментальную выборку, заданную в бинарной форме (0 — отсутствует, 1 — присутствует), и включает в себя 100 строк (векторов) поведения программ, содержащих 164 признака. Столбцы с первого по десятый содержат информацию о выявленных в процессе формализации признаках программ. В столбцах с 10-го по 162-й приведен список всех разрешений. Столбцы 163-й и 164-й содержат значения, полученные путем анализа используемых системных процессов в ОС для мобильных устройств.

Постановка задачи. Задача обнаружения сводится к задаче классификации предложенной экспериментальной выборки, описывающей признаки того или иного типа программ (вредоносная или безопасная). Для выбора наиболее подходящего метода классификации были проведены эксперименты с применением следующих методов классификации:

- классических (иерархической кластеризации, метод K-средних);
- нейросетевых (радиально-базисная функция (РБФ), линейная нейронная сеть, персептрон, сеть Ворда, модульная нейронная сеть, прямого распространения, прямого распространения с временным окном, равным 12 шагам, сеть Элмана, рекуррентная нейронная сеть, сеть Кохонена);
- машина опорных векторов на основе РБФ, так как лучший результат показала нейронная сеть на основе РБФ.

Пусть X — множество программ, Y — множество, состоящее из двух классов: virus, ok. В качестве метрики выбрано Евклидово расстояние между объектами. Оно является геометрическим расстоянием в n-мерном пространстве и вычисляется следующим образом:

$$p(x, x') = \left(\sum_{i=1}^{n} (x - x')^2 \right)^{1/2}.$$

Задана конечная экспериментальная выборка объектов:

$$X^m = \{x_1, \ ..., \ x_m\} \subset X,$$

где x — элементы выборки (вектора); m — число векторов.

Требуется разбить выборку на непересекающиеся подмножества, именуемые кластерами, так, чтобы каждый кластер состоял из объектов, близких по метрике p, а объекты разных кластеров существенно отличались по этой метрике. При этом каждому объекту $x_i \in X^m$ приписывается номер кластера X'.

Выбор архитектуры нейронной сети выполняется в соответствии с типом решаемой задачи. Для классификации экспериментальной выборки были выбраны типы нейронных сетей, перечисленные выше.

Для обучения выбранных нейронных сетей применялся метод обратного распространения ошибки. Данный метод обеспечивает настройки весов с учетом многослойной структуры сети. Ошибка обучения на выходе нейронной сети распространяется в обратном направлении к скрытым слоям. Для нейронов выходного слоя значение ошибки вычисляется просто как разность между ожидаемым и реальным выходными значениями.

В результате обучения нейронных сетей были построены графики обучения. В качестве примера на рис. 2 представлен график обучения многослойного персептрона. Число эпох составило 110, скорость обучения 0,1 единицы. На графике отображена зависимость ошибки обучения от числа эпох для

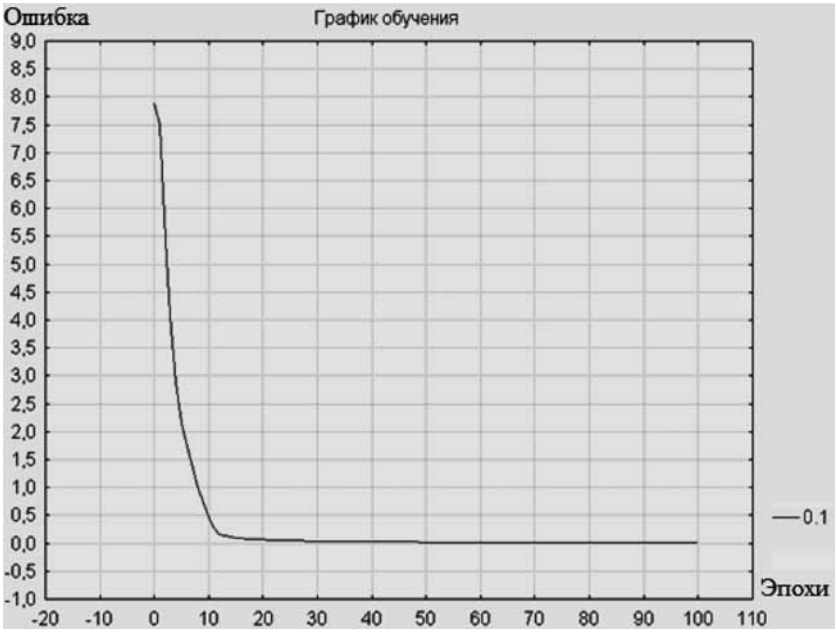

Рис. 2. График обучения многослойного персептрона

многослойного персептрона. Можно отметить, что при приближении к 50-й эпохе ошибка значительно уменьшилась, и минимальная ошибка достигнута при приближении к 100-й эпохе.

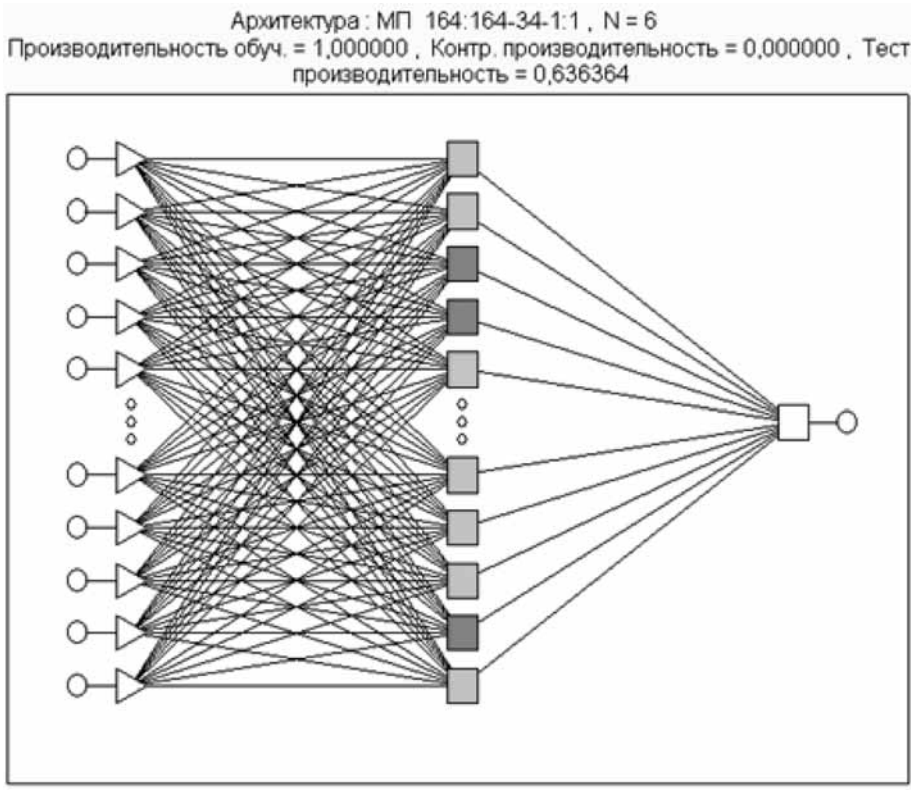

Рис. 3. Архитектура многослойного персептрона — указана активации каждого нейрона для наблюдения 1

Выбор функции активации осуществляется в зависимости от задачи, удобства программно-аппаратной реализации, а также алгоритма обучения. Аргумент функции активации каждого скрытого узла сети радиальной базисной функции представляет собой евклидову норму между входным вектором и центром радиальной функции. Аргумент функции активации каждого скрытого узла сети многослойного персептрона является скалярным произведением входного вектора и вектора синаптических весов данного нейрона. Аргумент функции активации для линейной нейронной сети представляет собой линейную дискриминантную функцию.

На рис. 3 изображена выбранная архитектура для многослойного персептрона с указанием активации каждого нейрона для того наблюдения, которое было задано. Интенсивность окраса нейронов соответствует их активациям, демонстрируя визуальную индикацию активности каждой сети.

На рис. 3 также представлены: производительность обучения в целом, производительность обучения на контрольной выборке, производительность обучения на тестовой выборке и информация о структуре нейронной сети (число входов и скрытых слоев).

Для машины опорных векторов выбрана в качестве ядра радиально базисная функция:

$$\exp\left(-\gamma\left|x_i - x_j\right|^2\right),$$

где γ — размер входного вектора; x_i — входной вектор; x_j — точка, являющаяся центром.

Нейронные сети и машина опорных векторов обучены на первых 68 и тестировалась на остальных 32 значениях векторов программ экспериментальной выборки, приведенной на рис. 1.

Решение задачи: требуется определить функцию $a : X \to Y$, которая любому объекту $x \in X$ ставит в соответствие номер кластера $y \in Y$. Множество Y в некоторых случаях известно заранее, однако чаще ставится задача определить оптимальное число кластеров с точки зрения того или иного критерия качества кластеризации.

В качестве критерия точности и качества работы классификаторов будем использовать следующую формулу:

$$\text{ОК}=\frac{\text{ЧО}\times 100}{\text{ЧН}},$$

где ОК — общий процент, как вредоносных, так и безопасных программ ошибки классификации; ЧО — число ошибок классификации; ЧН — суммарное число наблюдений.

Исследовательский эксперимент с применением различных методов классификации

В целях выбора походящего метода классификации для решения поставленной задачи был проделан исследовательский эксперимент, в результате которого был выбран наиболее подходящий метод классификации предложенной экспериментальной выборки, который выполняет поставленную задачу с наименьшим числом ошибок первого и второго рода. Экспериментальная выборка была классифицирована различными методами классификации. По результатам исследовательского эксперимента установлено, что точность работы классических методов классификации (иерархической кластеризации и K-средних) невысокая: ошибки первого рода — 26,08 %, ошибки второго рода — 22,7 %. Классификация с применением нейросетевых методов показала лучшие результаты по сравнению с классическими методами преимущественно к большому уровню помех, а именно — когда признаки безопасной программы присутствуют в различном соотношении во вредоносной программе [12]. Результаты проделанных экспериментов по классификации различными методами представлены в табл. 1.

Машина опорных векторов показала лучшие результаты по сравнению с нейронными сетями и классическими методами. Ошибочно выполнена классификация пяти типов программ, процент правильно классифицированных программ составил 72,3 %. Ошибок первого рода — 0 %, ошибок второго рода —

Таблица 1

Общая таблица результатов классификации

Ошибочно классифицированные векторы, %	Правильно классифицированные векторы, %	Число ошибочно классифицированных векторов	Число правильно классифицированных векторов	Всего векторов	Классы	Тип нейронной сети
11	88,9	2	16	18	ok	РБФ
20	80	3	12	15	virus	
11,1	88,8	2	16	18	ok	Линейная
26,6	73,3	4	11	15	virus	
66,6	33,3	12	6	18	ok	Персептрон
0	100	0	15	15	virus	
100	0	48	0	48	ok	Ворда
0	100	0	52	52	virus	
61,1	38,9	11	7	18	ok	Модульная
0	100	0	15	15	virus	
33,3	66,7	6	12	18	ok	Прямого распространения
7,1	92,9	1	14	15	virus	
66,6	33,4	12	6	18	ok	Прямого распространения с временным окном, равным 12 шагам
20	80	3	12	15	virus	
50	50	9	9	18	ok	Элмана
7,1	92,9	1	14	15	virus	
22,2	77,8	4	14	18	ok	Рекуррентная
66,6	33,4	10	5	15	virus	
26,6	73,3	4	11	15	ok	Кохонена
61,1	38,8	11	7	18	virus	
72,3	27,7	5	13	18	ok	Машина опорных векторов
100	0	0	15	15	virus	

27,7 %. В классе virus классификация была выполнена безошибочно и таким образом, что процент правильно классифицированных программ составил 100 %.

Следовательно, машина опорных векторов выполняет классификацию предложенной выборки с меньшим числом ошибок. Данный метод был задействован в качестве основы для разработки общей методики.

Описание методики обнаружения вредоносных программ

На основе проделанных экспериментов был предложен метод обнаружения вредоносных программ. В основе данного метода лежит машина опорных векторов. Поскольку данный метод на фоне большого уровня помех, т. е. размытости кластеров, совершает ошибки классификации, была задействована нечеткая логика [13, 14], позволяющая выполнять дополнительную классификацию, т. е. выполнить коррекцию, а также дополнить результат работы машины опорных векторов с учетом сильных помех. Метод обнаружения вредоносных программ объединяет в себе связку двух методов представленных на рис. 4 [15].

В качестве входных данных выступает разработанная экспериментальная выборка, также на ней проводится обучение машины опорных векторов. В состав обучающей выборки вошли 67 составленных векторов программ. Тестовая выборка включала в себя 33 вектора программ с внесением изменений с целью усложнить задачу обнаружения вредоносных программ. Далее происходит формализация дополнительных признаков, которые задаются в виде функций принадлежности для нечеткой логики. В состав функций принадлежности также входит результат предсказания машиной опорных векторов. Затем происходит формирование базы правил для корректного функционирования нечеткой логики. На основании всех признаков и работы машины опорных векторов получаем результат, выраженный в процентах.

Алгоритм функционирования метода обнаружения вредоносных программ состоит из следующих шагов.

Шаг 1. Извлечение и формирование вектора признаков из программы путем анализа файлов androidmanifest.xml, clases.dex и системных вызовов.

Шаг 2. Подача вектора признаков программы (вредоносная или безопасная программа) на вход SVM-классификатора.

Имеются экспериментальные данные вида $\{(x_1, y_1), \ ..., \ (x_m, y_m)\}$, где каждому объекту x_i поставлено в соответствие число y_i, принимающее знание 1 или −1 в зависимости от того, какому классу (virus или ok) принадлежит объект x_i. Каждому классу поставлен в соответствие вектор числовых значений характеристик $x_i = \left(x_i^1, x_i^2, \ ..., \ x_i^q\right)$, заданный в бинарной форме, где x_i^j — числовое значение j-ой характеристики для i-го объекта $\left(i = \overline{1, m}, j = \overline{1, q}\right)$. Данная экспериментальная выборка использована для разработки классификатора машины опорных векторов для классификации новых объектов. Задача классификации с учетом теоремы Куна—Таккера эквивалентна двойственной задаче поиска седловой точки функции Лагранжа, которая сводится к задаче квадратичного программирования, содержащей только двойственные переменные:

$$\begin{cases} -L(\lambda) = -\sum\limits_{i=1}^{m} \lambda_i + \frac{1}{2} \sum\limits_{j=1}^{m} \sum\limits_{i=1}^{m} \lambda_i \lambda_j y_i y_j \times \\ \times \exp\left(- < x_i - x_j, x_i - x_j > / \left(2\sigma^2\right) - b\right); \\ \sum\limits_{i=1}^{m} \lambda_i y_i = 0; \\ 0 \leqslant \lambda_i \leqslant C, i = \overline{1, m}, \end{cases}$$

где λ_i — двойственная переменная; x_i — объект из экспериментальной выборки; y_i — число (−1 или 1), характеризующее классовую принадлежность объекта x_i из экспериментальной выборки;

$k\left(x_i, x_j\right) = \exp\left(- < x_i - x_j, x_i - x_j > / \left(2\sigma^2\right)\right)$ — РБФ ядра; σ — параметр сглаживания (общая ширина разброса); b — порог функции активации; C — параметр регуляризации ($C > 0$); m — число объектов в экспериментальной выборке; $i = \overline{1, m}$.

В результате обучения машины опорных векторов определяются опорные векторы, являющиеся векторами характеристик тех объектов x_i из экспериментальной выборки, для которых значения соответствующих им двойственных переменных λ_i отличны от нуля ($\lambda_i \neq 0$). Опорные векторы находятся ближе всего к гиперплоскости, разделяющей классы, и несут всю информацию о разделении классов. Так как задача квадратичного программирования решена, то клас-

Рис. 4. Схема метода обнаружения вредоносных программ

сификация произвольного объекта λ будет выполнена по следующему правилу:

$$\alpha(z) = \text{sign}\sum_{i=1}^{m}\lambda_i y_i \exp\left(- < x_i - x_j, x_i - x_j > /\left(2\sigma^2\right) - b\right),$$

где $b = < w, x_i > - y_i$; $w = \sum_{i=1}^{m}\lambda_i y_i x_i$.

При этом суммирование в правиле выполняется только по опорным векторам.

Шаг 3. Подача блоку системы поддержки принятия решений результата классификации и дополнительных признаков, заданных в виде функций.

В качестве дополнительных признаков (функции принадлежности) для аппарата нечеткой логики были задействованы следующие: результат классификации машиной опорных векторов, число запрашиваемых разрешений, а также совпадения, выявленные в процессе анализа исходного кода вредоносных программ подозрительных классов. На основе общего анализа соотношения данных признаков во вредоносных программах была составлена база правил.

Система поддержки принятия решений основана на аппарате нечеткой логики, работающей по алгоритму Мамдани.

Шаг 4. Анализ результатов на основании правил.

Шаг 5. Вывод результата в процентах.

На основе предложенного метода был разработан исследовательский прототип программы, который представляет собой программу, имитирующую поведение двух типов программ, условий и помех, собирающую статистику и ведущую журнал событий. Главное окно программы представлено на рис. 5.

В целях проверки эффективности работы разработанных методов был осуществлен эксперимент с классификацией 100 типов программ и проведено сравнение полученных результатов с существующими в настоящее время антивирусными программами (табл. 2).

Антивирусные программы обладают низкой эффективностью обнаружения новых вредоносных программ [16], лучший результат составил 60 %, средний результат по всем рассматриваемым образцам составил 23,4 %. Исследовательский прототип модели системы обнаружения вредоносных программ показал лучший результат — 80 % и сравнительно небольшое число ложных срабатываний, равное 8 из 100 рассматриваемых образцов, что составило 19,35 %. При этом не учитывался результат работы аппарата нечеткой логики, который выполнил коррекцию результата классификации машины опорных векторов путем дополнительной классификации, и показал следующий результат: подозрительная 45 % и опасная 65 % программа, дополнив таким образом, работу машины опорных векторов и увеличив показатели эффективности обнаружения.

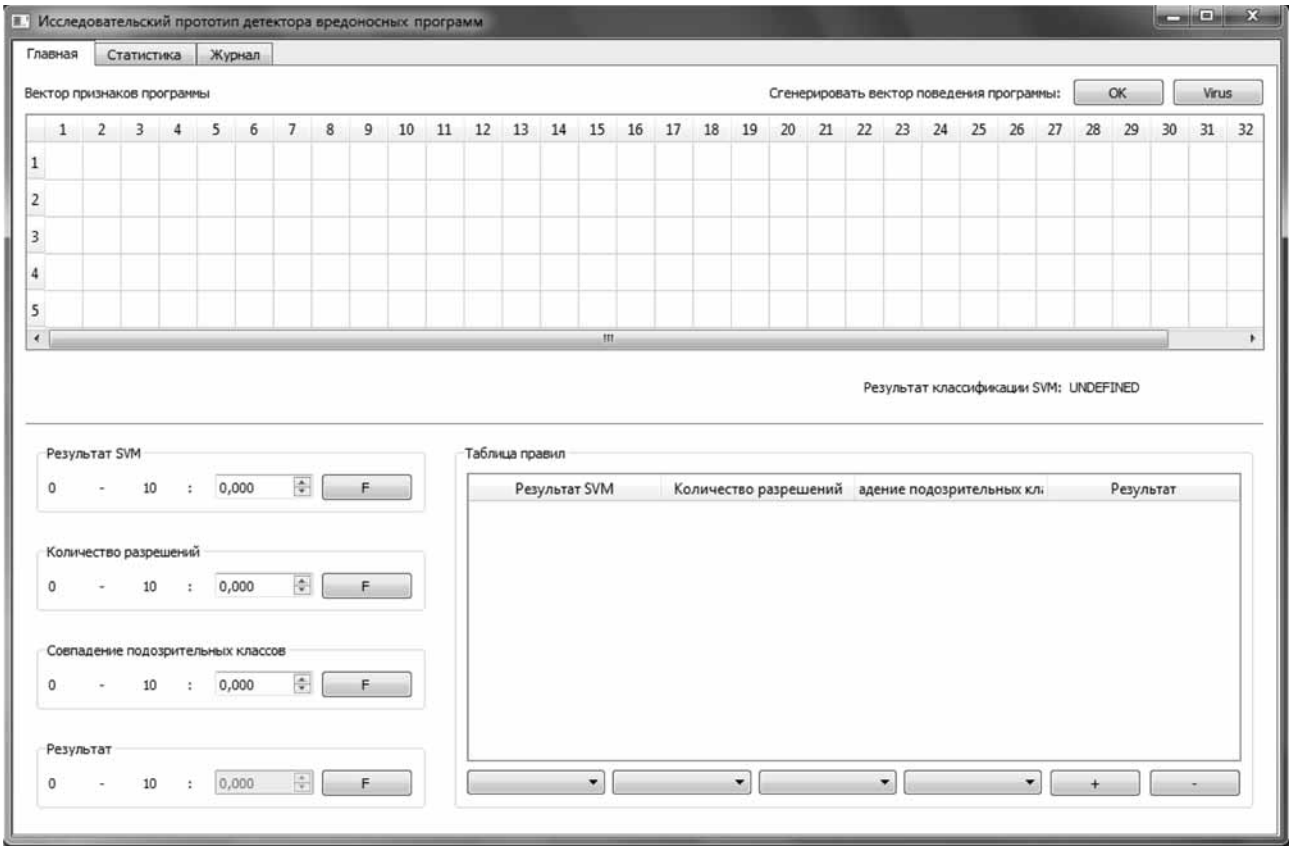

Рис. 5. Главное окно программы

Таблица 2

Результаты обнаружения программ

Программа	Процент обнаружения	Программа	Процент обнаружения	Программа	Процент обнаружения
Система обнаружения вредоносных программ	80,6	Kaspersky	22,2	F-Prot	8,7
GData	60,0	AVG	21,8	Rising	8,2
Ikarus	55,4	Avast	20,8	Fortinet	6,9
Emsisoft	54,8	AhnLab-V3	28,5	VirusBuster	6,6
McAfee-GW-Edition	50,6	Symantec	28,0	Commtouch	6,6
Panda	49,7	Microsoft	21,4	eSafe	6,1
BitDefender	49,5	Comodo	20,7	SUPERAntiSpyware	4,0
AntiVir (AVIRA GmbH)	47,9	TrendMicro	20,2	CAT-QuickHeal	2,5
McAfee	47,9	PCTools	20,0	ClamAV	2,2
F-Secure	46,1	nProtect	19,2	eTrust-Vet	1,4
NOD32 (ESET)	43,6	K7AntiVirus	17,9	Antiy-AVL	1
VIPRE (GFI Software)	43,2	TrendMicro-HouseCall	17,6	Jiangmin	1
DrWeb	36,5	Prevx	16,6	TheHacker	1
Norman	32,8	VBA32	11,4	ViRobot	1
Sophos	22,5				

Заключение

Предложенный метод обнаружения вредоносных программ позволяет выполнять обнаружение путем анализа поведенческого характера программ, а также увеличить эффективность обнаружения, что положительно сказывается на защите информации в мобильных ОС в целом. Данный метод позволяет своевременно обнаружить и нейтрализовать угрозу со стороны как новых, модифицированных, так и имеющихся типов вредоносных программ, а также он может выступать в дополнении к уже имеющимся классическим методам обнаружения.

Список литературы

1. **Sanz B., Santos I., Nieves J., Laorden C., Alonso-Gonzalez I., G. Bringas P.** MADS: Malicious android applications detection through string analysis. Network and System Security, Springer Berlin Heidelberg, 2011. Vol. 5. URL: http://www.researchgate.net/publication/256194745_MADS_Malicious_Android_Applications_Detection_through_String_Analysis

2. **Обнаружен троян**, встроенный в Android-прошивку. URL: http://www.3dnews.ru/904864

3. **Згоба А. И., Маркелов Д. В., Смиров П. И.** Кибербезопасность: угрозы, вызовы, решения // Вопросы кибербезопасности. 2014, № 5(8). С. 30—38. URL: http://cyberleninka.ru/article/n/kiberbezopasnost-ugrozy-vyzovy-resheniya

4. **Жернаков С. В., Гаврилов Г. Н.** Обзор современного состояния защиты информации в мобильных системах // Вестник БГТУ им. В. Г. Шухова. 2016. № 2. С. 171—176.

5. **Сравнения антивирусов,** DLP и других средств защиты. URL: http://www.anti-malware.ru/compare

6. **Fan Yuhui, Xu Ning.** The Analysis of Android Malware Behaviors//International Journal of Security and Its Applications. 2015. Vol. 9. N. 3. URL: http://www.sersc.org/journals/IJSIA/vol9_no3_2015/25.pdf

7. **Dunham K., Hartman S., Quintans M., Morales J. A., Strazzere T.** Android Malware and Analysis. NY, CRCPress, 2015. 91 p.

8. **Saba Arshad Munam Ali Shah, Abid Khan, Ahmed Mansoor.** Android Malware Detection & amp; Protection: A Survey // International Journal of Advanced Computer Science and Applications. 2016. Vol. 7, № 2. URL: https://thesai.org/Downloads/Volume7No2/Paper_62-Android_Malware_Detection_Protection_Survey.pdf

9. **Sanz B., Santos I., Laorden C.** et. al. Mama: Manifest Analysis for Malware Detection in Android // Cybernetics and Systems. 2013. Vol. 44, N. 6—7. P. 469—488. URL: http://paginaspersonales.deusto.es/isantos/papers/2013/2013-Sanz-MAMA.pdf

10. **Wei Wang, Xing Wang, Dawei Feng** et al. Exploring Permission-Induced Risk in Android Applications for Malicious Application Detection // IEEE Transactions on Information Forensics and Security. 2014. Vol. 9. N. 11. P. 1869—1882. URL: https://mine.kaust.edu.sa/Documents/papers/TIFS_Android_permission.pdf

11. **Arp D., Spreitzenbarth M., Hubner M., Gascon H., Rieck K.** DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. NDSS Symposium 2014, Switzerland. 2014. Vol. 4. N. 1. URL: https://user.informatik.unigoettingen.de/~krieck/docs/2014-ndss.pdf.

12. **Жернаков С. В., Гаврилов Г. Н.** Детектирование вредоносного программного обеспечения с применением классических и нейросетевых методов классификации // Вестник ВГУИТ. 2015. № 4. С. 85—92.

13. **Андрейчиков А. В., Андрейчикова О. Н.** Интеллектуальные информационные системы: Учебник. М.: Финансы и статистика, 2004. 424 с.

14. **Головко В. А.** Нейронные сети: обучение, организация и применение. Кн. 4: Учеб. пособие для вузов. М.: ИИРЖР, 2001. 178 с.

15. **Васильев В. И.** Интеллектуальные системы защиты информации: Учебное пособие. М.: Машиностроение, 2013. 82 с.

16. **Зак Ю. А.** Принятие решений в условиях нечетких и размытых данных: Fuzzy-технологии. М.: ЛИБРОКОМ, 2013. 179 с.

Methods of Detection of Malicious Software in Operating Systems for Mobile Devices (for Example, the Android Operating System)

S. V. Zhernakov, zhsviit@mail.ru, **G. N. Gavrilov**, grigrorijgavrilov@mail.ru, Ufa State Aviation Technical University, Ufa, 450008, Russian Federation

Corresponding author:

Gavrilov Grigoriy N., Postgraduate Student, Ufa State Aviation Technical University, 450008, Ufa, Russian Federation
e-mail: grigrorijgavrilov@mail.ru

Received on May 05, 2016
Accepted on July 22, 2016

The results of the analysis of the Android operating system security for mobile devices and the formalization of malware samples in order to identify features inherent in their behavior are presented. Based on the received results, an sample set was developed, describing the behavior of two types of programs: malicious and safe; the best method for classifying this sample was chosen by means of experiments with using different classification methods (classical, neural networks, and support vector machines). The task of improving the efficiency of malware detection was solved using a technique developed for this purpose based on support vector machines and fuzzy logic. This technique is implemented as a research prototype malware detection system.

Keywords: *Android, malware, neural networks, support vector machine, odd logic, technique, operating system*

For citation:

Zhernakov S. V., Gavrilov G. N. Methods of Detection of Malicious Software in Operating Systems for Mobile Devices (for Example, the Android Operating System), *Programmnaya Ingeneria*, 2016, vol. 7, no. 10, pp. 456—463.

DOI: 10.17587/prin.7.456-463

References

1. **Sanz B., Santos I., Nieves J., Laorden C., Alonso-Gonzalez I., G. Bringas P.** MADS: Malicious android applications detection through string analysis. *Network and System Security*, Springer Berlin Heidelberg, 2011, vol. 5, available at: http://www.researchgate.net/publication/256194745_MADS_Malicious_Android_Applications_Detection_through_String_Analysis.

2. **Discovered** a Trojan built into the Android firmware Available: http://www.3dnews.ru/904864 (in Russian).

3. **Zgoba A. I., Markelov D. V., Smirov P. I.** Kiberbezopasnost': ugrozy, vyzovy, reshenija (Cybersecurity. Threats, calls, solutions), *Voprosy kiberbezopasnosti*, 2014, no. 5 (8), pp. 30—38, available at: http://cyberleninka.ru/article/n/kiberbezopasnost-ugrozy-vyzovy-resheniya (in Russian).

4. **Zhernakov S. V., Gavrilov G. N.** Obzor sovremennogo sostojanija zashhity informacii v mobil'nyh sistemah (Overview of the current state of information security in mobile systems), *Vestnik BGTU im. V. G. Shuhova*, 2016, no. 2, pp. 171—176 (in Russian).

5. **Comparisons antivirus**, DLP and other remedies, available at: http://www.anti-malware.ru/compare (in Russian).

6. **Fan Yuhui, Xu Ning.** The Analysis of Android Malware Behaviors. *International Journal of Security and Its Applications*, 2015, no. 3, vol. 9, available at: http://www.sersc.org/journals/IJSIA/vol9_no3_2015/25.pdf.

7. **Dunham K., Hartman S., Quintans M., Morales J. A., Strazzere T.** *Android Malware and Analysis,* NY, CRCPress, 2015, 91 p.

8. **Saba Arshad, Munam Ali Shah, Abid Khan, Ahmed Mansoor**. Android Malware Detection & amp; Protection: A Survey, *International Journal of Advanced Computer Science and Applications*, 2016, vol. 7, no. 2, available at: https://thesai.org/Downloads/Volume7No2/Paper_62-Android_Malware_Detection_Protection_Survey.pdf (Accessed 08 March 2015)

9. **Sanz B., Santos I., Laorden C., Ugarte-Pedrero X., Nieves J., Bringas P. G., Alvarez G.** Mama: Manifest Analysis for Malware Detection in Android, *Cybernetics and Systems*, 2013, vol. 44, no. 6-7, pp. 469—488, available at: http://paginaspersonales.deusto.es/isantos/papers/2013/2013-Sanz-MAMA.pdf

10. **Wei Wang, Xing Wang, Dawei Feng, Jiqiang Liu, Zhen Han, Xiangliang Zhang.** Exploring Permission-Induced Risk in Android Applications for Malicious Application Detection, *IEEE Transactions on Information Forensics and Security*, 2014, vol. 9, no. 11, pp. 1869—1882, available at: https://mine.kaust.edu.sa/Documents/papers/TIFS_Android_permission.pdf

11. **Arp D., Spreitzenbarth M., Hubner M., Gascon H., Rieck K.** DREBIN: Effective and Explainable Detection of Android Malware in Your Pocke, *NDSS Symposium 2014*, Switzerland, 2014, vol. 4, no. 1, available at: https://user.informatik.unigoettingen.de/~krieck/docs/2014-ndss.pdf

12. **Zhernakov S. V., Gavrilov G. N.** Detektirovanie vredonosnogo programmnogo obespechenija s primeneniem klassicheskih i nejrosetevyh metodov klassifikacii (Detection of malicious software using classical and neural network classification methods), *Vestnik VGUIT,* 2015, no. 4, pp. 85—92 (in Russian).

13. **Andrejchikov A. V., Andrejchikova O. N.** *Intellektual'nye informacionnye sistemy: Uchebnik* (Intelligent information systems: Textbook), Moscow, Finance and Statistics, 2004, 424 p. (in Russian).

14. **Golovko V. A.** *Nejronnye seti: obuchenie, organizacija i primenenie.* Kn. 4: Ucheb. posobie dlja vuzov (Neural networks: education, organization and application. Bk. 4: Proc. manual for schools.), Moscow, IIRZHR, 2001, 178 p. (in Russian).

15. **Vasilyev V. I.** *Intellektual'nye sistemy zashhity informacii*: Ucheb. posobie (Intelligent protection of information systems. Textbook), Moscow, Mashinostroenie, 2013, 82 p. (in Russian).

16. **Zach Yu**. *Prinjatie reshenij v uslovijah nechetkih i razmytyh dannyh: Fuzzy-tehnologii* (Decision-making in a fuzzy and blurry data: Fuzzy-technology), Moscow, LIBROKOM, 2013, 179 p. (in Russian).

УДК 004.65 DOI: 10.17587/prin.7.464-471

А. М. Бородин, канд. техн. наук, доцент, e-mail: amborodin@acm.org,
С. Г. Мирвода, ст. преподаватель, e-mail: sergey@mirvoda.com,
С. В. Поршнев, д-р техн. наук, проф., зав. каф., e-mail: s.v.porhsnev@urfu.ru,
Уральский федеральный университет, Екатеринбург

Методы отладки индексов баз данных: опыт применения при разработке информационных систем уровня предприятия

Повреждение индекса может привести к существенным трудностям, начиная с временного отказа в обслуживании, заканчивая потерей или раскрытием секретной информации. Предложены методы для повышения качества и надежности индексных структур при разработке алгоритмов индексирования, использующихся в многомерных системах бизнес-анализа. Рассмотрены варианты применения предложенных методов для реальных ситуаций, возникающих в процессе разработки и отладки индексных структур основной памяти.

Ключевые слова: *базы данных, многомерные данные, индексирование, обеспечение качества, иерархия типов памяти*

Введение

Традиционно базы данных разделяют на два класса: транзакционные [1] и аналитические [2]. Первый класс предназначен для обеспечения задач накопления и долговременного хранения данных, второй же обеспечивает преобразование, агрегацию и другую интерпретацию данных в аналитических целях. Статья является результатом участия авторов в проекте по разработке ядра многомерной базы данных для специфических расчетных задач бизнес-анализа [3, 4]. Существенной сложностью проекта являлись изменяющаяся структура данных и быстро и непредсказуемо обновляющиеся данные. Кроме того, аналитические расчеты при выполнении обладали побочными явлениями, изменяя данные и их структуру.

Полнофункциональный прототип системы был разработан на основе базы данных Microsoft SQL Server. Прототип был внедрен в опытную эксплуатацию, но имел существенный недостаток: низкую производительность. Профилирование показало, что в основном производительность существенно снижалась при многомерных расчетах, измерения в которых были представлены иерархическими классификаторами. Были рассмотрены три варианта архитектуры системы для решения проблем производительности.

1. Использование Microsoft SQL Server Analysis Services (SSAS). Данный вариант основывается на схожести механизмов расчета в SSAS и в разрабатываемой системе.

2. Использование многомерного индекса GiST из PostgreSQL [5, 6].

3. Разработка связующего ядра базы данных, реализующего только необходимые аналитические функции. Использование MS SQL Server в качестве транзакционного хранилища данных.

В результате анализа было выявлено, что существующие базы данных не удовлетворяют требованиям проекта. Иерархии в SSAS не рекомендуются производителем в качестве основных структур для применения в кубах с большим числом измерений. Также в SSAS отсутствуют механизмы управления транзакциями и аналитика в реальном времени. Производительность GiST в PostgreSQL на тот момент была недостаточной (в настоящее время выполняются исследования по улучшению производительности). К тому же использование GiST требовало разработать собственные индексируемые типы данных для определения элементов иерархий в PostgreSQL. С учетом перечисленного было принято решение использовать вариант 3 архитектуры.

Несмотря на значительное упрощение необходимо было разработать полноценное ядро базы данных и индексирующие структуры, способные работать с существенным объемом данных (сотни гигабайт на старте проекта).

Были рассмотрены различные методы многомерного доступа к данным, применены к решаемой задаче и реализованы с необходимым для промышленного применения качеством. В ходе проекта было

отмечено, что реализация алгоритма с промышленным качеством потребляет несравнимо больше ресурсов, чем выбор, оценка производительности и подтверждение возможности применения необходимого алгоритма.

Большинство ресурсов расходовалось в процессе проверки качества разработанного алгоритма индексирования, а именно в процессе подтверждения корректности работы в условиях конкурентного расчета и обновления данных.

В данной статье описаны методы, которые подтвердили свою эффективность во время процесса отладки и обеспечили уверенность в промышленном качестве разработанного ядра базы данных.

1. Алгоритмы и структуры данных

Тестирование и контроль качества в базах данных это широко обсуждаемая тема [7—9]. К сожалению, разработка баз данных промышленного уровня выполняется коммерческими компаниями (за редким исключением известных проектов открытого программного обеспечения, например, PostgreSQL). Это приводит к тому, что опубликованная коммерческими компаниями информация обычно представлена тем, что компания разрешает опубликовать разработчикам и исследователям и не содержит потенциальных рисков потери интеллектуальной собственности. В независимости от этого большинство методов могут быть адаптацией общих методов, стратегий и идеологии отладки [10—13] к специфике разработки алгоритмов для баз данных.

1.1. Буфер страниц оперативной памяти

Специфика алгоритмов баз данных заключается в том, что они спроектированы в расчете на оперирование большим объемом данных, существенно превышающим размер оперативной памяти. Именно поэтому большинство алгоритмов функционируют в заданном буфере памяти. Буфер управляет некоторым образом организованными фрагментами данных, называемыми страницами. Буфер состоит из кадров, имеющих размер страницы. Страница данных идентифицируется адресом и может хранить данные во внешней памяти. Страница может просматриваться и редактироваться посредством буфера. Обычно страница имеет фиксированный размер.

Программный интерфейс буфера содержит функции, которые предоставляют возможность:

1) разместить страницу и вернуть ее адрес;

2) закрепить страницу в буфере и вернуть указатель на вмещающий ее кадр буфера;

3) открепить страницу в буфере;

4) освободить страницу из хранилища;

5) удостовериться, что текущее содержимое станицы записано в дисковое хранилище.

Незакрепленная страница является кандидатом на стирание, если требуется разместить в кадре новую страницу. Страница может быть закреплена в буфере несколько раз, чтобы пометить такую страницу как стираемую требуется соответствующее число откреплений.

Эта стратегия состыковывает внешнее хранилище с буфером оперативной памяти. Для простоты изложения будем считать, что один буфер памяти прикреплен только к одному внешнему хранилищу.

1.2. Раздельный список

Одной из важных (хоть и вспомогательной) задач в СУБД является выделение временных записей и поддержание временных неупорядоченных наборов записей. Временные наборы записей поддерживают только вставку и обход данных. Как правило, данные, находящиеся в таких записях, не нуждаются в сохранении, но сброс их на диск все равно может потребоваться вследствие недостаточного числа буферов памяти. Общим подходом к организации таких наборов записей является структура данных *раздельный список* (*segregated list*) [14]. Обычно раздельный список используется как временное хранилище данных.

Раздельный список оптимизирован для быстрого выделения памяти, скорость обеспечивается за счет способа быстро находить страницу с достаточным местом для хранения новой записи.

Корневая страница содержит адрес стартовой страницы в цепочке страниц с указанной степенью свободного места. Например, одна цепочка страниц с более чем 127 свободными байтами, одна цепочка страниц с более чем 255 байтами и т. д. Для расчета степени свободного места используют различные стратегии.

При появлении новой записи структура находит цепочку с минимальным подходящим свободным местом, извлекает одну страницу из цепочки, переносит запись на страницу и переносит модифицированную страницу в другую цепочку, подходящую по степени свободного места, обновляя стартовые страницы обеих модифицированных цепочек. Описанная структура данных предоставляет эффективный многопоточный доступ для вставки записей существенно разного размера.

1.3. B-дерево

B-дерево это хорошо известная, универсальная и широко используемая структура данных [14, 15]. База данных может быть реализована полностью на B-деревьях. Обычно B-деревья используют для реализации ассоциативных контейнеров типа ключ-значение, для реализации операций присоединения, для получения диапазона данных и для хранения данных с указанным порядком сортировки.

Традиционно B-дерево организует данные в виде дерева, используя структуру данных *сбалансированное дерево.* Каждый узел дерева содержит страницу данных, хранящую полную информацию об узле. Страницы-листья содержат данные, в то время как внутренние страницы содержат только ключи и адреса нижележащих страниц. Основной точкой входа структуры является корневая страница, которая может быть как внутренней, так и страницей данных (если данных достаточно мало, чтобы уместиться на одной странице). B-дерево поддерживает одномерный порядок ключей и позволяет эффективно найти запись по ключу, либо извлечь диапазон последовательных записей с ключами, обеспечивает возможность слияния индексов, присоединения индексов, удаления записей из индекса, пакетную вставку записей в индекс.

В разработанной системе было применено многоверсионное B-дерево [16], поддерживающее многоверсионный режим конкурентной работы. Это означает, что данные в дереве существуют в виде нескольких состояний, для каждой пользовательской сессии. Если данные изменились в одной из сессий, каждая измененная страница копируется и помечается как частная. Когда сессия сохраняет данные, старая корневая страница подменяется новой.

1.4. Разновидности R-дерева

R-дерево это структура, используемая в основном в пространственных базах данных. В то время как B-дерево организует иерархию индексированных данных, выстраивая последовательные диапазоны упорядоченных ключей данных, R-дерево предоставляет структуру выстроенных в иерархию МОП-структур (минимально ограничивающий прямоугольник) в многомерном пространстве независимых ключей. Существует множество разновидностей R-дерева [17], в основном отличающихся алгоритмом вставки, особенно выбором метода разделения узла (страницы) на две части, в момент переполнения данных. Разновидность R*-дерево [18, 19] при отсутствии специальных требований считается наиболее оптимальной.

Обычно R-деревья используются для реализации пространственного поиска, агрегации данных с помощью оконных запросов (запросов многомерных диапазонов) и поиска ближайших соседей.

Разработанная в рамках проекта база данных использует R*-дерево, совмещенное с многоверсионным подходом к управлению доступом (MVCC) [14].

Кроме того, вследствие частых существенных перестроений индекса при получении новых данных из внешних источников, был реализован алгоритм пакетной вставки. Для реализации пакетной вставки был использован алгоритм буферизации VAMsplit [20]. Было установлено, что он наиболее эффективно использует процессорное время и настолько же эффективен, как буферизованный PostgreSQL GiST [5].

2. Методы последовательных тестовых сценариев

При реализации проекта применялась методология разработки через тестирование, основные функции индексирующей подсистемы проверялись модульными тестами. Эти тесты реализовывали перечисленные далее базовые стратегии.

1. Различные операции CRUD (Вставка, Чтение, Обновление, Удаление).
2. Пакетная загрузка данных в индекс.
3. Совместный многоиндексный CRUD.
4. Создание и сохранение MVCC-индексов посредством сессии.
5. Комплексные сценарии, использующие существующую бизнес-логику.

Благодаря тому, что до создания собственной индексирующей системы был разработан и внедрен заказчикам рабочий прототип (основанный на СУБД MS SQL Server и Oracle), активно использовалась стратегия 5, применяемая к трансформации данных и к расчетам. Это позволило сравнивать ожидаемый и актуальный результаты тестовых сценариев.

Описанные выше методы тестирования показали хорошие результаты в определении наличия ошибок, но оказалось неожиданно сложно найти истинную причину ошибки. Индексная структура могла оказаться поврежденной, но повреждение при этом могло никак не проявляться достаточно долгое время. Поврежденная страница могла быть записана на диск, несколько раз прочитана, перезаписана и на момент срабатывания тестового сценария настоящую причину ошибки практически никогда не удавалось обнаружить. Поэтому, наряду с тестовыми сценариями, были разработаны дополнительные методы для того, чтобы определять ошибки раньше, отслеживать жизненный цикл структуры данных и предупреждать последствия ошибок во время промышленной эксплуатации путем обеспечения возможности восстановления данных.

2.1. Возможности среды Microsoft.NET

Перед началом проекта было принято решение разработать ядро базы данных поверх среды Microsoft. NET Framework. Данное решение явилось результатом многих предварительных условий и обсуждений (включая ранее написанный существенный объем кода промышленного прототипа), сравнения производительности, стоимости разработки и предпочтений заказчика.

Общеязыковая среда выполнения (*Common Language Runtime*, CLR) управляет памятью через сборку мусора [21]. Это существенно облегчает про-

граммирование, но не походит для объемов данных бóльших, чем оперативная память сервера. Именно с этим ограничением базы данных и борются долгое время. Поэтому для реализации низкоуровневых подпрограмм (таких как индексирование и курсоры) на языке C# применялся стиль программирования C. В связи с этим были приняты следующие решения.

1. Свести к минимуму выделение управляемой памяти. Управляемую память использовать только для временных переменных, но не для хранения долгоживущих.

2. Долгоживущие выделения памяти выполняются только через набор записей (через буфер, управляемо сбрасываемый на диск).

Неуправляемая память адресовалась через управляемые ссылки: *Page class*, *Record class*, *DataValue class*, *Index class* и т.д. Программная платформа .NET предоставляет стандартный способ управления жизненным циклом таких объектов через интерфейс *IDisposable*. Объекты, реализующие данный интерфейс, должны поддерживать возможность финализации через вызов метода *Dispose()*. Вызов стандартного финализатора — первый признак ошибки: объект с контролируемым жизненным циклом не был корректно освобожден. При промышленной эксплуатации это основание завести в эксплуатационный журнал необходимость выяснения причин и последующее расследование. При разработке это причина пометить сборку, как содержащую ошибки. Стандартный деструктор не должен быть выполнен даже в критической ситуации: отмена расчета, отказ диска, переполнение стека, недостаток оперативной памяти и другие системные ошибки. Чтобы система корректно функционировала, механизм выделения неуправляемой памяти должен самостоятельно ее освобождать.

2.2. Глобальная идентификация объектов

Каждая управляемая ссылка неуправляемого (в терминах .NET) ресурса (страница, запись, курсор, сессия) содержит два глобальных, автоинкрементных числовых идентификатора, перечисленных далее.

1. *Идентификатор* экземпляра управляемой ссылки. Различные *идентификаторы* для различных запросов к странице с тем же адресом.

2. *Идентификатор* сохраняемого объекта внутри контейнера. *Идентификатор* набора записей, *идентификатор* записи в наборе, *идентификатор* страницы (ее адрес).

Указанные идентификаторы легко читаемы (одно десятичное значение), показывают порядок создания объектов и могут быть получены детерминировано. После обнаружения ошибки, тестовый сценарий прокручивается до момента создания нарушившего индекс объекта. Затем состояние объектов просматривается программистом.

2.3. Проверка на непротиворечивость

Компоновка записи данных обычно является простой. Типы данных фиксированной ширины стандартизованы ABI (абстрактным бинарным интерфейсом). Обычно вызывает вопросы хранение данных переменной длины, кодировка строк и использование нуль-терминированных строк [22]. Но все эти вопросы достаточно подробно обсуждены в заслуживающих доверия научных источниках. Специальный формат строк можно использовать в инвертированных индексах, например, как в PostgreSQL GiN [23].

В противоположность страницам данных, индексные страницы сложно устроены и склонны к появлению в них ошибок. Целью R*-дерева является минимизация среднего числа обращений к диску *DA*, таким образом:

$$DA_{search} \sim O(R + \log N), \quad (1)$$

где N — размер данных; R — размер результата выполнения запроса. Процессорное время может быть минимизировано, когда коэффициент разветвления дерева $f \to \mathbf{e}$ [24]. Однако это ограничивает число размещаемых на странице записей максимум тремя, что неприемлемо, потому как почти все место на странице не будет использовано. Для оптимизации процессорного времени поиска требуется использовать Sqrt-декомпозицию или даже хранить небольшое R-дерево внутри страницы (узла R-дерева). Следует заметить, что указанные улучшения ведут к усложнению компоновки страниц.

Указанные выше причины требуют проверки страниц на непротиворечивость. Перечислим основные пункты данной проверки.

- Заголовок страницы валиден и соответствует типу страницы, ее позиции и статусу.
- Каждое смещение на странице ссылается на элемент внутри страницы. Смещения, добавляемые к размерам записей, лежат внутри границ страницы.
- Каждый адрес страницы ссылается на неудаляемую страницу.
- Ключи записи находятся внутри границ локального индекса (если доступна информация о родительской странице).
- Статистическая информация непротиворечива и корректна (свободные смещения и размеры памяти, все количественные показатели положительны и не равны нулю).

Эти проверки могут быть выполнены только после модификации страниц, так как они существенно замедляют системные операции. Но, как правило, в отладочных сборках системы полезно включать и выключать их в зависимости от выполняемой операции.

Данный подход позволяет программисту получить доступ к потенциально поврежденному объекту

со скоростью, близкой к промышленной, и переключить систему в режим исследования для определения ранних признаков ошибки.

2.4. Мониторинг модифицированных страниц

После прикрепления буферного кадра к страницам оперативной памяти операционной системы (ОС) становится возможным использовать флаг модификации памяти "dirty" ОС. Большинство современных ОС предоставляют функцию проверки факта изменения памяти после определенного события (вызова функции). После того как страница памяти занимает кадр, менеджер буфера помечает кадр как чистый. В отладочной сборке индексирующие алгоритмы информируют буфер, когда (до и после) происходит запись. Перед этим буфер записи проверяет, не был ли установлен флаг модификации памяти кем-то другим. Это означает, что ни одна подпрограмма не может разрушить содержимое кадра, переписав часть данных. После всех проверок буфер записи дополнительно проверяет, была ли действительно проведена запись, и сбрасывает флаг модификации.

Этот метод требует написания большого количества условно компилируемого кода, но, когда допустимо, это гарантирует, что повреждение данных на страницах определенной структуры произошло по вине именно этой структуры.

2.5. Блокировка записи незафиксированных страниц

Еще одна возможность изолировать память от перекрестной записи (которая может случиться вследствие похожести компоновки страниц и повреждения адресов страниц) это защита от чтения/записи страниц памяти ОС. Большинство алгоритмов индексирования спроектированы таким образом, что пользовательская сессия содержит только небольшую часть страниц, фиксированных в буфере памяти. Операционная система позволяет защитить незафиксированные кадры от чтения и записи.

Этот подход помогает обеспечить корректный порядок закрепления и открепления во время разработки алгоритмов.

2.6. Нехватка буфера

Уменьшить сходство близко выделенных страниц и тем самым облегчить отладку можно за счет ограничения размера буфера.

Буфер с ограниченным пространством, вынужденный разместить большое число страниц, будет постоянно обмениваться страницами между хранилищем и кадрами буфера. Несмотря на падение производительности при использовании данного метода, у него есть неоспоримое преимущество — он не требует особых усилий, достаточно просто настроить необходимый размер буфера.

2.7. Использование управляемой памяти для индекса

В ходе проекта было установлено, что реализация индекса в управляемой памяти (с помощью собираемых сборщиком мусора объектов) требует существенно меньше усилий, чем с помощью неуправляемой буферизованной памяти.

В целях тестирования были созданы модели некоторых индексов с помощью массивов и объектов .NET. Подобный подход не избавляет от ошибок разрешения конфликтов в многоверсионных индексах, но существенно более устойчив к ошибкам управления памятью. Сравнение состояния дерева в управляемой памяти с таким же в неуправляемой позволяет определить трудно заметные, потенциально приводящие к ошибкам изменения в узлах дерева.

В действительности, данный метод является одним из вариантов проверки на непротиворечивость с одним очень важным ограничением: тестовые данные должны быть существенно меньше доступного объема ОЗУ. В противном случае индекс в управляемой памяти начинает отображаться на жесткий диск и производительность падает на несколько порядков.

3. Детерминированное параллельное тестирование по сценарию

Многопоточные операции с индексом увеличивают энтропию в системе. Сложно обнаруживаемые ошибки практически никогда не обнаруживаются в одних и тех же объектах. Кроме того, они практически не поддаются проверке на непротиворечивость. Более того, по определению, ошибки конкурентного доступа не воспроизводятся при обычной (пошаговой) отладке программистом. Приведем несколько методов, которые могут уменьшить проблему непредсказуемости ошибок конкурентного доступа.

3.1. Запуск в режиме записи

Так как способом конкурентной работы был выбран MVCC, большинство многопоточных тестовых сценариев связаны с манипуляциями над многоверсионными деревьями. Получив определенный сценарий многопоточного тестирования и набор тестовых данных, становится возможным создать контрольную карту действий по манипулированию объектами дерева.

Это означает, что нужно записывать каждый вызов буфера, аргументы вызова и сохранять их во внешнее хранилище с последовательным доступом, размещаемое, желательно, в ОЗУ. Если пропускная способность исследуемых данных измеряется гигабайтами в секунду, то подобный поток изменений требуется отправлять по высокоскоростной сети на

другой сервер, использовав (по возможности) специализированную базу данных для хранения событий либо любой из вариантов кодирования повторов (RLE). Следует отметить, что задержки, вызванные необходимостью фиксировать события, снижают шансы воспроизвести ошибку.

Существует еще одна важная деталь для записи: функции, порождающие потоки, должны регистрировать их, получая для них уникальные идентификаторы. Это требуется для обеспечения возможности отладки сценариев взаимной блокировки.

3.2. Запуск в режиме отладки

В большинстве случаев обнаружения ошибки в результате запуска сценария эта ошибка воспроизводится при отладочном запуске.

После формирования журнала событий в режиме отладочного запуска нужно установить соответствие потоков из журнала потокам запущенного процесса. Если ожидается взаимодействие более 64 потоков, имеет смысл применять очередь с приоритетом (кучу) для хранения ожидаемых сопоставлений потоков. Число 64 получено опытным путем и не имеет теоретического обоснования. Во время отладки постоянно вносились изменения в конфигурацию для того, чтобы уменьшить задержку событий, влияющих на вероятность воспроизведения ошибки.

3.3. Управление отладочными конфигурациями

Использование описанных методов требует стабильной среды выполнения: сценарий, который выполнялся в архитектуре ×86, может получить взаимную блокировку в архитектуре ×64, некорректно установленный размер буфера также может привести к взаимной блокировке, большое число проверок снижает вероятность воспроизведения ошибок.

Вместе с тем тестовый сценарий должен быть максимально независимым от среды выполнения, что не всегда возможно. Вычисления могут вызывать функцию получения текущего времени *now()* и использовать результат вызова, как операнд или условие, тогда возможная взаимная блокировка становится непредсказуемой и практически не воспроизводимой.

Следующих функций также следует избегать: чтение системных статистических параметров, проверка безопасности, обращение к файловой системе и любые вызовы к графическому пользовательскому интерфейсу, объектам синхронизации ОС, а также любое межпроцессное взаимодействие.

4. Анализ

Большинство описанных методов решают две различные задачи: определить поведение системы как ошибку и отследить причину этого поведения. Эти методы должны применяться, чтобы накапливать описания и условия возникновения ошибок. Описание обязано быть детализировано для того чтобы надежно определить поведение как ошибочное.

Комбинации всех описанных методов позволили в срок внедрить систему в промышленную эксплуатацию. В настоящий момент в системе в среднем обнаруживается один факт повреждения индекса на каждом сервере раз в 18 месяцев. Каждый сервер обслуживает в среднем 200 пользователей с максимальным числом конкурирующих расчетов, равным 50. Следует отметить, что в режиме промышленной эксплуатации не было допущено ни одной потери или раскрытия данных по причине повреждения индекса. Реализованные механизмы избыточности позволили восстановить данные во всех ошибочных случаях. Все серверы оборудованы регистровой ECC-памятью, что уменьшает вероятность случайной ошибки памяти [25].

Несмотря на то что в системе находятся ошибки, представленные методы позволили системе функционировать в рамках соглашения об уровне предоставления услуги.

По оценкам авторов, использование приведенных методов отладки с самого начала проекта сэкономит примерно 10 человеко-лет. Общее время реализации описанного проекта оценивается примерно в 50 человеко-лет, включая исследования, разработку, обеспечение качества, внедрение и гарантийное сопровождение.

Заключение

Не существует методов, гарантирующих отсутствие ошибок. Даже формальные доказательства корректности только скрывают проблему и препятствуют отладке [26].

Описанный набор методов может быть обобщен и на другие предметные области для обеспечения качества, но большинство методов применимо только к древовидным индексам, отображенным на буфер памяти. Комбинация методов позволяет добиться существенно лучших результатов.

Исследование было проведено как подготовка к работам по улучшению производительности индекса PostgreSQL GiST [27].

Список литературы

1. **Codd E. F.** The relational model for database management: version 2. Boston, MA, USA: AddisonWesley Longman Publishing Co., Inc., 1990. 567 p.

2. **Codd E. F., Codd S. B., Salley C. T.** Providing olap (on-line analytical processing) to user-analysts: An it mandate. Codd and Date 32, 1993.

3. **Borodin A., Kiselev Y., Mirvoda S., Porshnev S.** On design of domain-specificquery language for the metallurgical industry// Beyond Databases, Architectures and Structures, Springer, 2015. P. 505—515.

4. **Aksyonov K., Antonova A.** Development of a Simulation Model of Cutting Slabs in a Continuous Casting Machine: Applied Mechanics and Materials// Proceedings of 2nd International

Conference on Applied Mechanics and Mechanical Automation (AMMA 2015). 2015. P. 224—228.

5. **Korotkov A.** Fast gist index build. URL: https://wiki.postgresql.org/images/0/07/Fast_GiST_index_build.pdf

6. **Бартунов О. С., Сигаев Т. Г.** Написание расширений для PostgreSQL с использованием GiST. URL: http://www.sai.msu.su/~megera/postgres/talks/gist_tutorial.html

7. **Mrozek D., Małysiak-Mrozek B., Mikołajczyk J., Kozielski S.** Database underpressure—testing performance of database systems using universal multi-agent platform // Man-Machine Interactions 3. Springer, 2014. P. 631—641.

8. **Smirnov K., Chernishev G., Fedotovsky P., Erokhin G., Cherednik K**. R-tree reevaluation effort: a report. Technical report. 2014.

9. **Чернов А. Ф.** Модификация индексов на основе R-деревьев для ускорения поиска // Информационные системы и технологии. 2011. № . 6 (68). С. 10—18.

10. **Zeller A.** Why programs fail: a guide to systematic debugging. Elsevier, 2009. 388 p.

11. **Shapiro E. Y.** Algorithmic program debugging. MIT press, 1983. 246 p.

12. **LeBlanc T. J., Mellor-Crummey J. M.** Debugging parallel programs with instantreplay // IEEE Transactions on Computers. 1987. Vol. 36. N. 4. P. 471—482.

13. **Beck K.** Test-driven development: by example. Addison-Wesley Professional. 2003.

14. **Garcia-Molina H., Ullman J. D., Widom J.** Database System Implementation. Prentice Hall Upper Saddle River, 2000. 653 p.

15. **Bayer R.** Symmetric binary b-trees: Data structure and maintenance algorithms // Acta informatica 1972. Vol. 1. N. 4. P. 290—306.

16. **Becker B., Gschwind S., Ohler T., Seeger B., Widmayer P.** An asymptotically optimal multiversion b-tree//The VLDB Journal—The International Journal on Very Large Data Bases. 1996. Vol. 5, N. 4. P. 264—275.

17. **Gaede V., Günther O.** Multidimensional access methods // ACM Computing Surveys(CSUR) 1998. Vol. 30, N. 2. P. 170—231.

18. **Beckmann N., Kriegel H. P., Schneider R., Seeger B.** The R*-tree: an efficient and robust access method for points and rectangles// Proc. of the 1990 ACM SIGMOD Internat. Conf. on Management of Data. 1990, P. 322—331.

19. **Korotkov A.** A new double sorting-based node splitting algorithm for R-tree // Proceedings of the 5th Spring/Summer Young Researchers' Colloquium on Software Engineering. SYRCoSE '11, 2011. P. 36—41.

20. **Manolopoulos Y., Nanopoulos A., Papadopoulos A. N., Theodoridis Y.** R-trees: Theory and Applications. Springer Science & Business Media, 2010. 194 p.

21. **Rahman M.** Clr memory model // C# Deconstructed. Springer, 2014. P. 61—86.

22. **Kamp P. H**. The most expensive one-byte mistake // Communications of the ACM. 2011. Vol. 54, N. 9. P. 42—44.

23. **Коротков А. Е., Панферов В. В.** Применение обобщенного дерева поиска для нечеткого поиска строки // Наука и образование: научное издание МГТУ им. Н. Э. Баумана. 2011. № 3. С. 11.

24. **Бородин А. М., Мирвода С. Г., Поршнев С. В.** Анализ многомерных данных высокой размерности: проблемы доступа к данным и возможный подход к их решению // Научно-технические ведомости Санкт-Петербургского государственного политехнического университета. Информатика. Телекоммуникации. Управление. 2013. № 6 (186). С. 59—66.

25. **Leray J.** Effects of atmospheric neutrons on devices, at sea level and in avionicsembedded systems// Microelectronics Reliability 2007. Vol. 47, Issue 9. P. 1827—1835.

26. **Bloch J.** Extra, extra-read all about it: Nearly all binary searches and mergesortsare broken. Official Google Research Blog Date. URL: https://research.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html

27. **Borodin A.** [proposal] improvement of gist page layout, URL: https://www.postgresql.org/message-id/CAJEAwVE0rrr+OBT-P0gDCtXbVDkBBG_WcXwCBK=GHo4fewu3Yg@mail.gmail.com

Database Index Debug Techniques: Application for Corporative Information System

A. M. Borodin, amborodin@acm.org, **S. G. Mirvoda,** sergey@mirvoda.com,
S. V. Porshnev, s.v.porshnev@urfu.ru, Ural Federal University, Yekaterinburg, 620002, Russian Federation

Corresponding author:

Borodin Andrey M., Associate Professor, Ural Federal University, Yekaterinburg, 620002, Russian Federation, e-mail: amborodin@acm.org

Received on June 23, 2016
Accepted on July 18, 2016

Index corruption may lead to serious problems ranging from the temporary system outage to the loss of sensitive data. In this article we discuss the techniques, that we found helpful in assuring the data index consistency during the development of specific indexing algorithms for a multidimensional BI system featuring both OLAP and OLTP aspects.

Use of the techniques described in this article from the very beginning of the project development helped to save sufficient resources during the development and debugging.

The main purpose of a database is to reliably store data. A database index provides a way of storing data in a specific format for the fast query execution, reducing the usage of crucial resources, such as the disk access, buffer memory, CPU cycles, and cache lines. An index is a data structure and a set of algorithms dealing with the memory hierarchy, concurrent data modification, resources locks, and complex computations on a big amount of data.

Keywords: *data, database, index, quality assurance, durability, debug*

For citation:

Borodin A. M., Mirvoda S. G., Porshnev S. V. Database Index Debug Techniques: Application for Corporative Information System, *Programmnaya Ingeneria,* 2016, vol. 7, no. 10, pp. 464—471.

DOI: 10.17587/prin.7.464-471

References

1. **Codd E. F.** *The relational model for database management: version 2*. Boston, MA, USA, AddisonWesley Longman Publishing Co., Inc., 1990. 567 p.

2. **Codd E. F., Codd S. B., Salley C. T**. Providing olap (on-line analytical processing) to user-analysts: An it mandate. Codd and Date 32, 1993.

3. **Borodin A., Kiselev Y., Mirvoda S., Porshnev S.** On design of domain-specificquery language for the metallurgical industry, *Beyond Databases, Architectures and Structures*, 2015, pp. 505—515.

4. **Aksyonov K., Antonova A.** Development of a Simulation Model of Cutting Slabs in a Continuous Casting Machine: Applied Mechanics and Materials, *Proceedings of 2nd International Conference on Applied Mechanics and Mechanical Automation* (AMMA 2015). April 19—20, 2015, pp. 224—228. Vol. 775.

5. **Korotkov A.** Fast gist index build, available at: https://wiki.postgresql.org/images/0/07/Fast_GiST_index_build.pdf

6. **Bartunov O. S., Sigaev T. G.** Writing GiST extensions for PostgreSQL, available at: http://www.sai.msu.su/~megera/postgres/talks/gist_tutorial.html

7. **Mrozek D., Małysiak-Mrozek B., Mikołajczyk J., Kozielski S.** Database underpressure—testing performance of database systems using universal multi-agent platform, *Man-Machine Interactions*, Springer, 2014, pp. 631—641.

8. **Smirnov K., Chernishev G., Fedotovsky P., Erokhin G., Cherednik K.** R-tree reevaluation effort: a report. Technical report, 2014.

9. **Chernov A. F.** Modifikatsiya indeksov na osnove R-derev''ev dlya uskoreniya poiska (Modification of index access methods, based on R-trees), *Informatsionnye sistemy i tekhnologii*, 2011, no. 6 (68), pp. 10—18 (in Russian).

10. **Zeller A.** *Why programs fail: a guide to systematic debugging*, Elsevier, 2009, 388 p.

11. **Shapiro E. Y.** *Algorithmic program debugging*, MIT press, 1983, 246 p.

12. **LeBlanc T. J., Mellor-Crummey J. M.** Debugging parallel programs with instantreplay, *IEEE Transactions on Computers,* 1987, vol. 36, no. 4, pp. 471—482.

13. **Beck K.** *Test-driven development: by example*, Addison-Wesley Professional, 2003.

14. **Garcia-Molina H., Ullman J. D., Widom J.** *Database System Implementation*, Prentice Hall Upper Saddle River, 2000, 653 p.

15. **Bayer R.** Symmetric binary b-trees: Data structure and maintenance algorithms, *Acta informatica,* 1972, vol. 1, no. 4, pp. 290—306.

16. **Becker B., Gschwind S., Ohler T., Seeger B., Widmayer P.** An asymptoticallyoptimal multiversion b-tree, *The VLDB Journal—The International Journal on Very Large Data Bases*, 1996, vol. 5, no. 4, pp. 264—275.

17. **Gaede V., Gunther O.** Multidimensional access methods, *ACM Computing Surveys(CSUR)*, 1998, vol. 30, no. 2, pp. 170—231.

18. **Beckmann N., Kriegel H. P., Schneider R., Seeger B.** The R*-tree: an efficient and robust access method for points and rectangles, *Proc. of the 1990 ACM SIGMOD Internat. Conf. on Management of Data* 1990, pp. 322—331.

19. **Korotkov A.** A new double sorting-based node splitting algorithm for R-tree, *Proceedings of the 5th Spring/Summer Young Researchers' Colloquium on Software Engineering. SYRCoSE '11*, 2011, pp. 36—41.

20. **Manolopoulos Y., Nanopoulos A., Papadopoulos A. N., Theodoridis Y.** *R-trees: Theory and Applications,* Springer Science & Business Media, 2010, 194 p.

21. **Rahman M.** Clr memory model, *C# Deconstructed*, Springer, 2014, pp. 61—86.

22. **Kamp P. H**. The most expensive one-byte mistake, *Communications of the ACM,* 2011, vol. 54, no. 9, pp. 42—44.

23. **Korotkov A. E., Panferov V. V.** Primenenie obobshhennogo dereva poiska dlja nechetkogo poiska stroki (Using GiST for fuzzy string search), *Nauka i obrazovanie,* 2011, no. 3, pp. 11. (in Russian).

24. **Borodin A. M., Mirvoda S. G., Porshnev S. V.** Analiz mnogomernyh dannyh vysokoj razmernosti: problemy dostupa k dannym i vozmozhnyj podhod k ih resheniju (High dimensional data analisys: data access problems and possible solutions), *Nauchno-tehnicheskie vedomosti Sankt-Peterburgskogo gosudarstvennogo politehnicheskogo universiteta. Informatika. Telekommunikacii. Upravlenie,* 2013, no. 6, pp. 59—66 (in Russian).

25. **Leray J.** Effects of atmospheric neutrons on devices, at sea level and in avionicsembedded systems, *Microelectronics Reliability,* 2007, vol. 47, issue 9, pp. 1827—1835.

26. **Bloch J.** Extra, extra-read all about it: Nearly all binary searches and mergesortsare broken. Official Google Research Blog Date, available at: https://research.googleblog.com/2006/06/extra-extra-read-all-about-it-nearly.html

27. **Borodin A**. [proposal] improvement of gist page layout, https://www.postgresql.org/message-id/CAJEAwVE0rrr+OBT-P0gDCtXbVDkBBG_WcXwCBK=GHo4fewu3Yg@mail.gmail.com

УДК 001.89, 004.415.2 DOI: 10.17587/prin.7.472-480

В. А. Васенин, д-р. физ.-мат. наук., проф., зав. лаб., e-mail: vasenin@msu.ru,
А. А. Зензинов, мл. науч. сотр., e-mail: andrey.zenzinov@gmail.com,
Научно-исследовательский институт механики МГУ имени М. В. Ломоносова,
К. В. Лунев, аспирант, e-mail: kirilllunev@gmail.com, Механико-математический факультет МГУ имени М. В. Ломоносова, Москва

Использование наукометрических информационно-аналитических систем для автоматизации проведения конкурсных процедур на примере информационно-аналитической системы "ИСТИНА"

Представлен подход к автоматизации проведения конкурсных процедур в организациях науки и образования на основе механизмов информационно-аналитических систем. Такие системы активно используют в вузах и научно-исследовательских центрах для подготовки принятия управленческих решений, направленных на повышение эффективности их деятельности на всех уровнях — от отдельного ученого, педагога до организации в целом. Продемонстрирован опыт, полученный при реализации этого подхода в действующей в МГУ имени М. В. Ломоносова информационно-аналитической системе "ИСТИНА".

Ключевые слова: *CRIS-системы, наукометрия, конкурсы, автоматизация*

Введение

Одной из основополагающих мер, способствующих повышению эффективности работы субъектов научно-технической и образовательной деятельности, является проведение различного рода конкурсов. К их числу относятся конкурсы:

— определяющие лучших по итогам за определенный период времени работников или коллективы (научные группы, кафедры/лаборатории и т. п.) в той или иной области деятельности (в том числе — премии, надбавки к заработной плате);

— на право выполнения проектов (получения грантов), требования и условия выполнения которых определяются конкурсной документацией;

— на занятие вакантных должностей, которые определяются штатным расписанием организации.

Организация подобных конкурсов сопряжена с проведением следующих процедур: определение круга потенциальных участников; информирование потенциальных участников о проведении конкурса; сбор конкурсных заявок; поиск потенциальных экспертов; распределение конкурсных заявок между экспертами; сбор экспертных заключений, их анализ и определение победителей. Для автоматизации проведения описанных выше конкурсных процедур необходима реализация специализированной аппаратно-программной платформы. В качестве такой конкурсной платформы могут использоваться специальные информационно-аналитические системы (ИАС). Подобные системы имеют многие научные фонды. Положительно зарекомендовавшими себя примерами таких систем являются КИАС РФФИ (Конкурсная информационно-аналитическая система Российского Фонда Фундаментальных Исследований) [1], ИАС РНФ (Информационно-аналитическая система Российского Научного Фонда) [2], ИС РГНФ (Информационная система Российского Гуманитарного Научного Фонда) [3]. Конкурсы по Федеральным целевым программам проводятся с использованием Портала регистрации заявок на участие в конкурсе [4]. Система КИАС РФФИ позволяет автоматизировать значительную часть описанных выше конкурсных процедур, а именно — сбор заявок и последующую их экспертизу. В этой системе предусмотрена возможность хранения научной информации о заявителе или коллективе заявителей (список публикаций с его или их участием и другие научные результаты) для дальнейшего использования при подаче заявок на проекты

со схожим тематическим направлением. Основным недостатком такого подхода является то обстоятельство, что система не предполагает регулярное обновление этой научной информации, поскольку ее ввод в систему не является обязательным шагом при подаче заявки. Следует отметить также и тот факт, что КИАС РФФИ, ИАС РНФ, ИС РГНФ разрабатывались целевым назначением для автоматизированного проведения конкурсных процедур соответствующих фондов, соответственно, эти системы не допускают проведения иных конкурсов.

Существуют специализированные платформы, предоставляющие услуги по автоматизации конкурсных процедур. Такие платформы ориентированы в основном на конкурсы в социальных сетях, на развлекательных порталах. Среди зарубежных решений обычно выделяют сервисы, предоставляемые системой WizeHive Select[5], которая ориентирована на проведение конкурсов в научной среде: гранты, награды, конкурсы на участие в стажировках, а также ряд других. Среди клиентов сервисов WizeHive можно выделить Стэнфордский Университет и Университет Южной Калифорнии.

Примером отечественной платформы для автоматизации проведения конкурсов является konkurs-online.ru [6], клиентами которого являются Фонд "Сколково", Министерство образования и науки РФ, Петрозаводский государственный университет и др. Перечисленные сервисы автоматизируют все основные этапы конкурса и обладают гибкими механизмами настройки. К недостаткам сторонних конкурсных платформ можно отнести то обстоятельство, что их использование не является бесплатным, а обработка и хранение данных происходит на стороне сервиса.

На настоящее время в научных организациях активно начинают использовать различного рода ИАС, среди которых можно выделить информационные системы для хранения и управления данными о научных исследованиях, проводимых в тех или иных организациях (университетах, научных центрах и т. п.). Такие системы принято называть CRIS-системами (*Current Research Information System*). В МГУ имени М. В. Ломоносова функциональные возможности таких систем реализуются с использованием программных механизмов ИАС "ИСТИНА" (Информационная Система Тематического Исследования НАукометрических данных) [7], в СПбГУ — ИАС сопровождения научно-исследовательской деятельности СПбГУ (ИАС НИД СПбГУ) [8]. Использование программных механизмов подобных систем в качестве конкурсной платформы позволяет не только автоматизировать многие конкурсные процедуры, но и способствует решению ряда других, сопутствующих конкурсам задач, например:

- предоставление открытого сервиса для проведения тех или иных видов конкурсов, позволяющего контролировать проведение конкурсных процедур на всех этапах проведения конкурса;
- стимулирование наполнения ИАС данными о научной деятельности конкурсантов;
- получение и агрегация информации о состоянии мировых и отечественных исследований в различных областях научно-технической деятельности;
- поиск в ИАС экспертов по заданным тематическим направлениям.

С учетом изложенных соображений представляется целесообразным автоматизировать проведение конкурсных процедур с использованием возможностей ИАС типа CRIS, которые далее будем именовать Системой.

Настоящая работа посвящена описанию подхода к решению данной задачи, а также его реализации в Системе, которая называется ИАС "ИСТИНА".

1. Автоматизация конкурсных процедур

Научные конкурсы можно классифицировать по нескольким позициям: по предмету конкурса, а также по тому, кто является участником конкурса — коллектив исследователей или отдельная персона — ученый или педагог, которых далее будем именовать персоналиями. Сущности, объекты, составляющие предмет конкурса, можно разделить на две группы, которые:

— отображают результаты, полученные до момента подачи заявки и подлежат оценке в ходе экспертизы;

— характеризуют проект, предлагаемый к исследованию и разработке или реализации в каком-то виде (макет, прототип, конечное изделие).

Конкурсы, соответствующие первой группе, будем далее для краткости изложения называть конкурсами результатов, а конкурсы, соответствующие второй группе, — конкурсами проектов.

Участником конкурса, как уже было отмечено выше, может быть как отдельный персоналий, так и коллектив исследователей. Учитывая классификацию по предмету, получим четыре вида конкурсов: конкурс результатов, полученных отдельными персоналиями; конкурс проектов, предложенных к выполнению отдельными персоналиями; конкурс результатов, полученных коллективами; конкурс проектов, предложенных к выполнению коллективами.

1.1. Подача заявки

В том случае, если в конкурсных процедурах используется ИАС, для конкурсов результатов в такой Системе может быть вся необходимая информация о результатах, т. е. конкурс проводится среди пользователей Системы, которые ввели в нее информацию о своих результатах. Если вся информация уже введена в Систему, то пользователю достаточно лишь нажать на кнопку, чтобы подать заявку на конкурс. Подача заявки на коллективный конкурс результатов при этом отличается незначительно. Она вполне может осуществляться одним представителем коллектива, который согласно положению о проведении конкурса имеет на это соответствующие полномочия.

Подача заявок на конкурс проектов предполагает проведение некоторого исследования области знаний: часто в заявках требуется оценить состояние мировых

исследований в данной области, актуальность проекта и его научную новизну. Такого рода информация не вычисляется автоматически и не хранится в Системе в явном виде, в отличие от результатов деятельности, а вводится на этапе подачи заявки. Подобная информация должна учитывать специфику отдельного конкурса и быть актуальной на момент подачи заявки. Следует заметить, что такая информация представляет особый интерес в наукометрии, поскольку отражает неколичественные (качественные) особенности научных областей. Извлечение этих особенностей из текстовых данных, однако, является нетривиальной задачей.

1.2. Модель

Основными действующими лицами в конкурсных процедурах являются: организатор конкурса; конкурсная комиссия; экспертная комиссия; заявители. Процесс проведения конкурса можно разделить на несколько этапов: подготовка конкурса (определение темы, числа победителей, поиск источников финансирования); объявление конкурса; сбор заявок; экспертиза; определение победителей; объявление результатов; награждение победителей. Все эти этапы, за исключением части этапа подготовки конкурса и награждения, как правило, можно автоматизировать с использованием ИАС. Отдельное внимание следует уделить этапу определения победителей. Этот этап обычно представляет собой заседание конкурсной комиссии, которая после обсуждения предварительно подготовленных материалов и голосования принимает общее решение и определяет победителей. Можно автоматизировать этот этап, если определять победителей на основе заранее подготовленной оценки эффективности их деятельности — рейтинга. Таким рейтингом может быть среднее значение оценок, выставленных заявке экспертами, или иной показатель, рассчитываемый по заданной формуле. Принимая во внимание то обстоятельство, что обсуждение конкурсной комиссии является неотъемлемой частью процедуры определения победителей, эти рейтинговые оценки упрощают и ускоряют принятие решений, делают ее более обоснованной, но не являются определяющими.

На рис. 1 изображен бизнес-процесс проведения конкурса в ИАС "ИСТИНА".

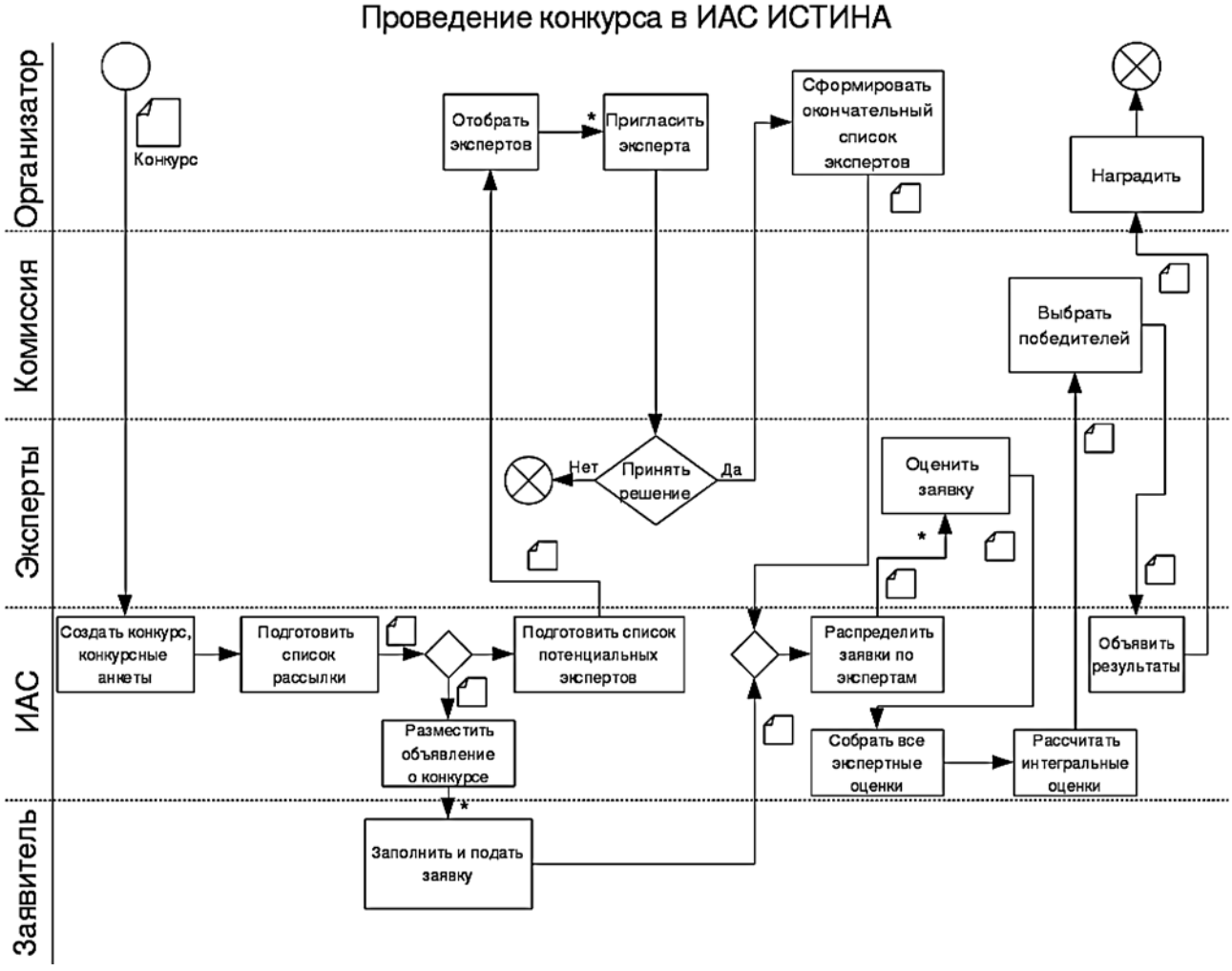

Рис. 1. Бизнес-процесс проведения конкурса в ИАС "ИСТИНА"

Рассмотрим подробнее реализацию основных этапов конкурса на этом примере.

На этапе подготовки конкурса организатор предоставляет ИАС описание условий конкурса, на основе которых в Системе создаются необходимые объекты: веб-страницы с информацией о конкурсе; формы для подачи заявок; формы для экспертных заключений. Процесс создания конкурсов также можно автоматизировать с помощью реализации механизмов импорта описаний конкурсов из заранее подготовленного файла в XML-формате, содержащего описание условий проведения конкурса. Такой файл может быть составлен организаторами конкурса самостоятельно, без участия разработчиков Системы.

К подготовке конкурса можно также отнести подготовку списка потенциальных участников (списка рассылки) и списка потенциальных экспертов. Список рассылки зависит от тематики конкурса и его условий. С одной стороны, объявления о конкурсах по естественно-научным направлениям не следует рассылать сотрудникам, работающим на гуманитарных направлениях, и наоборот. С другой стороны, конкурсы с общей тематикой могут быть интересны всем, и рассылку можно делать по всем пользователям Системы.

Подача заявки на конкурс доступна всем зарегистрированным пользователям. После того как заявки на конкурс собраны, они назначаются экспертам на рецензирование. Процессы подготовки списка потенциальных экспертов и распределения заявок между экспертами описаны в подразд. 2.5.

После того как все эксперты завершат работу по оцениванию конкурсных заявок, на заседании конкурсной комиссии определяют победителя конкурса.

1.3. Разграничение доступа

В конкурсных процедурах предусмотрен обмен конфиденциальной информацией. Такой информацией являются: заполненные конкурсные заявки; информация о том, каким экспертам какие заявки назначены; экспертные рецензии. В связи с этим Системе предъявляются требования по разграничению доступа к такого сорта конфиденциальным данным. Основные правила подобного разграничения:

- заявитель имеет право заполнять и редактировать свою заявку до ее подписания;
- эксперт имеет право доступа на просмотр только тех заявок, которые ему назначены;
- организатор имеет право просматривать общий список заявок;
- организатор имеет право просматривать экспертные рецензии.

Более подробная информация об используемой в ИАС "ИСТИНА" модели разграничения доступа к данным и ее программной реализации представлена в работе [7].

2. Подсистема "Конкурсы" в ИАС "ИСТИНА"

Представленная выше модель автоматизации конкурсных процедур реализована в виде программной подсистемы "Конкурсы" в ИАС "ИСТИНА" [7], которая создана и развивается в МГУ имени М. В. Ломоносова. Опишем далее основные функциональные возможности такой подсистемы.

2.1. Описание конкурса

В личном кабинете пользователя ИАС "ИСТИНА" доступен список предстоящих, а также список недавно завершенных конкурсов. Конкурсы могут иметь свою структуру, а именно — у конкурса верхнего уровня может быть несколько конкурсов нижнего уровня, называемых номинациями. Примером такой структуры может быть конкурс научных работников в двух номинациях: для сотрудников не старше 35 лет и для сотрудников старше 35 лет.

Страница описания конкурса содержит информацию о названии, о датах проведения конкурса, номинациях. Каждая номинация имеет свое описание и ссылки на подачу заявки. Пользователь может подавать заявку на любое число номинаций. Номинации могут отличаться друг от друга по дате подачи заявок.

Для тех пользователей, которые получили привелегии эксперта по данному конкурсу, на странице с описанием этого конкурса отображаются назначенные на рецензирование заявки, разделенные на две группы: обработанные (рецензия подана) и необработанные.

К описанию конкурса можно прикреплять файлы, например, конкурсную документацию, образцы заполнения заявок, отсканированные версии приказов, результаты конкурса.

2.2. Заявка на конкурс

Каждый отдельный конкурс имеет свою форму подачи заявок, характеризующуюся набором полей. Разработка отдельных веб-страниц в ИАС для каждой такой формы представляется излишней, а изменения состава полей, порядка их следования в форме, а также текстового описания требуют внесения изменений в код Системы. Разработанные универсальные механизмы подачи конкурсных заявок позволяют устранить необходимость вмешательства в программный код или, как минимум, снизить степень такого вмешательства.

Форма заявки представляет собой набор полей, распределенных по нескольким разделам. Подобное деление на разделы призвано приблизить структуру заявки к тем, которые используются в конкурсах РНФ, РФФИ и других, которые пользуются популярностью у ученых МГУ. На рис. 2 изображен интерфейс для заполнения двух разделов формы заявки.

Каждое поле заявки характеризуется следующими параметрами: имя поля; принадлежность поля разделу (ссылка на объект типа "раздел"); порядок

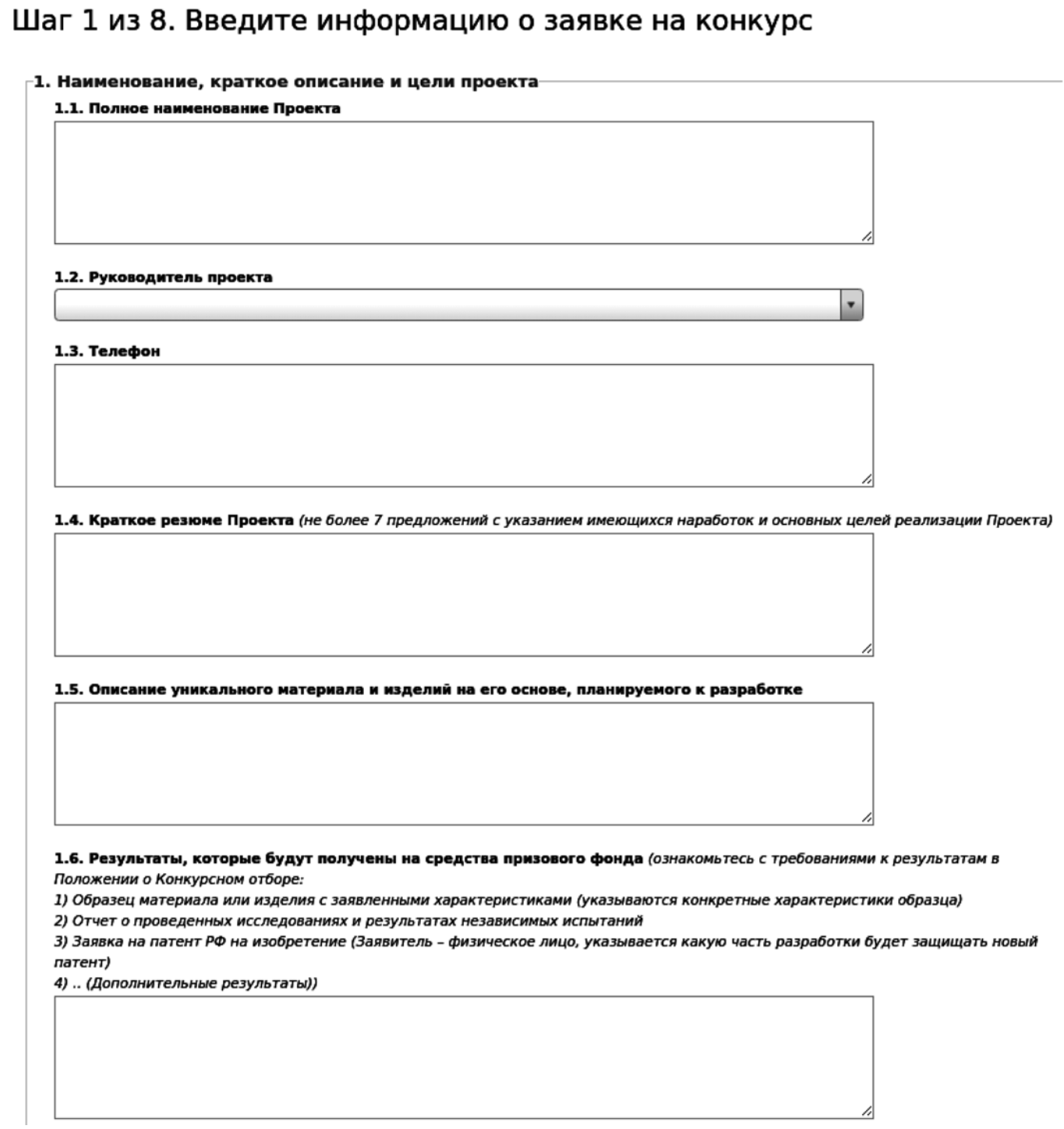

Рис. 2. Фрагмент интерфейса формы подачи заявки на конкурс

поля в разделе; тип объекта; является ли поле обязательным для заполнения. Параметр "тип объекта" описывает тип информации, который доступен для ввода в соответствующее поле заявки. Значением этого параметра может быть любой тип объектов, используемых в ИАС. Для указания ссылки на определенного персоналия таким типом объекта является "сотрудник"; для списка публикаций — "публикация"; для документов, файлов — "прикрепляемый файл"; для вопросов с ответами вида "да/нет" — "булево значение поля"; для остальных полей — "текстовое значение поля". Тип объекта поля влияет на то, как оно будет отображаться при подаче и просмотре заявки. Поле с типом "сотрудник" при создании заявки отображается в виде поисковой строки. При вводе первых букв фамилии осуществляется поиск по объектам ИАС, имеющим тип "сотрудник", полученные результаты выводятся в виде списка. Заполнение поля осуществляется выбором элементов из списка результатов поиска. При просмотре заявки, кроме ФИО сотрудника можно отображать и другие его данные, хранящиеся в ИАС, например, ученую степень, звание, место работы, e-mail.

В качестве другого примера можно привести ввод публикаций в поле заявки с типом объекта "публикация". Заполнение этого поля также осуществляется через поисковую строку. Поиск ведется по фамилиям авторов и названию статьи, а также поддерживает множественный ввод публикаций. При просмотре заявки это поле отображается в виде списка литературы со ссылками, оформленными по ГОСТ.

2.3. Экспертиза заявок

Форма экспертной рецензии во многих конкурсных системах также представляет собой набор текстовых полей, вследствие чего для реализации экспертного интерфейса используют механизмы, схожие с описанными в подразд. 2.2. Конкурсная документация определяет набор критериев, по которым следует оценивать заявки. Для каждого критерия в интерфейсе предусмотрено текстовое поле для ввода комментария и поле с выбором оценки в баллах (рис. 3), максимальное число которых также определяется в конкурсной документации.

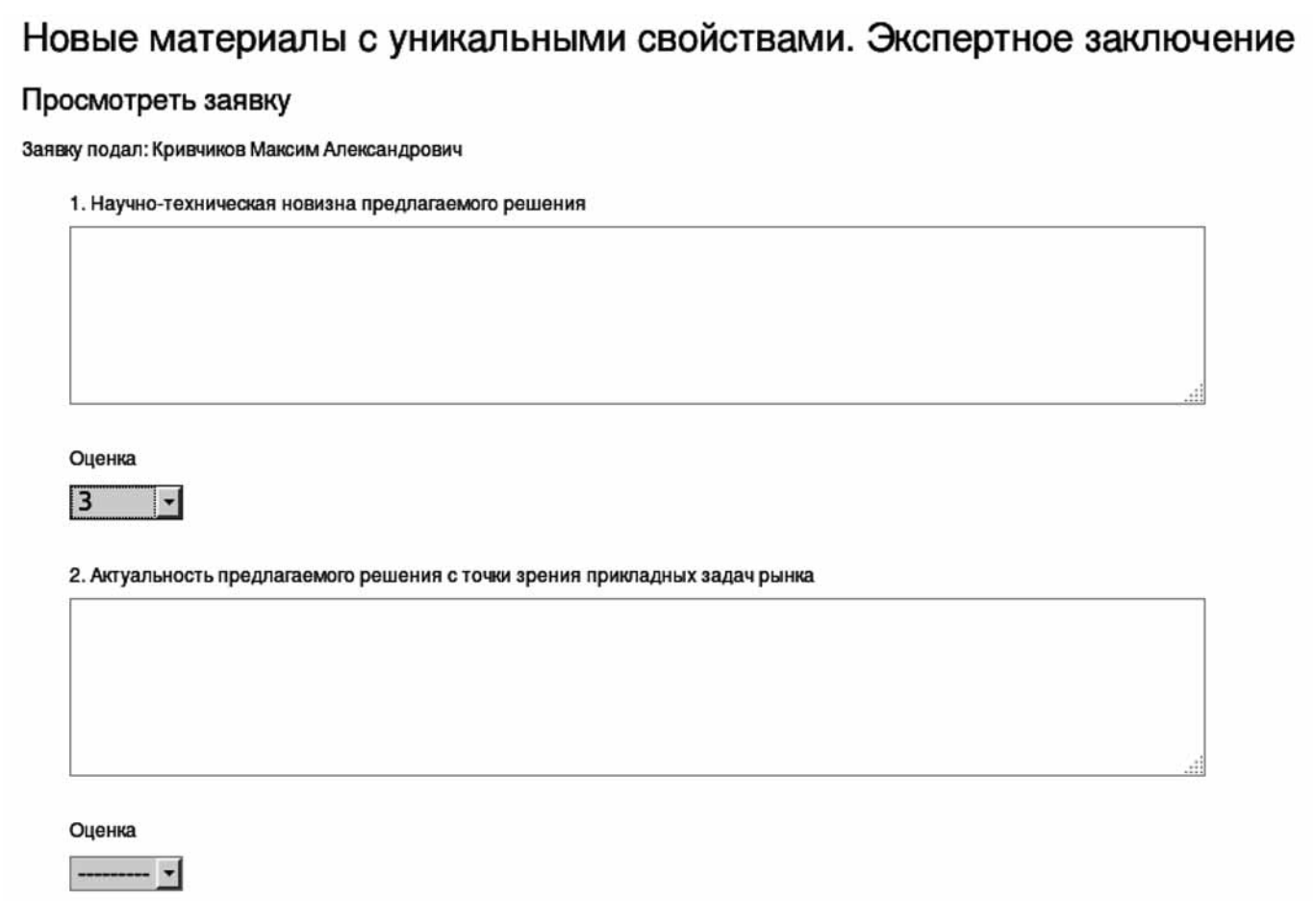

Рис. 3. Фрагмент интерфейса для составления экспертных заключений

2.4. Интерфейс для организаторов и конкурсной комиссии

Для конкурсной комиссии и организаторов реализованы страницы для просмотра поданных заявок. Члены конкурсной комиссии имеют право менять статусы заявок, а именно: возвращать заявку на доработку; отклонять заявку; утверждать заявку. Рис. 4 иллюстрирует пример интерфейса просмотра результатов конкурса. Поле "статус заявки" является редактируемым. При изменении статуса заявки заявителю на его электронный адрес отправляется письмо с уведомлением об изменении статуса.

2.5. Поиск экспертов по ключевым словам

Алгоритм поиска экспертов состоит из двух основных этапов:

— предварительной обработки данных, на котором вся доступная информация об эксперте собирается в одно множество ключевых слов с весами;

— поиска релевантных персон по запросу, состоящему из одного или нескольких ключевых слов.

Далее кратко опишем алгоритмы, которые используются на каждом из этих этапов работы алгоритма.

Каждый эксперт в Системе ассоциируется с набором ключевых слов. Для более полного описания такого набора для каждого эксперта используются различные способы извлечения, сбора и агрегации ключевых слов. Основным их источником являются слова, отмеченные в качестве ключевых в профиле эксперта в Системе. Помимо этого, наборы таких ключевых слов дополняются словами из других информационных источников, отражающих результаты деятельности научных коллективов, а именно: научные статьи; сведения о результатах выполнения проектов, в которых был задействован эксперт; другие источники. В этих целях также применяют способы выделения ключевых понятий из текстов аннотаций к публикациям [9]. Система позволяет встраивать новые методы сбора ключевых слов, расширяя тем самым знания об эксперте. Каждому слову назначается свой вес, который показывает степень важности слова для эксперта.

Несмотря на обилие методов, позволяющих собирать ключевые слова из данных об эксперте, зачастую не хватает сведений для выполнения качественного поиска. При составлении запроса

Заявки на конкурс по факультетам

Show by 10 items Search:

Ссылка на профиль	Заявка	Подразделение	Статус заявки	Дата подачи	Рейтинг	Дата расчёта рейтинга
[redacted]	[redacted]	Научно-исследовательский институт механики	Подписана	20 июня 2016 г. 0:00:00	697	11 июля 2016 г. 0:00:00
[redacted]	[redacted]	Научно-исследовательский институт механики	Подписана	10 июня 2016 г. 0:00:00	567	11 июля 2016 г. 0:00:00
[redacted]	[redacted]	Научно-исследовательский институт механики	Подписана	8 июня 2016 г. 0:00:00	364	11 июля 2016 г. 0:00:00
[redacted]	[redacted]	Научно-исследовательский институт механики	Подписана	19 июня 2016 г. 0:00:00	358	11 июля 2016 г. 0:00:00
[redacted]	[redacted]	Научно-исследовательский институт механики	Подписана	10 июня 2016 г. 0:00:00	289	11 июля 2016 г. 0:00:00
[redacted]	[redacted]	Научно-исследовательский институт механики	Подписана	19 июня 2016 г. 0:00:00	286	11 июля 2016 г. 0:00:00
[redacted]	[redacted]	Научно-исследовательский институт механики	Подписана	15 июня 2016 г. 0:00:00	257	11 июля 2016 г. 0:00:00
[redacted]	[redacted]	Научно-исследовательский институт механики	Подписана	16 июня 2016 г. 0:00:00	236	11 июля 2016 г. 0:00:00
[redacted]	[redacted]	Научно-исследовательский институт механики	Подписана	20 июня 2016 г. 0:00:00	211	11 июля 2016 г. 0:00:00
[redacted]	[redacted]	Научно-исследовательский институт механики	Подписана	7 июня 2016 г. 0:00:00	203	11 июля 2016 г. 0:00:00

Items 1 to 10 shown. Total: 23 First Previous 1 2 3 Next Last

Рис. 4. Интерфейс просмотра результатов конкурса с выборкой по подразделению

нередко возникают ситуации, когда одно из слов не представлено в Системе и его необходимо заменить на другое слово, информация о котором в Системе имеется. Для преодоления этих трудностей разработан алгоритм расширения ключевого слова одним или несколькими ключевыми словами. Согласно этому алгоритму собирается корпус наборов ключевых слов, достигающий размера в несколько сотен тысяч уникальных слов. После этого, с использованием методов машинного обучения с учителем, а также теоретико-графовых алгоритмов и алгоритмов обработки естественного языка, проводится "обучение" формулы, которая для пары слов показывает их уровень близости. Для выбранного ключевого слова близкими могут быть слова, являющиеся синонимами, орфографическими вариантами, контекстно-близкими, исправленными опечатками, переводами на другие языки или аббревиатурами. Далее для каждого слова из Системы выбирают наиболее похожие к нему слова и составляют словарь расширений. Каждое ключевое слово, ассоциированное с экспертом, расширяется словами из составленного словаря. Таким образом, информация об эксперте обогащается новыми семантически близкими словами. Вес слову-расширению выставляется пропорционально мере его близости к исходному слову. После этого по списку экспертов и по их ключевым словам строится инвертированный индекс, который позволяет по слову быстро определить экспертов, наиболее релевантных этому слову. На этом этап предварительной обработки данных завершается.

Дальнейший поиск происходит по следующему алгоритму. Пользователь Системы формулирует запрос в терминах одного или нескольких ключевых слов, отделенных друг от друга символом-разделителем. После этого Система расширяет каждое из слов запроса с помощью словаря расширений. Такой подход позволяет найти замену для многих из слов, которые в Системе не представлены. С использованием построенного на первом этапе инвертированного индекса определяют кандидатов в эксперты. Последним шагом является вычисление степени близости набора слов запроса до каждого кандидата в эксперты. Для этого подсчитывают косинусное расстояние между векторами слов эксперта и запроса. Результаты ранжируют по значениям данной метрики, после чего пользователю возвращают список наиболее подходящих под сделанный запрос экспертов.

Заключение

Авторами разработаны универсальные механизмы автоматизации проведения конкурсных процедур. Реализация этих механизмов на платформе информационно-аналитических систем анализа наукометрических данных позволяет значительно упростить и ускорить проведение конкурсов.

За период 2014—2016 гг. в Московском государственном университете имени Ломоносова проведен ряд конкурсов на платформе ИАС "ИСТИНА". В 2015 г. в сотрудничестве с некоммерческой организацией "Иннопрактика" проведен конкурс прикладных проектов по тематике "Новые материалы с уникальными свойствами", формы заявок на который предполагали проведение тщательного анализа научной области и актуальности предлагаемых результатов. Из 55 поданных заявок было отобрано 10 победителей, исходя из результатов экспертизы, которая также проводилась через интерфейс ИАС "ИСТИНА". На этой платформе также были проведены конкурсы, направленные на оценку результатов научной деятельности отдельных ученых, такие как: Конкурс молодых ученых (2015 г.); Конкурс работ, способствующих решению задач Программы развития Московского университета (2014, 2016 гг.). На последний конкурс в 2016 г. подано более 3000 заявок на номинации "достижения в научно-исследовательской деятельности" и "достижения лекторов межфакультетских учебных курсов, курсов междисциплинарной тематики, онлайн курсов, реализуемых на платформе МГУ им. Ломоносова "Университет без границ". В ходе проведения этих конкурсов разработанные механизмы доказали свою эффективность.

Дальнейшее развитие механизмов автоматизации конкурсных процедур предполагает улучшение алгоритмов поиска экспертов по ключевым словам, а также улучшение механизмов заполнения заявок, что может включать в себя частичное заполнение заявок данными, полученными из анализа заявок на предыдущие конкурсы.

Разработанные механизмы могут также применяться и при проведении других конкурсов, проходящих в вузах и научных организациях. В качестве примера можно привести конкурсы на замещение вакантных должностей научных работников и профессорско-преподавательского состава.

Список литературы

1. **КИАС** РФФИ. URL: https://kias.rfbr.ru
2. **ИАС** РНФ. URL: http://grant.rscf.ru
3. **ИС** РГНФ. URL: http://grant.rfh.ru
4. **Портал** регистрации заявок на участие в конкурсах. Дирекция НТП. URL: http://konkurs2014.fcpir.ru
5. **Grants** management system — WizeHive Select. WizeHive. URL: http://www.wizehive.com/grant-management-system/
6. **Конкурс-**онлайн. Сервис для автоматизации конкурсов. URL: http://www.konkurs-online.ru
7. **Садовничий В. А., Афонин С. А., Бахтин А.В и др.** Интеллектуальная система тематического исследования научно-технической информации (ИСТИНА). М.: Изд-во МГУ, 2014. 262 с.
8. **Информационно**-аналитическая система сопровождения научно-исследовательской деятельности СПбГУ. URL: https://ias.spbu.ru
9. **Афонин С. А., Лунев К. В.** Выявление тематических направлений в коллекции наборов ключевых слов // Программная инженерия. 2015. № 2. С. 29—39.

The Usage of CRIS-systems for the Contest Procedures Automation in Terms of the ISTINA Information System

V. A. Vasenin, vasenin@msu.ru, **A. A. Zenzinov,** andrey.zenzinov@gmail.com,
Institute of Mechanics, Lomonosov Moscow State University, Moscow, 119192, Russian Federation,
K. V. Lunev, kirilllunev@gmail.com, Mechanics and Mathematics Department,
Lomonosov Moscow State University, Moscow, 119991, Russian Federation

Corresponding author:

Zenzinov Andrey A., Junior Research Fellow**,** Institute of Mechanics, Lomonosov Moscow State University, Moscow, 119192, Russian Federation,
e-mail: andrey.zenzinov@gmail.com

Received on July 04, 2016
Accepted on July 29, 2016

CRIS (Current-Research Information Systems) systems are designed for collection, storing and processing information about the scientific and educational activities at multiple levels: from several researchers to the entire organization (university, institute). Information analysis in such systems is usually used for preparation of management decisions aimed at organization efficiency improvement. Conduction of various stimulating contests looks as one of the approaches to stimulate research organizations and universities activity. Procedures automation for each contest both significantly simplifies them and solves a number of administrative problems. For this purpose, it seems convenient to use mechanisms of CRIS-systems, which are widely used in scientific and educational organizations to date.

In this paper we present an approach for the automation of the contest procedures. Such approach has been implemented in the ISTINA information system of Lomonosov Moscow State University.

Keywords: *CRIS-systems, scientometrics, contests, automation.*

For citation:

Vasenin V. A., Zenzinov A. A., Lunev K. V. The Usage of CRIS-systems for the Contest Procedures Automation in Terms of the ISTINA Information System, *Programmnaya Ingeneria,* 2016, vol. 7, no. 10, pp. 472—480.

DOI: 10.17587/prin.7.472-480

References

1. **KIAS** RFFI, available at: https://kias.rfbr.ru (in Russian).
2. **IAS** RNF, available at:: http://grant.rscf.ru (in Russian).
3. **IS** RGNF, available at: http://grant.rfh.ru (in Russian).
4. **Portal** for contest applications registration, available at: http://konkurs2014.fcpir.ru (in Russian).
5. **Grants** management system — WizeHive Select. WizeHive, available at: http://www.wizehive.com/grant-management-system/
6 **Konkurs-onlajn.** Servis dlya avtomatizacii konkursov (Contest-online. Contest automation service), available at: http://www.konkurs-online.ru (in Russian).
7. **Sadovnichij V. A., Afonin S. A., Bahtin A. V.** et al. *Intellektual'naya sistema tematicheskogo issledovaniya nauchno-tekhnicheskoj informacii (ISTINA)* (Intelligent system for thematic study of scientific and technical information), Moscow, Izdatel'stvo Moskovskogo universiteta, 2014, 262 p. (in Russian).
8. **Informacionno**-analiticheskaya sistema soprovozhdeniya nauchno-issledovatel'skoj deyatel'nosti SPbGU (Research activities support information-analytical system), available at: https://ias.spbu.ru (in Russian).
9. **Afonin S. A., Lunev K. V.** Vyyavlenie tematicheskih napravlenij v kollekcii naborov klyuchevyh slov (Thematic areas identification in the collection of keywords sets), *Programmnaya inzheneriya,* 2015, no. 2, pp. 29—39. (in Russian).

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4
Технический редактор *Е. М. Патрушева.* Корректор *Т. В. Пчелкина*

Сдано в набор 05.08.2016 г. Подписано в печать 21.09.2016 г.Формат 60×88 1/8.Заказ PI1016
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз".Отпечатано в ООО "Адвансед солюшнз".
119071, г.Москва, Ленинский пр-т, д. 19, стр.1. Сайт: **www.aov.ru**