ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г.

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

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 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.

УДК 004.41

В. В. Липаев, д-р техн. наук, проф., гл. науч. сотр., Институт системного программирования РАН, г. Москва,
e-mail: vlip28@mail.ru

Четыре эталона при производстве сложных программных продуктов больших систем реального времени

На основе анализа результатов исследований и собственного опыта автора выделены и рассмотрены особенности четырех базовых эталонов, используемых при производстве сложных программных продуктов систем реального времени. Представлены основные особенности этих эталонов и применения, их адекватности базовым требованиям при производстве программных продуктов для обеспечения их высокого качества.

Ключевые слова: *программные продукты, эталоны, требования, компоненты, тесты, динамические комплексы программ, имитаторы внешней среды*

При заказе и создании сложных комплексов программ реального времени важно учитывать, что только заказчик и потенциальный пользователь системы может корректно и в полном объеме формулировать **базовые требования (первый эталон)**. Такие требования впоследствии позволяют с большей степенью уверенности судить, насколько успешно проведена разработка соответствующего программного продукта. Эта деятельность требует привлечения специалистов наивысшей квалификации и тесной совместной работы представителей заказчика (основных потребителей) и разработчиков разной квалификации. Реализацию требований в сложных проектах можно отразить тремя основными эталонами, которые выполняются следующими специалистами [1, 2]:

- программистами, создающими программные модули, компоненты и комплексы программ в целом, которые формируются на базе требований;
- тестировщиками, разрабатывающими тесты, и корректирующими модули, компоненты и программный продукт для полного его соответствия базовым требованиям;
- разработчиками динамических имитаторов внешней среды, выполняющими, при необходимости, корректировку сложных программных продуктов с тем, чтобы обеспечить их соответствие базовым требованиям в реальном времени.

Детализация базовых требований заказчика к отдельным программным модулям и компонентам, к комплексу программ в целом, к выполняемым ими процессам и результатам обработки информации и к возможностям их эффективной реализации отражается в более точных и полных спецификациях этих требований (**второй эталон**). Таким образом, должна быть предусмотрена корректировка, конкретизация и развитие совокупности базовых требований к программам в процессе их системного проектирования и в дальнейшем, по мере реализации проекта при тесном взаимодействии заказчика и разработчика. Предварительный анализ и моделирование процессов обработки данных при системном проектировании комплекса программ должны проходить этапы от простого установления базовых отношений между понятиями через определение интерфейсов доступа и атрибутов к модели структуры, состояний и взаимодействий между реальными модулями, компонентами и процессами программного комплекса.

Наборы тестов являются **третьим эталоном функций и характеристик модулей, компонентов и программных комплексов,** которые должны адекватно отражать и полностью покрывать возможности проверки реализации базовых требований к конкретному программному эталону. При разработке необходимо учитывать пригодные для применения тестирования исходные ***эталоны — требования и/или сценарии использования функций*** системы. Их необходимо анализировать и определять в терминах адекватных требований к функциям и характеристикам заданного программного комплекса, а также ***к тестами***, устанавливающими их содержание и корректность. Такая форма описаний содержания программного комплекса, компонентов и модулей должна формироваться и использоваться для сквозной "сверху вниз" верификации спецификаций базовых требований к тестам, а также подвергаться верификации на соответствие исходным требованиям к комплексу, компонентам и модулям программ разного уровня. Таким образом, отражается ***суть программного продукта на следующих трех языках***: требований, программирования и тестирования, что способствует обеспечению его высокого качества.

В динамических системах реального времени тесты, учитывающие взаимодействие с внешней средой, могут формироваться автономно в виде моделей объектов, взаимодействующих с основной системой и с отдельными ее программными продуктами **(четвертый эталон)**. Группа специалистов, реализующих динамическое тестирование, при необходимости должна создавать специальные инструментальные средства — ***генераторы динамических тестов***, которые на основе некоторого набора правил автоматически моделируют тесты. Эти правила можно получать из спецификаций базовых ***требований к программному продукту (часть первого эталона)***. В результате тестирования сложных программных комплексов необходимо достоверно устанавливать ***степень их соответствия утвержденным базовым требованиям к динамическим функциям и характеристикам качества системы***. Для этого они должны проходить тщательные динамические испытания в условиях их последующего применения. В реальных системах создание таких условий может быть очень ресурсозатратно и связано с крупными рисками проявления неустраненных в процессе тестирования дефектов и ошибок. Альтернативой такому развитию событий является создание и реализация математических моделей, ***динамически имитирующих реальную внешнюю среду*** для последующей генерации, учитывающих эту среду тестов в целях анализа режимов функционирования комплексов программ. Таким образом, формируется возможность выполнять тестирование, выявлять и устранять ошибки, а также ***достигать высокого уровня соответствия программного продукта заданным базовым требованиям в динамическом режиме его тестирования*** **(первому эталону)**.

Рис. 1. Состав эталонов при производстве сложных программных продуктов больших систем реального времени

Использование эталонов (рис. 1) при производстве программных продуктов предназначено для повышения их качества до уровня базовых требований. Для оценивания качества сложных заказных программ в некоторых организациях применяется уровень дефектов (ошибок) на тысячу строк программного кода при соответствующем эталоне. Наибольший интерес представляют реальные оценки качества полностью завершенных и сертифицированных программных продуктов специальных систем (четвертый эталон, например, Шатл). Для них предельный уровень составил около 0,05 ошибки на тысячу строк кода. Экспертные оценки числа невыявленных ошибок в программах других эталонов предполагают на порядок большие значения, а именно — 0,5 ошибок (третий эталон), около пяти ошибок (второй эталон) [3, 4]. Эти данные могут служить ориентирами при производстве программных продуктов.

Первый эталон. Требования к программному продукту

Разработка требований состоит в преобразовании потребностей заказчика, выраженных в виде пользовательского представления о системе и программах, в ***формализованные функциональные возможности производства***. В ходе этого процесса должно создаваться четкое представление о будущей системе, которая будет удовлетворять требованиям заказчика и не потребует специальных мероприятий в связи с ее практическим применением. В результате определяется комплекс подлежащих оценке требований. К их числу относятся требования к функциям и характеристикам, которыми должна обладать система, и к значе-

ниям этих функций, позволяющим удовлетворить требованиям заказчика.

Исходные требования заказных систем должны быть зафиксированы в ***соглашении*** (договоре и/или техническом задании) между заказчиком и выполняющими проект руководителями и специалистами. Такое соглашение должно отражать потребности заказчика и пользователей в таком виде, чтобы разработчики могли построить удовлетворяющий их комплекс программ, его компоненты и модули. Требования должны являться конкретным **базовым эталоном** при разработке комплекса программ. При тестировании и других испытаниях эти требования должны предоставлять возможность достоверно установить их выполнение и отсутствие дефектов и ошибок. Для обеспечения соответствия программного продукта требованиям необходимо ***регулярное планирование и управление их обнаружением в эталонах*** с учетом ***ограниченных ресурсов***, выделяемых заказчиком на весь жизненный цикл продукта. Эти данные должны устанавливать цель, функции, содержание и значения результатов исполнения программ, которые следует получать системе или пользователям при определенных условиях и исходных данных. Ниже рассматриваются современные объекты тестирования — сложные комплексы программ ***высокого качества*** (сотни тысяч строк) для систем обработки информации и управления в реальном времени, разрабатываемые большими коллективами — командами специалистов, создающими ***программные продукты, компоненты и модули.*** Таким образом, в результате тестирования программные комплексы должны соответствовать **целостному комплексу требований** (**первому эталону**) — к функциям, характеристикам, архитектуре и качеству, согласованному с разработчиками и утвержденному заказчиком.

Анализ корректности сформированных требований к системе и программному продукту включает: выявление и идентификацию противоречивых, пропущенных, неполных, неоднозначных, нелогичных или непроверяемых требований; расстановку приоритетов и разрешение вопросов, возникающих в связи с определением требований. Сюда же относятся требования, которые не могут быть реализованы или те, которые реализовывать нецелесообразно. Необходимо достигать соглашения совместно с заинтересованными лицами по решениям, касающимся противоречивых, нецелесообразных и неосуществимых требований, устанавливать, чтобы требования были откорректированы.

Для конкретного комплекса программ доминирующие требования выделяются и определяются его ***функциональным назначением***. Программы как ***объекты*** производства, испытаний и оценки качества можно описать в жизненном цикле следующими ***обобщенными характеристиками***:

- проблемно-ориентированной областью применения, техническим и социальным назначением программного комплекса;
- конкретным классом и назначением решаемых функциональных задач с достаточно определенной областью применения квалифицированными пользователями;
- масштабом и сложностью комплекса программ и базы данных, решающих единую целевую задачу системы;
- необходимыми составом и требуемыми значениями характеристик качества функционирования программ и размером допустимого ущерба — риска неблагоприятных событий вследствие недостаточного их качества;
- реальными ограничениями всех видов ресурсов проекта;
- степенью связи решаемых задач с реальным масштабом времени или допустимой длительностью ожидания результатов решения задач;
- степенью необходимой документированности программного продукта.

Системная эффективность применения программных продуктов в стандартах **ISO 9126, ISO 25000** определяется степенью удовлетворения потребностей заинтересованных лиц — заказчиков и/или пользователей. Такие потребности в ряде случаев желательно измерять экономическими категориями: прибылью, стоимостью, трудоемкостью, предотвращенным ущербом, длительностью применения и другими подобными им. В стандартах эта эффективность отражается основной обобщенной характеристикой качества — ***функциональной пригодностью***. ***Ограниченные ресурсы*** для реализации требований функциональной пригодности могут негативно отражаться на следующих конструктивных характеристиках: на надежности, на безопасности, на пропускной способности, на качестве взаимодействия с внешней средой и с пользователями, на качестве документации и других эксплуатационных факторах.

Структура документации требований и формы отдельных документов, используемых для конфигурационного управления программами, должны позволять ***точно документально описывать*** и идентифицировать каждую оформленную версию программных компонентов и продуктов в целом в любое время на всем протяжении их жизненного цикла. Особое значение при управлении конфигурацией имеет ***документация на реализованные изменения и тесты,*** с помощью которых проверяется корректность версий компонентов и комплекса в целом. Эта документация должна позволять восстанавливать ***историю разработки и проверки*** каждого изменения любого компонента. На базе всего комплекса использованных тестов для каждой версии программного продукта создаются и документируются ***эталонная тестовая (контрольная) задача и контрольные результаты ее***

решения. При создании особо сложных систем целесообразно выделение специального коллектива, обеспечивающего организацию и реализацию основных ***системных работ по документообороту***. Организация процессов документирования должна обеспечивать гибкое и точное изменение базовых документов — ***сопровождение и конфигурационное управление версиями и редакциями каждого документа требований*** [5, 6].

Для реализации на практике требований и планов их документирования в жизненном цикле программных продуктов необходимы ***организационные мероприятия***, гарантирующие участникам проектов ***определенную культуру, дисциплину разработки и применения документов требований***. Специалистам и заказчикам разной квалификации и специализации такая организационная система должна обеспечивать возможность эффективного взаимодействия при решении требуемых комплексных задач. К их числу относятся накопление, хранение и обмен упорядоченной информацией о состоянии и изменениях компонентов проекта.

Второй эталон. Сборка модулей и программных компонентов в комплекс программ

Планирование разработки и сборки модулей и компонентов для комплекса программ должно включать [3, 6]:

- выбор метода (нисходящий или восходящий) разработки и сборки модулей и программных компонентов;
- планирование порядка разработки программ модулей и компонентов для комплекса программ;
- документирование квалификации сотрудников, которая необходима для разработки модулей и компонентов;
- выбор, подготовку и утверждение детализированных планов производства программных модулей и компонентов;
- подготовку графиков разработки и выполнения для модулей и компонентов комплекса программ;
- применение графиков для планирования производства компонентов и комплексов программ.

В зависимости от роли и места сборки в общем технологическом процессе создания комплексов программ различают два метода разработки: ***нисходящий*** и ***восходящий***. В случае нисходящего метода компоненты высокого уровня интегрируются до окончания проектирования, комплексирования и реализации комплекса программ. В случае восходящего метода перед разработкой и сборкой компонентов более высокого уровня сначала интегрируются модули и компоненты нижнего уровня.

Нисходящий технологический процесс основан на сочетании пошаговой детализации технического задания на последовательных этапах проектирования с ее завершением на таком уровне, когда проект может быть описан из готовых компонентов и модулей, хранящихся в базе данных. На основании требований технического задания и описания проблемной области создается параметризованное функциональное описание разрабатываемого комплекса программ. При проектировании осуществляется последовательная разработка спецификаций требований к создаваемому комплексу, к его составным компонентам и модулям.

При построении версий из повторно используемых компонентов (ПИК) может не оказаться элементов, полностью подходящих по функциональным и программным признакам. В этом случае недостающие части модулей разрабатывают как ПИК с тем, чтобы конструктивно-технологические требования ко всем составным частям комплекса соответствовали типовой структуре, выработанной для всей проблемной области. При наличии ограничений осуществляется контроль ПИК и создаваемых компонентов по конструктивным показателям (объем памяти, интерфейсы, время функционирования). По завершении этого этапа в инструментальном комплексе должны содержаться описания (спецификации) всех программных и информационных модулей и компонентов. Эти компоненты должны быть согласованы в функциональном и структурном планах и по конструктивно-технологическим параметрам. В результате может быть подготовлена автоматизированная сборка комплекса программ.

Нисходящее проектирование является частью процесса разработки сложных программных комплексов, при котором сначала разрабатывают компоненты верхнего уровня, а затем компоненты, находящиеся на нижних уровнях иерархии. При наличии неполного состава готовых ПИК комплекс можно представить в виде одного абстрактного компонента с упрощенными, временными функциональными заглушками, которые последовательно заменяют на реальные модули и компоненты (рис. 2).

Заглушки-имитаторы должны иметь такой же интерфейс, что и готовые ПИК, но с ограниченной функциональностью. После того как компоненты верхнего уровня запрограммированы, таким же образом реализуют его субкомпоненты. Процесс продолжается до тех пор, пока не будут полностью реализованы компоненты и модули самого нижнего уровня. Затем весь комплекс программ и система собираются целиком.

Восходящий технологический процесс применяют в тех случаях, когда разработчики хорошо знают структуру и имеют весь состав готовых модулей и компонентов. При этом исходят из того, что существует ***прототип*** создаваемого комплекса; соглашения, принятые при разработке прототипа, распространяются и на новый программный комплекс. Предполагается функциональное подобие нового комплекса и прото-

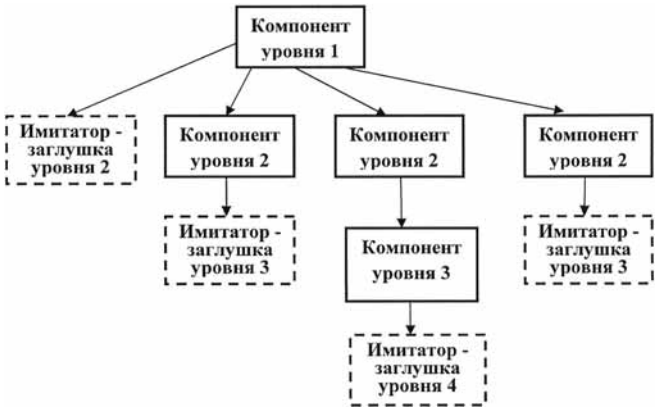


Рис. 2. Нисходящее тестирование сборки компонентов

типа, разработка осуществляется итерационным путем. Вначале оценивают состав и качество имеющихся готовых ПИК, из которых планируют собрать версию, выбирают направления и объекты для последовательных доработок. Доработки, как правило, касаются изменения и совершенствования функциональных возможностей, интерфейсов разных видов при смене оборудования или операционной системы, изменений эргономических требований или модификации дисциплины обмена с абонентами сети или вычислительной системы. Доработки не всегда удается сделать так, чтобы полностью удовлетворить требованиям комплекса программ. По этой причине по завершении доработок в какой-то части комплекса осуществляется повторная его сборка и оценка полноты реализации требований технического задания. Этот цикл работ может повторяться в зависимости от соотношения требующихся и завершенных доработок компонентов и модулей, особенно в условиях сжатых сроков разработки. Итерации, завершающиеся сборкой, осуществляют для распараллеливания работ по доработке компонентов и по комплексированию предварительного варианта группы компонентов и модулей. Восходящий технологический процесс применяют обычно в коллективах, в течение ряда лет разрабатывающих однотипные комплексы программ для разных заказчиков (рис. 3).

При восходящем технологическом процессе по завершении испытаний программного продукта полезно осуществлять отбор компонентов, которые могут рассматриваться как повторно используемые, а также осуществлять ввод в базу данных уже прошедших ранее тестовые испытания ПИК. В качестве нового может рассматриваться компонент, полученный в процессе его доработки из ранее гарантированного ПИК. В этом случае он отличается от старого либо расширенными функциями и областью определения параметров, либо учетом какого-либо нового вида интерфейса или устройства. Вопрос о целесообразности замены в ранее отлаженных комплексах ПИК на новый, доработанный, следует решать с большой осторожностью. Это особенно важно при использовании ПИК в комплексах, сильно связанных по информации (например, в системах реального времени). В результате включение нового ПИК вместо старого может приводить к необходимости повторной комплексной отладки всего программного комплекса.

Третий эталон. Тесты при разработке сложных комплексов программ

Основная **цель тестирования комплекса программ** и его функциональных компонентов состоит в том, чтобы обнаруживать, регистрировать и устранять дефекты и ошибки, которые внесены во время последовательной разработки и реализации **базовых требований к его функциям и характеристикам**. Понятие **тестирование** далее используется в широком смысле этого слова, включая (не явно) устранение обнаруженных дефектов и ошибок. Особые специалисты и работы требуются для ***отладки*** комплексов программ, корректировки, устранения ошибок и обеспечения их соответствия исходным требованиям к заданным функциям и качеству. Для этого необходима локализация ошибок, их диагностика, разработка корректировок программ, контроль выполненных изменений и ***регрессионное тестирование*** для регистрации результатов устранения обнаруженных ошибок (см. рис. 2 и 3).

Процессы корректировки являются особенно сложными и требуют участия в них соответствующих

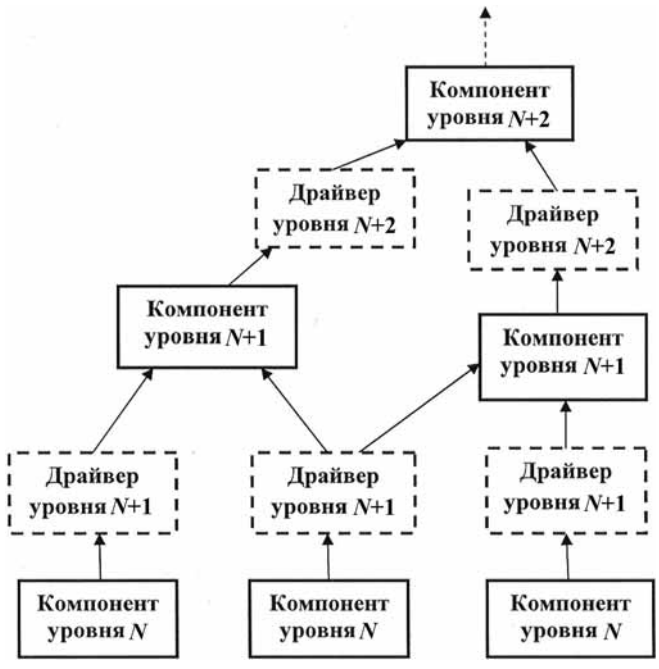


Рис. 3. Восходящее тестирование сборки компонентов

программистов модулей и компонентов. Иногда в такой корректировке участвуют не только программисты, в продуктах которых обнаружены ошибки, но и руководители разработки компонентов и может быть всего комплекса программ. Тем самым, процессы ***выходят из сферы тестирования*** **(третий эталон)** в область разработки текстов программ **(второй эталон)**, для чего необходимы специалисты, участвующие в создании всего комплекса программ [2, 5, 6].

В ходе тестирования продуктов ***разрабатываемых "снизу вверх"*** (восходящая сборка) вызываемые компоненты предшествуют вызывающим, которые являются источниками информации для вызывающих. Вызываемые компоненты при этом могут использоваться как генераторы тестов для вызывающих компонентов. В планах следует учитывать эти связи и распределять во времени поставку компонентов для сборки и комплексирования с учетом последовательности их времени разработки и иерархии связей. На графике планирования работ эти связи должны отражаться между моментами времени завершения разработки и тестирования вызываемого компонента и началом тестирования вызывающего компонента в составе фрагмента комплекса программ. Однако реальные процессы и последовательности программирования и автономного тестирования компонентов не всегда позволяют соблюдать рациональную логику последовательной разработки компонентов "снизу вверх" с учетом времени подготовки и взаимодействия в комплексе программ. Тогда для предварительного тестирования групп вызывающих компонентов "сверху вниз" приходится разрабатывать временные имитаторы-заглушки для тестов, подменяющие вызываемые компоненты.

Планирование работ по тестированию должно учитывать ресурсы и работы, которые необходимо выполнить, чтобы своевременно ***подготовить тестовую среду*** для планомерного продолжения тестирования. Тестировщики каждого компонента должны определять требования к аппаратному, программному и сетевому обеспечению в целях создания и поддержки адекватных изменений тестовой среды. Нужно планировать работы по программированию, приобретению, установке и настройке компонентов, моделей или генераторов тестовой среды. Создание плана тестирования — ***итеративный процесс***, требующий обратной связи с различными участниками проекта и согласованности с определенными в нем процессами, стратегиями тестирования и сроками выполнения работ. С этим планом должен быть коррелирован и предшествовать ему план программирования и подготовки к тестированию модулей и компонентов функционально сложных задач.

Тест-менеджер должен ***утвердить стратегию программирования компонентов, их тестирования*** и тестовые процедуры, которые должны быть подробно описаны в плане тестирования; определить какие компоненты и модули, сценарии и тесты когда будут выполняться. Кроме того, предполагается, что руководитель проекта согласен с тем, что план тестирования компонентов и связанные с ним тестовые сценарии достаточно хорошо проверяют покрытие тестами требований, эталонов или сценариев использования комплекса программ и системы. Подробное изучение системных требований или сценариев применения системы вместе с тщательным определением параметров плана тестирования и требований к тестам необходимы для эффективного тестирования модулей и компонентов программного комплекса.

План тестирования должен определять и учитывать объем и длительность работ по тестированию каждого компонента. Обычно выстраивают ***структуру последовательности работ***, в которой на одном уровне определяют категории работ по тестированию компонентов, а на другом уровне — подробные описания работ. Структура детализации работ используется в сочетании с хронометражем для определения длительности выполнения этапов тестирования каждого компонента или модуля. Кроме того, план тестирования должен отражать ***оценки затрат*** на тестирование. Оценка затрат может определять число сотрудников группы тестирования компонентов в проекте в часах или число специалистов, если для выполнения определенного объема работ выделяется конкретный срок. По возможности в план тестирования помещают такие оценки затрат, как планируемое число тестовых процедур и сценариев.

Верификация и аттестация системной архитектуры при нисходящем тестировании имеют больше возможностей выявить ошибки в архитектуре и требованиях к комплексу программ на раннем этапе разработки. Обычно это структурные и функциональные ошибки, раннее выявление которых предполагает их исправление почти без дополнительных затрат. При восходящем тестировании структура высокого уровня не утверждается вплоть до последнего этапа разработки всего комплекса программ.

Демонстрация комплекса программ может быть возможна при нисходящей разработке уже на ранних этапах. Этот факт является важным психологическим стимулом использования нисходящей модели разработки программных комплексов, поскольку демонстрирует заказчику осуществимость всей управляющей системы. Аттестация проводится на начальном этапе процесса тестирования путем создания ***демонстрационной версии комплекса*** (с заглушками). Однако если система создается из ПИК, то и при восходящей разработке также можно создать ее демонстрационную версию.

Сложность реализации тестов при нисходящем тестировании выше, так как необходимо моделировать программы-заглушки компонентов и модулей нижних уровней. ***Программы-заглушки*** могут быть

упрощенными функциями версий представляемых компонентов. При восходящем тестировании для того чтобы использовать компоненты нижних уровней, необходимо разработать ***тестовые драйверы***, которые эмулируют окружение компонента или модуля в процессе тестирования (см. рис. 3).

Наблюдение за ходом тестирования — при нисходящем и восходящем тестировании могут возникать проблемы, связанные с контролем за состоянием и результатами тестирования. В большинстве комплексов, разрабатываемых "сверху вниз", более верхние уровни системы, которые реализованы первыми, не генерируют выходные данные, однако для проверки этих уровней нужны какие-либо имитированные выходные результаты. Испытатель должен создать искусственную среду для генерации результатов теста. При восходящем тестировании также может возникнуть необходимость в создании искусственной среды (***тестовых драйверов***) для исследования функционирования компонентов и модулей нижних уровней.

На практике при разработке и тестировании сложных систем чаще всего используется ***композиция восходящих и нисходящих методов***. Разные сроки разработки для разных компонентов предполагают, что группа, проводящая тестирование и интеграцию, должна работать с какими-либо готовыми компонентами. Поэтому во время процесса тестирования сборки в любом случае приходится разрабатывать как заглушки, так и тестовые драйверы.

Структура документации и формы отдельных документов, используемых для конфигурационного управления тестированными программами, должны позволять точно документально описывать и идентифицировать каждую оформленную версию программных компонентов и продуктов в целом в любое время на всем протяжении их жизненного цикла. Особое значение при управлении конфигурацией имеет ***документация на реализованные изменения и тесты,*** с помощью которых проверяется корректность версий компонентов и комплекса в целом. Эта документация должна позволять восстанавливать ***историю разработки и проверки*** каждого изменения любого компонента.

Четвертый эталон. Требования к генерации динамических тестов внешней среды в реальном времени

Характеристики динамического (контролируемого во времени) режима функционирования программных продуктов зависят не только от их внутренних свойств, но и ***от свойств внешней среды***, в которой они применяются **(четвертый тип эталонов)**. Для сокращения неопределенностей и прямых ошибок при оценивании качества программ необходимо до начала испытаний определить основные параметры внешней среды и потоки информации, при которых должен функционировать весь комплекс программ. Комплекс при этом должен поддерживать требуемые (планируемые) при его оценивании функциональные возможности (функции) и характеристики качества. Для этого заказчик и разработчик должны совместно структурировать, описать и согласовать ***модель внешней среды*** и ее параметры как в среднем, типовом режиме применения, так и в наиболее вероятных и критических режимах, в которых должны обеспечиваться требуемые характеристики качества динамического функционирования программного продукта. Такая ***модель должна отражать следующие характеристики***:

- характеристики внешних динамических потоков информации, в том числе их распределение по видам источников, характеристикам качества данных и возможным дефектам;
- интенсивность и структуру типовых, оперативно поступающих на этапе тестирования сообщений от пользователей и администраторов, их необходимую квалификацию, отражающуюся на вероятности ошибок и качестве выдаваемой информации;
- характеристики возможных негативных и несанкционированных воздействий от внешней среды при применении программного продукта;
- необходимые характеристики вычислительных средств, которые определяют среду функционирования программного продукта с требуемым качеством.

В отличие от натурного эксперимента моделирование внешней среды и динамических тестов на компьютере имеет значительно большие возможности как по заданию и контролю исходных данных, так и фиксации всех промежуточных и выходных результатов функционирования испытываемого программного продукта. В реальных системах ряд компонентов иногда оказывается недоступным для контроля их состояния, так как либо невозможно поместить измерители контролируемых сигналов в реальные подсистемы, подлежащие тестированию, либо это сопряжено с изменением характеристик самого анализируемого объекта. В отличие от этого натурные эксперименты зачастую невозможно остановить на некоторой промежуточной фазе или повторить с абсолютно теми же исходными данными.

Одними из наиболее сложных и дорогих имитаторов внешней среды, применяемых для испытаний крупных программных продуктов, являются следующие модели: полета космических аппаратов; диспетчерских пунктов управления воздушным движением; объектов систем противовоздушной обороны; сложных административных систем и др. [3, 6]. Применяемые для этого моделирующие испытательные стенды (МИС) проблемно-ориентированы и размеры программ, моделирующих в них динамическую внешнюю среду, могут даже значительно превышать размеры соответствующих испытываемых программных продуктов.

Для обеспечения высокого качества сложных комплексов программ реального времени необходимы со-

ответствующие ***проблемно-ориентированные интегрированные системы автоматизации динамического тестирования***. Подобные системы должны обладать способностью достаточно полно отразить испытания программ с реальными объектами внешней среды. При этом высокая стоимость и риск испытаний с реальными объектами почти всегда оправдывают значительные затраты на компьютерные интегрированные системы. Последнее особенно актуально, если предстоят испытания критических программных продуктов с высокими требованиями к их качеству функционирования, с длительным жизненным циклом и множеством развивающихся версий.

Инструментальные средства автоматизации динамических испытаний комплексов программ реального времени должны обеспечивать:

- моделирование внешней среды — поддержку процесса тестирования с помощью модели динамической имитации данных из внешних источников для программного продукта, поддерживающего аппаратные компоненты системы;
- определение и формирование динамических тестов — ввод тестовых наборов, генерацию тестовых данных, ввод ожидаемых, эталонных результатов;
- управление тестами и участком программы, для которого средство тестирования может автоматически выполнять тестовые наборы;
- анализ и обработку тестовых результатов — возможность средства тестирования автоматически анализировать части тестовых результатов, включая сравнение ожидаемых и реальных результатов; статистическую обработку результатов.

Методы ***динамических испытаний комплекса программ***, в большей или меньшей степени, ориентированы на обнаружение ошибок определенных типов, преимущественно в структуре комплекса программ и реализуемых ими маршрутах обработки информации. Такая ориентация позволяет упорядочивать последовательность приоритетного применения методов в целях устранения, прежде всего, ошибок, в наибольшей степени отражающихся на корректности исполнения программ. Этот подход дает возможность сосредоточиваться на методах, позволяющих решать частные задачи, достигая при этом необходимого качества их решения и соответствия предъявляемым требованиям при минимальных затратах.

Средства генерации динамических тестов и имитации внешней среды в составе поставляемого программного продукта предназначены для оперативной подготовки исходных данных при проверке различных режимов функционирования в процессе применения программного продукта и при диагностике проявившихся дефектов. Минимальный набор средств генерации тестов должен передаваться пользователям для контроля режимов эксплуатации рабочих версий в реальном времени. Такой набор должен входить в комплект поставки каждой пользовательской версии программного продукта.

Важной функцией испытательных стендов является их использование в качестве ***тренажеров для операторов-пользователей***. Так как качество функционирования комплексов программ может существенно зависеть от особенностей, характеризующих конкретного человека, участвующего в эксплуатации и обработке информации, то эти характеристики необходимо измерять. Необходимо также иметь возможность их улучшать до уровня, обеспечивающего выполнение ***заданных требований к программному продукту***. По этой причине в процесс испытаний органически должен входить процесс динамической тренировки и измерения параметров, характеризующих режимы реальной работы и реакции операторов, а также использование МИС для обучения и регулярной подготовки операторов-пользователей в процессе тиражирования и эксплуатации программного продукта.

Для определения качества программных продуктов существенна ***адекватность имитаторов динамических тестов***. Уровень их адекватности зависит от степени учета второстепенных факторов, характеризующих функционирование реальных объектов или источников информации, при создании моделей. Точность моделей, прежде всего, определяется алгоритмами, на которых они базируются, и полнотой учета в них всех особенностей моделируемых объектов. Кроме того, на адекватность имитации влияет уровень дефектов и ошибок в программах имитации.

Регистрация и обработка характеристик динамических тестовых данных должны обеспечивать их контроль на соответствие заданным требованиям к программному продукту, обобщенным характеристикам каждого объекта внешней среды и исходным данным сеанса испытаний. Так проводится процесс испытаний по конечным результатам функционирования комплекса программ, которые выдаются внешним абонентам для определения интегральных характеристик качества и их соответствия требованиям, предъявляемым к программному продукту.

Список литературы

1. **Бейзер Б.** Тестирование черного ящика. Технология функционального тестирования программного обеспечения и систем. Пер. с англ. СПб.: ПИТЕР, 2004.
2. **Блэк Р.** Ключевые процессы тестирования. Пер. с англ. М.: ЛОРИ, 2006.
3. **Липаев В. В.** Тестирование компонентов и комплексов программ. Учебник. М.: СИНТЕГ, 2005.
4. **Тэллес М., Хсих Ю.** Наука отладки. Пер. с англ. М.: Кудиц-образ, 2003. 560 с.
5. **Уайт Б. А.** Управление конфигурацией программных средств. Практическое руководство. Пер. с англ. М.: ДМК Прес, 2002.
6. **Фатрелл Р. Т., Шафер Д. Ф., Шафер Л. И.** Управление программными проектами: достижение оптимального качества при минимальных затратах. Пер. с англ. М.: Вильямс, 2003.

УДК 004.4'418

Н. И. Вьюкова, ст. науч. сотр.,
В. А. Галатенко, д-р физ.-мат. наук, ст. науч. сотр.,
С. В. Самборский, ст. науч. сотр., НИИ системных исследований РАН, г. Москва,
e-mail: niva@niisi.msk.ru

Open64: инфраструктура разработки компиляторов

Рассмотрена инфраструктура построения компиляторов Open64. Представлена общая организация Open64, внутреннее представление программ, средства описания целевых архитектур. Дано сравнение Open64 с другими свободно распространяемыми компиляторами — GCC и LLVM.

Ключевые слова: *компилятор, оптимизация гнезд циклов, внутреннее представление программы, WHIRL*

Введение

Компилятор Open64 [1] — это открытый оптимизирующий компилятор, который поддерживает несколько целевых архитектур. Исходные тексты и бинарные дистрибутивы, находящиеся в открытом доступе, обеспечивают поддержку архитектур Intel IA-64 (Itanium) и AMD64 (x86_64). Пользователям, желающим применять Open64 для других архитектур, необходимо обращаться за поддержкой в организации, ведущие соответствующие разработки (MIPS; Loongson — Китайская Академия Наук; CUDA — NVIDIA Inc.; PowerPC — Университет Tsinghua; SL — SimpLight Nanoelectronics Inc.).

Компилятор Open64 представляет собой ответвление компилятора MIPSPro фирмы SGI для MIPS-процессора R10000. В 2000 г. он был выпущен под названием Pro64 с лицензией GNU GPL. Годом позже проект был принят Университетом Делавэра и переименован в Open64. В настоящее время он по большей части используется в качестве исследовательской платформы группами специалистов, занимающихся разработкой компиляторов и микропроцессорных архитектур.

Компилятор Open64 поддерживает языки Фортран 77/90/95 и Си/Си++, а также API OpenMP для программирования кроссплатформенных многопоточных приложений с разделяемой памятью. Компилятор обеспечивает высококачественный межпроцедурный анализ, анализ потоков данных, анализ зависимостей по данным, анализ регионов в массивах. Сильной стороной Open64 являются средства оптимизации гнезд циклов.

Инфраструктура Open64 существует и развивается в виде большого числа ответвлений, каждое из которых имеет соответствующие его специфике функциональные возможности, и может иметь определенные ограничения.

Компания AMD разработала на основе Open64 компилятор x86 Open64 [2], дополненный оптимизациями для многоядерных архитектур и средствами разработки многопоточных программ. Компания поддерживает Open64 для своих процессоров как дополнение и альтернативу GCC [3].

На базе Open64 компанией Nvidia разработан оптимизирующий компилятор для архитектуры CUDA.

Специалистами Китайской академии наук было осуществлено портирование и доработка Open64 для микропроцессора Loongson II [4], совместимого с архитектурой MIPS. Ими был реализован ряд оптимизаций, таких как профилирование потока управления (*edge profiling*), сжатие массивов (*array contraction*), а также модуль оптимизаций на уровне исходного кода, повышающих степень локальности данных (*LIDO — Locality Inspired Data Optimizer*).

Проект OpenUH [5] дополняет функциональные возможности, предоставляемые Open64, расширенными средствами распараллеливания на основе OpenMP 3.0. В OpenUH также реализована поддержка расширения языка Фортран, принятого в стандарте 2008 г., именуемого Coarray Fortran (CAF). Это расширение позволяет писать на Фортране параллельные

программы для распределенных систем. Привлекательной стороной OpenUH является возможность генерации оптимизированного кода на языке C или Fortran 77, которые могут быть затем скомпилированы обычными компиляторами для целевых платформ, не поддерживаемых напрямую Open64.

В данной работе кратко рассмотрен опыт практического применения компилятора, а также представлены общая структура компилятора; внутреннее представление программ; способ описания целевых архитектур в Open64. Приведено его сравнение с другими свободно-распространяемыми компиляторами Clang/LLVM [6] и GCC.

1. Open64 с точки зрения пользователя

Опции командной строки. Вызов Open64 для компиляции с языков C, C++ осуществляется при помощи команд **opencc** и **openCC** соответственно. Для компиляции с языка Fortran используют команды **openf90**, **openf95**. Основные опции командной строки такие же, как для компилятора GCC, однако существуют и дополнительные опции, отражающие специфические возможности Open64. Получить краткую сводку опций можно при помощи команды **opencc – help**. Подробное руководство пользователя представлено в работе [2].

Оптимизация кода. Отличительной чертой Open64 являются средства оптимизации гнезд циклов, которые автоматически используются при задании оптимизации **-O3**. Эти механизмы тестировались на простом примере программы на языке C, выполняющей копирование двумерного массива по столбцам:

```
unsigned int i, j;
  for (i = 0; i<N; i++)
     for (j = 0; j<M; j++)
        b[j][i] = a[j][i];
```

Компилятор Open64 автоматически выполнил для этого гнезда циклов инвертирование циклов, развертку внешнего цикла (по **j**) и векторизацию внутреннего цикла (по **i**), а также вставку команд предвыборки данных (prefetcht0). Увидеть, какие именно преобразования были выполнены, можно при помощи опций Open64 **-PHASE:lno -CLIST: = ON**, которые задают вывод файла на исходном языке (в данном случае C) после выполнения оптимизаций гнезд циклов. На последующих стадиях компилятор также выполнил развертку цикла по **i**.

В средства GCC, начиная с 2008 г., входит инфраструктура оптимизации гнезд циклов на основе полиэдральной модели [7], но эта функциональность до настоящего времени носит статус экспериментальной. Она не входит в стандартную конфигурацию GCC и соответствующие оптимизации не включаются по умолчанию при задании оптимизации **–O3**. В частности, на рассмотренном выше примере GCC-4.8 не выполнил инвертирование циклов даже при явном задании опции **–floop-interchange**. Тем не менее GCC векторизовал внешний цикл (по **i**), и использовал 128-битные пересылки. При этом массивы в программе были размещены с достаточным для этого выравниванием.

Хотя компилятор Open64, в отличие от GCC, использовал 64-битные пересылки, за счет инвертирования гнезда циклов и предвыборки данных, сгенерированный им код оказался примерно в 15 раз быстрее по сравнению с кодом, сгенерированным GCC. Компилятор Clang/LLVM не выполнил ни одного из упомянутых выше преобразований, а полученная с его помощью программа выполняется в 3 раза медленнее, чем программа, скомпилированная GCC.

На простых тестах вычислений с плавающей запятой (Flops) было получено обратное распределение "призовых мест" по эффективности сгенерированного кода: на первом месте Clang/LLVM, затем GCC и Open64.

Рассмотренные выше примеры ни в коей мере не претендуют на сколько-нибудь полный анализ производительности кода, генерируемого Open64, GCC и Clang/LLVM, но они дают некоторое представление о сильных и слабых сторонах этих компиляторов.

Диагностика. Инфраструктура Open64, которая использует в качестве компилятора переднего плана относительно старую версию gcc-4.2, в отношении средств диагностики уступает Clang/LLVM и современному GCC. В Clang/LLVM изначально была реализована выдача качественной диагностики, включающей высокоинформативные сообщения, а также точное положение ошибки в строке исходного текста, которое отмечается при помощи маркера. При выдаче на терминал используется цветовая подсветка сообщений. В GCC начиная с версии 4.8 реализованы средства диагностики, аналогичные имеющимся в Clang/LLVM (правда, пока без подсветки, поддержка которой ожидается в версии gcc-4.9).

Следует отметить развитые средства анализа, применяемые для выявления ошибок в версии gcc-4.8. Этот компилятор сумел обнаружить ошибку при обращении к элементу массива, которую не выявил ни один из компиляторов gcc-4.7, LLVM/Clang-3.3, Open64-5.0, в следующем коде:

```
#define N 2048
#define M 4096
float a[M][N] = {0.0f};
float b[M][N] = {0.0f};
int main () {
  unsigned int i, j;
  for (i = 0; i<N; i++)
     for (j = 0; j<M; j++)
        b[i][j] = a[i][j];
  return 0;
}
```

Здесь в заголовках циклов перепутаны граничные значения индексов *M* и *N*. Сообщение, выданное gcc-4.8:

```
lnoerr.c:46:21: warning: iteration 2048u invokes undefined
behavior [-Waggressive-loop-optimizations]
     b[i][j] = a[i][j];
                   ^
lnoerr.c:44:5: note: containing loop
     for (j = 0; j<M; j++)
     ^
```

Расширения. Инфраструктура Open64 поддерживает большинство расширений, которые реализованы в версии gcc-4.2, в том числе ассемблерные вставки (операторы **asm**) и почти все атрибуты (за исключением **init_priority** и **java_interface**), целочисленные комплексные типы, вложенные функции, вычисляемые **goto** с переходом на метку вне текущего блока и многие builtin-функции.

Сборка. Сборка Open64 из поставляемого дистрибутива исходных текстов выполняется стандартным образом, при помощи команд **configure**, **make**, **make install**. Однако программа конфигурирования рассчитана только на архитектуры Itanium и x86. В принципе возможна сборка Open64 и для других архитектур, в том числе сборка кросс-компиляторов, однако она требует довольно сложной последовательности шагов, описание которых в документации отсутствует.

2. Структура компилятора Open64

Инфраструктура компиляции Open64 поддерживает языки C, C++, Fortran 90 и Fortran 95, а также директивы OpenMP. Средством коммуникации между компонентами Open64 является внутреннее представление WHIRL [8], которое кратко описано далее в разд. 4. Основными компонентами Open64 являются:

- компиляторы переднего плана (*front ends*), задачей которых является трансформация входной программы с языка высокого уровня во внутреннее представление;
- компилятор нижнего уровня (*backend*), который, в свою очередь, состоит из компонентов, выполняющих:
  - — глобальную оптимизацию;
  - — оптимизацию гнезд циклов;
  - — межпроцедурный анализ и межпроцедурные оптимизации;
  - — генерацию кода.

Существуют также компоненты, выполняющие автоматическое распараллеливание кода и регенерацию исходного кода из внутреннего представления.

Компиляторы переднего плана для языков C и C++ реализованы на основе GCC версии 4.2.0. Для Fortran90/95 используют компиляторы переднего плана SGI Pro64 (Cray). Компиляторы переднего плана транслируют входную программу во внутреннее представление очень высокого уровня (см. подразд. 4.1).

С начальным внутренним представлением имеет дело оптимизатор высокого уровня, который раскрывает вложенные вызовы и операции доступа к элементам агрегатных типов, а также проход, осуществляющий inline-подстановки. Затем оно понижается до внутреннего представления высокого уровня, над которым оперируют проходы межпроцедурного анализа, оптимизатор гнезд циклов, а также проход предварительных оптимизаций (PREOPT).

Далее порождается внутреннее представление среднего уровня, с которым работают стандартный оптимизатор и первый модуль выявления регистровых переменных. Второй модуль выявления регистровых переменных работает с низкоуровневым внутренним преставлением.

Управление процессом компиляции с учетом опций командной строки выполняет драйвер компиляции. Драйвер осуществляет вызов требуемого компилятора переднего плана, модулей inline-подстановок, модулей межпроцедурного анализа, модулей нижнего уровня, ассемблера, компоновщика. Драйвер обеспечивает интерфейс между этими компонентами посредством промежуточных файлов, а также путем задания подходящих опций командной строки для каждого модуля.

3. Проходы анализа и оптимизации

Инфраструктура Open64 предоставляет следующие три уровня оптимизаций:

–O1 — выполняются локальные оптимизации (очень быстрые);

–O2 — включается расширенный набор консервативных оптимизаций;

–O3 — включаются агрессивные оптимизации, подразумевающие существенные трансформации исходной программы.

Различные виды оптимизаций и анализа Open64 можно классифицировать следующим образом.

Стандартные оптимизации основаны на SSA-представлении программы. Инфраструктура Open64 включает описание целевой архитектуры, на основе которого принимают решения по выбору стратегии оптимизации.

Трансформации циклов для однопроцессных конфигураций доступны для всех поддерживаемых языков. Для принятия решения о выборе трансформаций используется унифицированная стоимостная модель. Для гнезд циклов проводится анализ зависимостей. Набор трансформаций включает: *loop fission* (расщепление циклов); *loop fusion* (слияние циклов); *unroll and jam* (развертка и сжатие циклов); *loop interchange* (инвертирование гнезда циклов); *loop peeling* (вынос начальных или конечных итераций за тело цикла); *loop tiling* (разбиение пространства итераций на блоки в целях оптимального использования кэша); *vector data prefetching* (предвыборка данных в гнездах циклов).

В исходных текстах Open64 содержатся также средства программной конвейеризации циклов, однако при компиляции не удалось наблюдать применение данной оптимизации для архитектуры x86 ни на одном из тестовых примеров.

Механизм поддержки параллельного выполнения включает трансляцию директив OpenMP (в вызовы библиотеки pthreads), а также содержит средства автоматического распараллеливания. Последнее включает автоматическую приватизацию массивов, выявление Doacross-циклов и анализ секций массивов для определения циклов, которые могут выполняться параллельно.

4. Внутреннее представление программы

Этот раздел посвящен рассмотрению внутреннего представления программ в компиляторе Open64, включая лежащую в его основе идеологию, уровни представления и специфику каждого уровня.

4.1. Уровни представления

Внутреннее представление программы в компиляторе Open64, называемое WHIRL, служит интерфейсом между компонентами компиляции нижнего уровня. Различают пять уровней WHIRL:

- самый высокий (VH — *Very High*);
- высокий (H — *High*);
- средний (M — *Medium*);
- низкий (L — *Low*);
- самый низкий (VL — *Very Low*).

Структура WHIRL разрабатывалась с учетом возможности поддержки множества входных языков, таких как C, C++, Ada, Fortran77, Fortran 90 и др. Представление WHIRL, генерируемое компилятором переднего плана, является машинно-независимым. Оно ориентировано не на конкретную микропроцессорную архитектуру, а на абстрактную C-машину, моделирующую семантику языка C.

В то же время WHIRL обеспечивает генерацию эффективного кода для различных целевых микропроцессоров. В процессе компиляции уровень представления понижается. На низких уровнях оно зависит от целевой архитектуры и содержит только операции, поддерживаемые целевым процессором. С понижением уровня представления изменяются его характеристики. В таблице приведены особенности внутреннего представления и то, как они изменяются с понижением его уровня.

Различия между высокоуровневым и низкоуровневым представлениями

Характеристика	Высокоуровневое представление	Низкоуровневое представление
Типы конструкций	Много различных типов	Число типов конструкций невелико
Последовательность конструкций, представляющая программу	Короткая	Длинная
Форма	Иерархическая	Плоская, последовательная

Теоретически все виды оптимизации могут быть выполнены над представлением самого низкого уровня, соответствующим последовательности машинных команд, поскольку эффект от любой оптимизации в конечном счете должен быть выразимым в терминах результирующего машинного кода. Однако это может быть нежелательным по ряду причин. Разработчики Open64 при выработке внутреннего представления и определении его уровней исходили из следующих положений.

- Соответствие исходной программе. Высокоуровневое представление предоставляет оптимизатору достоверную информацию об исходной программе, что существенно для качественного выполнения многих оптимизаций.
- Гранулярность. Соответствие между гранулярностью представления и гранулярностью сущностей, с которыми оперирует та или иная оптимизация, упрощает и позволяет сделать более эффективной ее реализацию.
- Вариации. При работе с низкоуровневым представлением оптимизатору приходится иметь дело с множеством различных вариантов кода, которые могут выполнять однотипные задачи, что усложняет распознавание ситуаций, в которых применима данная оптимизация.

В целом подход заключается в том, чтобы применять оптимизации на максимально высоком уровне представления, на котором это возможно без потери качества.

Структура WHIRL спроектирована таким образом, что она применима ко всем уровням промежуточного представления, за исключением наиболее низкого, соответствующего последовательности машинных команд. Для каждого уровня внутреннего представления сначала выполняется набор соответствующих ему оптимизаций, а затем вызывается "функция понижения", трансформирующая текущее представление в представление следующего, более низкого уровня. В конце процесса компиляции генератор кода транслирует представление WHIRL самого низкого уровня в свое собственное представление, отражающее последовательность машинных команд.

4.1.1. Самый высокий уровень WHIRL

Этот уровень WHIRL — VH генерируется компиляторами переднего плана и практически полностью отражает содержание входной программы. На этом уровне выполняются оптимизации, работающие на уровне конструкций входного языка. Представление VH-уровня может быть оттранслировано обратно в программу на исходном языке с минимальными потерями семантики.

На этом уровне WHIRL допускаются вложенные вызовы, а также операторы COMMA, RCOMMA (последовательность операторов), CSELECT (аналог оператора "?" языка C, реализуемый при помощи переходов). Допускаются также операторы, соответствующие агрегатам языка Fortran90: TRIPLET, ARRAYEXP, ARRSECTION, WHERE. На более низких уровнях WHIRL все эти конструкции отсутствуют.

4.1.2. Высокий уровень WHIRL

На этом уровне WHIRL (уровень H) не допускаются вложенные вызовы, так же как инструкции, вложенные посредством операторов COMMA и RCOMMA. Этот уровень WHIRL все еще может быть транслирован в программу на C или Fortran, однако соответствие исходной программе будет не слишком близким.

На этом уровне сохранены управляющие конструкции верхнего уровня, представляемые операторами DO_LOOP, DO_WHILE, WHILE_DO, IF, CAND, CIOR. Форма фортранных инструкций ввода-вывода сохранена в виде операторов IO и IO_ITEM. Форма индексации массивов присутствует в виде операторов ARRAY.

Над представлением уровня H выполняются межпроцедурные оптимизации, оптимизации гнезд циклов и предварительные оптимизации для последующих проходов.

4.1.3. Средний уровень WHIRL

На этом уровне (уровень M) начинают проявляться особенности архитектуры и набора команд целевой платформы. Предполагается, что псевдорегистры WHIRL имеют те же размеры, что и размеры физических регистров процессора. Могут присутствовать физические регистры, а также тип булевых значений, если процессор поддерживает предикатное выполнение.

Поток управления полностью отражается при помощи операторов TRUEBR, FALSEBR (условные переходы), GOTO (безусловный переход), COMPGOTO (вычисляемый переход). Ввод-вывод представлен в виде библиотечных вызовов. Операторы ARRAY заменены на адресные выражения.

Доступ к битовым полям выражен в виде операторов LDBITS, ILDBITS, которые включают чтение из памяти значения целого типа подходящего размера и выборку битового поля, определяемого смещением и размером. Аналогично, операторы STBITS, ISTBITS включают вставку битового поля в значение целого типа и запись этого значения в память. Далее эти операторы понижаются до EXTRACT_BITS и COMPOSE_BITS, которые выполняют только выборку и вставку битового поля из/в значение целого типа соответственно. Такая однородная форма представления позволяет во время оптимизаций выявлять общие подпоследовательности кода. На этом уровне выполняются глобальные скалярные оптимизации (WOPT).

4.1.4. Низкий уровень WHIRL

Оптимизирующий компонент WOPT выполняет два прохода выявления регистровых переменных (RVI — *Register Variable Identification*). Первый выполняется на уровне M; назначение уровня L — открыть возможности для второго прохода RVI. На уровне M используют операторы LDID, STID (чтение и запись по адресам, заданным как ссылки на таблицу символов). Это упрощает alias-анализ, т. е. анализ перекрытий объектов в памяти, который важен для выявления зависимостей по данным. На уровне L эти операторы трансформируются в операторы ILOAD, ISTORE, в которых адрес задан как база и смещение. В результате базовые адреса становятся доступными для RVI, а сами операторы ILOAD, ISTORE транслируются в машинные команды чтения и записи. Константы транслируются в последовательности команд, при помощи которых они могут быть сформированы на целевом процессоре. Вызовы CALL транслируются в операторы PICCALL (для платформ с разделяемыми библиотеками). Высокоуровневое представление вычисляемого оператора goto языка Фортран (COMPGOTO) транслируется в низкоуровневое (XGOTO), содержащее ссылку на таблицу переходов и выражение для вычисления адреса перехода.

4.1.5. Самый низкий уровень WHIRL

Это наиболее низкий уровень представления (уровень VL), который непосредственно транслируется в представление генератора кода, соответствующее последовательности машинных команд. На уровне VL существует строгое взаимно-однозначное соответствие между WHIRL и последовательностью машинных команд. Данный уровень предназначен только для генератора кода и для выполнения некоторых машинно-зависимых оптимизаций (*peephole optimizations*).

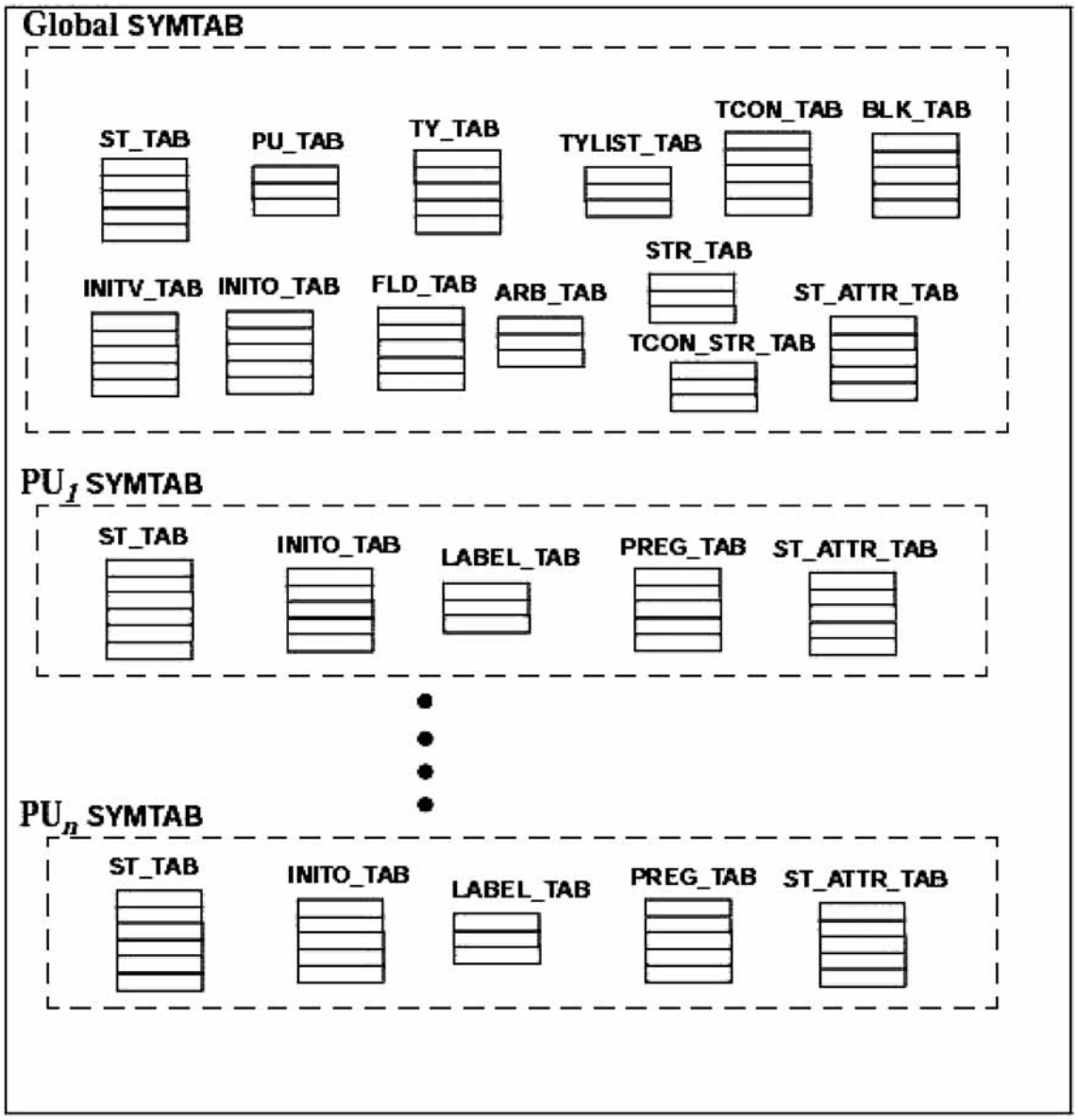
4.2. Компоненты WHIRL и его расширения

Представление WHIRL состоит из таблицы символов и древовидных структур, соответствующих программным единицам (PU), т. е. функциям и подпрограммам. Древовидные представления программных единиц строятся из узлов WHIRL. Проходы анализа и оптимизации Open64 могут создавать поверх этого базового представления дополнительные структуры, описывающие необходимые свойства, например, представление WHIRL/SSA или граф потока управления.

4.2.1. Структура таблицы символов

Таблица символов WHIRL состоит из набора таблиц, каждая из которых содержит однородные элементы информации. Таблица символов подразделяется на глобальную и локальные части. Глобальная часть содержит информацию, относящуюся к входному файлу в целом. Локальные части содержат данные о локальных переменных функций и подпрограмм.

На рисунке дана схема представления таблицы символов в том виде, в котором она создается компилятором переднего плана. Перечисленные далее три



Представление таблицы символов WHIRL

таблицы могут присутствовать как в глобальной (Global SYMTAB), так и в локальных частях (PU SYMTAB), соответствующих программным единицам (n — число локальных частей).

- ST_TAB — основной компонент таблицы символов. Содержит записи обо всех именованных объектах программы и константах, хранящихся в памяти.
- INITO_TAB — начальные значения инициализированных объектов программы. Включает ссылки на элементы таблицы INITV_TAB, содержащей значения компонентов, составляющих сложные объекты.
- ST_ATTR_TAB — атрибуты элементов таблицы ST_TAB, такие как имя секции ELF-файла или выделенный физический регистр, в котором содержится объект.

Следующие таблицы, которые могут содержаться только в глобальной части таблицы символов, содержат следующую информацию о программных единицах и типах данных входного модуля:

- PU_TAB — программные единицы, содержащиеся во входном файле;
- TY_TAB — типы данных, предопределенные и описанные в программе;
- FLD_TAB — поля структур и объединений;
- TYLIST_TAB — типы аргументов функций в программе;
- ARB_TAB — измерения массивов в программе;

- TCON_TAB — значения нецелочисленных констант (элементы, описывающие строковые константы, содержат ссылки на TCON_STR_TAB);
- BLK_TAB — блоки данных в программе, например, секции в объектном файле;
- INITV_TAB — начальные значения скалярных компонентов инициализированных объектов данных;
- STR_TAB — все текстовые строки, включая имена переменных, типов, меток и т. д.;
- TCON_STR_TAB — строковые литералы, описанные во входной программе.

Следующие две таблицы могут располагаться только в локальной части таблицы символов:

- LABEL_TAB содержит информацию, связанную с метками в программных единицах;
- PREG_TAB содержит информацию, ассоциированную с псевдорегистрами, используемыми в программных единицах.

4.2.2. Представление программных единиц

Представления программных единиц WHIRL являются строго древовидными (направленные ациклические графы не допускаются). Тип узла WHIRL определяется полем оператора. Вершиной дерева, представляющего программную единицу, является узел типа FUNC_ENTRY (функция).

Операторы WHIRL подразделяются на следующие три основные категории.

1. *Структурные управляющие конструкции*, такие как FUNC_ENTRY (функция), BLOCK (программный блок), REGION (область кода с одной точкой входа, имеющая ациклический граф потоков управления), DO_LOOP, DO_WHILE, WHILE_DO (циклы), IF (оператор if-then-else). Узлы этой категории, за исключением функций, блоков и регионов, существуют только на уровнях VH, H.

2. *Простые операторы с побочными эффектами*: управляющие операторы (GOTO, RETURN и т. п.), различные операторы вызова функций, операторы ASM (ассемблерные вставки), операторы ввода-вывода, прагмы, операторы предвыборки данных и др.

3. *Операции*, т. е. операторы без побочных эффектов, вычисляющие некоторый результат: арифметические, логические, побитовые операции, операции сдвига и вращения, операции преобразования типа, условные и др.

Перечисленные категории образуют строгую иерархию. Это означает, что структурная управляющая конструкция не может быть потомком узла более низкой категории, а простой оператор не может быть потомком выражения.

4.2.3. Основные элементы структуры узла

Структура узла зависит от его типа, который определяется полем оператора (**operator**). Два поля **desc** и **res** определяют типы данных, с которыми работает операция: **res** — тип результата, **desc** — общий тип всех операндов. Операция, которую требуется выполнить, полностью определяется совокупностью этих трех полей. Для определения операции не требуется содержимое полей, описывающих операнды.

Поддерживаемые типы данных включают в себя:

— тип булевых значений (0, 1);

— знаковые и беззнаковые целочисленные типы размером 8, 16, 32 и 64 бит;

— типы адреса размером 32 и 64 бит;

— типы данных с плавающей точкой размером 32, 64, 80 и 128 бит в представлении IEEE, а также 128-битные значения в формате SGI;

— комплексные типы данных с 32-битными, 64-битными и 128-битными компонентами;

— тип VOID;

— структурный тип;

— тип битового поля.

Потомки узла представляются полем **kid_count** (число потомков) и полями **kid[0]**, **kid[1]** и т. д., в которых содержатся указатели на узлы-потомки (например, операнды операции). В узле зарезервировано пространство для минимум двух потомков, но их может быть и больше.

Исключение составляет узел BLOCK, в котором число потомков не фиксировано, а сами потомки (операторы, составляющие блок) представлены двусвязным списком. Вместо полей **kid[0]**, **kid[1]** узел BLOCK имеет поля **first**, **last**, содержащие голову и хвост списка.

Потомками узла BLOCK являются узлы-операторы. Эти узлы имеют дополнительные поля **prev**, **next**, при помощи которых формируется двусвязный список. Эти узлы имеют также поле **linenum**, в котором упакована информация о входном файле, номере строки и позиции в строке исходного текста, где находится оператор.

Узлы, в зависимости от своего типа, могут иметь поля, отражающие определенную специфику. Например, узлы, которые описывают адреса в памяти, имеют поле **offset**, определяющее смещение относительно символа в таблице символов или относительно начала некоторого блока данных. В узлах других типов поле **offset** может иметь другое назначение.

Примеры листовых узлов WHIRL — узлы, описывающие ссылки на формальные параметры функции, константы, адреса в памяти, псевдорегистры или фиксированные физические регистры. Эти узлы не содержат ссылок на другие узлы WHIRL, но могут содержать ссылки на элементы таблицы символов.

4.2.4. Механизм аннотаций

Некоторым проходам компиляции могут требоваться дополнительные поля в узлах WHIRL. Для этих целей в Open64 предусмотрен механизм аннотаций, или отображений. В каждом узле существует поле, в котором хранится идентификатор узла (**map_id**). Значение этого поля служит индексом для обращения к таблице аннотаций.

Таблиц аннотаций на каждом проходе можно создать столько, сколько необходимо. Как правило, аннотации определенного вида требуются не для всех узлов, а только для некоторых подкатегорий. С этой целью узлы разделены на пять подкатегорий: структурные управляющие конструкции, операции чтения-записи, массивы (ARRAY), выражения, прочие операции. Механизм аннотаций использует "ленивый" алгоритм выделения памяти для аннотаций: как только данный вид аннотации впервые создается для некоторого узла данной подкатегории, выделяется пространство под аннотации для всех узлов этой подкатегории в данной программной единице.

4.2.5. SSA-представление

SSA-представление было включено в инфраструктуру Open64 для того, чтобы повысить точность информации о зависимостях и обеспечить сохранение корректности этой информации при трансформациях программы. Для того чтобы сохранить структуру узлов WHIRL и избежать переработки существующего кода, который работает с этим представлением, SSA-представление было реализовано "поверх" WHIRL как набор таблиц, ссылающихся на узлы WHIRL.

4.3. Преимущества и недостатки внутреннего представления Open64

Преимуществом внутреннего представления WHIRL, используемого компилятором Open64, является его однородность: на протяжении всего процесса компиляции (за исключением фазы генерации кода) используется один и тот же древовидный вид представления.

В компиляторе GCC используются три уровня представления — два древовидных (GENERIC и GIMPLE) и низкоуровневое линейное представление RTL. Из них высокоуровневое древовидное представление GENERIC служит лишь "общим знаменателем", к которому компиляторами переднего плана приводятся входные программы; все оптимизации выполняются на двух более низких уровнях. Наличие двух разных видов представления, древовидного и линейного, усложняет общую структуру компилятора. В частности, многие вспомогательные структуры (CFG, базовые блоки, данные о циклах) и операции над ними должны быть реализованы для обоих представлений.

В компиляторе LLVM на всем протяжении процесса компиляции используется одно линейное низкоуровневое представление, включающее высокоуровневую систему типов.

Важным принципом для инфраструктуры Open64 является постепенное понижение уровня представления и применение оптимизаций на максимально высоком уровне, на котором это возможно без потери качества. В частности, над представлением высокого уровня (H) выполняются оптимизации гнезд циклов, которые было бы затруднительно реализовать над низкоуровневым представлением, подобным тому, которое применяется в LLVM.

Недостатком представления Open64, по мнению авторов, является жесткое определение структур, описывающих элементы таблицы символов и узлы WHIRL, что может затруднять реализацию поддержки новых языковых средств. Например, тип данных **`TY_IDX`** (индекс элемента в таблице типов) представлен 32-битным целым, в котором закодированы биты, соответствующие квалификаторам **`restrict`**, **`volatile`**, **`const`**, при этом отсутствуют резервы для представления других возможных квалификаторов, например, квалификатора **`_Atomic`**, появившегося в новом стандарте языка C [9].

5. Структура описания целевых архитектур в Open64

В этом разделе рассматривается структура описания целевых архитектур в Open64 и дается ее сравнение с представлением аналогичных описаний в компиляторах GCC и LLVM.

5.1. Элементы описания архитектуры

Описание целевой архитектуры в Open64 представлено в виде файлов на языке C++, расположенных в нескольких подкаталогах исходных текстов компилятора. Основная часть описания находится в подкаталогах каталога **`osprey/common/targ_info.`** Здесь и далее *targ* обозначает имя архитектуры: **`ia64`**, **`loongson`**, **`MIPS`**, **`NVISA`**, **`ppc32`**, **`pr1`**, **`SL`** или **`x8664`**.

1. Файлы подкаталога **`proc/`***`targ`***`/*.cxx`** определяют имя архитектуры и имена поддерживаемых процессоров, описывают свойства различных процессоров, такие как наличие гнезд задержки, суперскалярность, наличие механизма аппаратного планирования. В них также содержатся описания команд различных процессоров с точки зрения планировщика, включая латентности и требуемые вычислительные ресурсы.

2. Файлы подкаталога **`isa/`***`targ`***`/*.cxx`** содержат описание системы команд (ISA) целевой архитектуры: список всех команд и псевдокоманд; группы инструкций по типу выполняемых функций (чтение, запись, пересылки, вызовы функций и др.). В них также описаны группы инструкций с одинаковыми типами кодировки (упаковки), группы регистров, а также задержки, не отрабатываемые аппаратно, которые требуется учитывать при генерации кода.

3. Файл **`abi/`***`targ`***`/abi_properties.cxx`** содержит описания ABI (*Application Binary Interface*), поддерживаемых целевой платформой: имена ABI; данные об использовании регистров для различных ABI (регистры, посредством которых передаются аргументы и возвращаются результаты функций, регистры, сохраняемые и не сохраняемые при вызовах функций, регистры, используемые как указатель стека и указатель кадра, и другие подобные им).

В перечисленных выше подкаталогах находятся файлы на языке C++, содержащие функции с описаниями свойств архитектуры. Эти функции выполняются на стадии сборки Open64 и генерируют файлы на языке C, содержащие декларации и функции, которые используются компилятором для получения информации о свойствах архитектуры.

Рассмотрим в качестве примера описание процессоров архитектуры x86 из файла **osprey/common/targ_info/proc/x8664/proc.cxx**:

```
int main ()
{
  PROC_Create( "x8664",
               "opteron",
               "barcelona",
               "orochi",
               "em64t",
               "core",
               "wolfdale",
               NULL );
return 0;
}
```

Здесь **PROC_Create** — функция, генерирующая файлы **targ_proc.h**, **targ_proc.c**, которые содержат декларации и функции, непосредственно используемые компилятором для доступа к информации о наборе поддерживаемых процессоров.

Другие части описания архитектур, рассматриваемые далее, представляют собой программы на языке C++, используемые непосредственно компилятором Open64. Эти программы реализуют определенные аспекты функциональных возможностей компилятора или предоставляют ему информацию о свойствах элементов целевой архитектуры.

4. Файл **osprey/common/util/***targ***/c_qwmultu.c** содержит информацию для вспомогательных утилит. В частности, информацию о поддержке операций над различными типами данных.

5. Файлы подкаталога **osprey/common/com/***targ* позволяют реализовать некоторые функции компиляции, зависящие от свойств инструментальной машины и опций командной строки, в частности, опций, задающих конкретный тип процессора, ABI и др. Например, файл **targ_const.cxx** содержит процедуры для вычисления константных выражений на инструментальной машине так, как они были бы вычислены на целевом процессоре. Файл **config_cache_targ.cxx** описывает свойства кэша для разных процессоров данной архитектуры. Файл **config_asm.h** содержит форматы для вывода элементов ассемблерного кода.

6. Файлы подкаталога **osprey/be/lno/***targ***/** содержат информацию для оптимизатора гнезд циклов, в частности, данные о латентностях некоторых операций.

7. Файлы подкаталога **osprey/be/be/***targ***/** задают настройки для драйвера компиляции нижнего уровня, в числе которых обработка ключей командной строки, задающих, например, ABI, ISA (архитектура), тип процессора.

8. Файлы подкаталога **osprey/be/cg/***targ***/** содержат реализацию методов, используемых генератором кода: процедуры генерации кода для стандартных операций, регистров; процедуры работы со стековыми регистрами; процедуры вывода элементов ассемблерного кода (переключение секций, начало и конец функции, типы перемещений, директивы отладочной информации и др.).

9. Файлы подкаталога **osprey/be/com/***targ***/*** содержат архитектурно-зависимые процедуры генератора кода, включая: процедуры трансляции операторов низкоуровневого WHIRL в команды целевой архитектуры; функции, описывающие возможность и целесообразность замены операций умножения и деления сериями более быстрых операций (сдвиг, сложение, вычитание); возможность использования непосредственных значений в качестве операндов различных операций и др.

5.2. Сравнение с компиляторами GCC и Clang/LLVM

В этом разделе дается сравнение методов описания целевых архитектур в исходных текстах компиляторов Open64, GCC и Clang/LLVM по ряду характеристик.

- *Структура описания*, а именно его распределение по файлам и каталогам. В случаях как LLVM, так и GCC описание каждой архитектуры содержится в одном каталоге. В исходных текстах компилятора Open64 описание архитектуры распределено по нескольким каталогам, что затрудняет изучение и разработку таких описаний.

Положительная черта Open64 и LLVM заключается в том, что в каждом файле содержится описание одной группы свойств (система команд, регистры, ABI и т. д.). В исходных текстах GCC некоторые файлы могут быть очень большими и содержать описание множества различных свойств целевой платформы.

- *Средства генерации.* Все три компилятора, GCC, LLVM и Open64, имеют средства генерации, позволяющие транслировать описания свойств архитектуры и процессоров, представленные в компактной и наглядной форме, в структуры данных и функции (на языках C или C++), при помощи которых компилятор осуществляет доступ к информации о свойствах архитектуры.

Средства генерации компиляторов GCC и LLVM включают специализированные декларативные языки и трансляторы с них. В случае Open64 использован более простой подход: файл описания архитектуры представляет собой исходный текст (на языке C++) программы, на этапе сборки компилятора генерирующей файлы на языке C. Полученные файлы затем компилируются и включаются в собираемый компилятор. Для сокращения и упрощения генерирующих

программ используют обращения к библиотечным функциям (подробнее см. разд. 5.1).

Наличие средств генерации значительно упрощает разработку описаний архитектур. Тем не менее далеко не всю информацию удается представить в виде описаний в терминах высокоуровневых языков, поддерживаемых генераторами. Во всех трех рассматриваемых компиляторах значительная часть информации о целевых архитектурах представлена в виде программ на языке реализации компилятора (C, C++).

- *Гибкость средств генерации.* Наиболее развитые языковые средства для описания архитектур реализованы в компиляторе LLVM. Сущности, из которых состоит описание архитектуры (команда, система команд, регистр, ABI и др.), определяются в идеологии LLVM как экземпляры классов. Предоставляются стандартные классы для каждого вида сущностей и существует возможность создавать производные классы, включающие дополнительные поля, с учетом потребностей генерации и оптимизации кода для данной архитектуры.
- *Полнота.* Под полнотой здесь понимается то, насколько предоставляемые описания охватывают потребности процесса компиляции в целом. В наибольшей степени это качество представлено в LLVM, где описание системы команд включает информацию, достаточную не только для генерации ассемблерного кода (как в Open64 и GCC), но и для генерации объектного кода, дизассемблирования, а также JIT (*Just-In-Time*) компиляции.
- *Избыточность.* Под избыточностью понимается необходимость задавать информацию о некотором свойстве целевой платформы в разных видах для разных компонентов компилятора. Это может упрощать написание компонентов компилятора, но усложняет структуру описаний целевых платформ.

Свойство избыточности отчасти присуще описаниям архитектур в GCC (в части описания регистров, процессоров), но в наибольшей степени оно присутствует в Open64. Например, информация об ABI задается в нескольких частях описания (см. пп. 3, 5, 7 в разд. 5.1). Это же относится к информации о латентностях команд.

Таким образом, можно видеть, что методы описания целевых архитектур в Open64 по многим характеристикам уступают методам, применяемым в GCC и LLVM. Недостаточная развитость средств описания архитектур является одной из причин того, что число целевых процессоров, поддерживаемых этим компилятором, относительно невелико.

Заключение

Существенным преимуществом компилятора Open64 по сравнению с компиляторами GCC и LLVM являются развитые средства оптимизации гнезд циклов, включающие слияние, расщепление, инвертирование, развертку, предвыборку данных и другие трансформации. Этот вид оптимизации автоматически включается при задании опции `-O3` и проводится с учетом особенностей системы команд, а также структуры и размеров кэшей.

Интерес представляет также возможность выполнять трансформации гнезд циклов на уровне исходного языка, что позволяет использовать этот вид оптимизаций для целевых платформ, не поддерживаемых Open64.

В то же время Open64 значительно отстает от GCC и LLVM по динамике развития. Это выражается, в частности, в поддержке существенно меньшего числа целевых платформ и входных языков, в отсутствии поддержки новых языковых средств и стандартов — C11 [9] и C++11 [10].

Одной из причин такого положения может являться недостаточность оперативной поддержки и слабая документированность проекта, что не способствует привлечению к нему новых разработчиков. Существует довольно подробное описание внутреннего представления программ [7], однако документация по проекту в целом и по средствам описания целевых платформ явно недостаточна. В исходных текстах генератора кода Open64 можно найти довольно подробные комментарии, но в других частях исходных текстов компилятора объем комментариев явно недостаточен.

Список литературы

1. **Open64** Home. URL: http://www.open64.net/
2. **Using** the x86 Open64 Compiler Suite. URL: http://developer.amd.com/wordpress/media/2012/10/open64.pdf
3. **GCC,** the GNU Compiler Collection. URL: http://gcc.gnu.org.
4. **Shuchang Z., Ying L., Fang L., Le Y., Lei H., Shuai L., Chunhui M., Zhitao G., Ruiqi L.** Open64 on MIPS: porting and enhancing Open64 for Loongson II. URL: http://www.capsl.udel.edu/conferences/open64/2009/Papers/102-Open64onMIPS2.pdf
5. **OpenUH** Research Compiler. URL: http://www2.cs.uh.edu/~openuh.
6. **Вьюкова Н. И., Галатенко В. А., Самборский С. В.** LLVM как инфраструктура разработки компиляторов для встроенных систем // Программная инженерия. 2013. № 6. С. 2—10.
7. **Optimization** opportunities in GRAPHITE — GCCSummit'09 URL: http://gcc.gnu.org/wiki/Graphite?action=AttachFile&do=view&target=grosser09-optimization-opportunities-in-graphite_paper.pdf
8. **Open64** Compiler WHIRL Intermediate Representation. 2007. URL: http://www.mcs.anl.gov/OpenAD/open64A.pdf
9. **ISO/IEC** 9899:2011 — Information technology — Programming Languages — C.
10. **ISO/IEC** 14882:2011 — Information technology — Programming Languages — C++.

УДК 004.42:62-519

В. С. Заборовский[1, 2], д-р техн. наук, проф., зав. каф., e-mail: vlad@neva.ru,
М. Ю. Гук[1], нач. отд.,
А. С. Ильяшенко[1], науч. сотр.,
В. А. Мулюха[1], канд. техн. наук, ст. науч. сотр.,
А. В. Силиненко[1, 2], канд. техн. наук, доц.,
К. С. Селезнёв[1], вед. программист,
[1]ЦНИИ РТК, г. Санкт-петербург,
[2]СПбГПУ

Программное обеспечение для отработки алгоритмов сетевого интерактивного управления роботами с борта МКС

Рассматриваются вопросы создания программного обеспечения для системы интерактивного удаленного управления динамическим объектом с использованием механизма силомоментного очувствления. Предложены новые модели и алгоритмы управления, которые позволяют расширить возможности применения сетевых технологий для приложений, критичных к задержкам в каналах связи. Создана программная среда, предоставляющая широкие возможности для отработки алгоритмов управления объектами в реальном масштабе времени. Анализируется алгоритм динамической настройки параметров модели, используемой для формирования силомоментных воздействий рукоятки джойстика на оператора. Рассматривается архитектура программного обеспечения, используемого для очувствления задержек в канале передачи.

Ключевые слова: *двухконтурное интерактивное удаленное управление, силомоментное очувствление, динамическая адаптация*

Введение

Существующие компьютерные сети, построенные на базе протокола TCP/IP, функционируют используя каскадное соединение различных сегментов канальной инфраструктуры, что позволяет формировать виртуальный транспортный канал, соединяющий источник и приемник данных. Виртуальный канал — это ключевая абстракция, которая положена в основу многих современных информационных сервисов. Стандартизация транспортных протоколов (TCP, UDP) и применение алгоритмов адаптации пропускной способности виртуальных соединений (протокол TCP) к текущему состоянию сетевой среды, основанных на механизме повторной передачи потерянных или задержанных пакетов, позволяют обеспечить высокий уровень надежности доставки информации в современных компьютерных сетях. Однако использование существующих механизмов адаптации виртуальных каналов вносит случайные задержки в процесс доставки пакетов. По этой причине при использовании информационных приложений к сетевой инфраструктуре предъявляется ряд требований, а именно: возможность организации виртуального канала между всеми узлами сети; существование максимального значения RTT (*round trip time*) для всех допустимых в данной сети виртуальных каналов; малое значение до-

пустимой вероятности потери IP-пакетов, образующих конкретное виртуальное соединение.

К сожалению, по мере распространения сетевых технологий в различные отрасли промышленности и научные исследования, в частности, создание мобильных, беспроводных или сенсорных сетевых структур, одно или даже все из отмеченных выше требований нарушаются. Как следствие, разработка программного обеспечения для новых информационных приложений, которые используют в специальных условиях или технических системах нового класса, является актуальной инженерной задачей.

В данной статье рассматриваются вопросы создания программного обеспечения для сетевой системы интерактивного управления динамическим объектом — напланетным роботом с борта орбитальной космической станции с использованием механизма очувствления, который реализован с помощью силомоментной обратной связи. Предложены новые модели и алгоритмы управления, применение которых позволяет расширить возможности сетевых технологий для разработки приложений, критичных к задержкам в каналах связи. Созданная программная среда построена на "open source"-решениях, которые снижают стоимость разработки и предоставляют широкие возможности для отработки алгоритмов управления объектами в реальном времени. В работе приведены результаты анализа алгоритма динамической настройки параметров модели, используемой для формирования силомоментных воздействий рукоятки ЗМ — задающего манипулятора (или джойстика) на оператора. Рассматривается архитектура программного обеспечения, используемого для очувствления задержек в канале передачи управляющих воздействий.

Проводимые исследования выполняются в рамках программы космического эксперимента (КЭ) "Контур-2" [1], который проводится специалистами ЦНИИ РТК и кафедры "Телематика" СПбГПУ совместно с Институтом мехатроники и робототехники Германского аэрокосмического агентства (DLR-RMC) на Российском сегменте Международной космической станции (РС МКС).

Сетевые технологии в космическом эксперименте "Контур-2"

Создание систем управления на базе сетевых технологий открывает новый этап в развитии как теории, так и практики управления. На этом этапе важную роль начинают играть такие аспекты создания программного обеспечения для систем управления, в рамках которых широко используются принципы самоорганизации и адаптации. Наиболее известными примерами успешного применения сетевых технологий для промышленных и научных приложений являются следующие проекты: по созданию нового поколения систем автомобильной телематики, например Google driverless car; по проведению хирургических операций с помощью дистанционно управляемых роботов, например, "четырехрукого" робота-хирурга "da Vinci"; по использованию антропоморфных роботов на борту МКС, например, Justin (DLR-RMC), Robonaut (NASA). Особые условия для реализации принципов адаптации возникают в тех случаях, когда в контуре управления сетевым объектом находится человек-оператор. Такой оператор должен иметь средства оперативной идентификации параметров среды передачи данных и состояния управляемого объекта, чтобы в режиме реального времени оценивать результаты своей деятельности, основываясь на объективной информации, получаемой по каналам связи. Одним перспективным направлением использования человека-оператора в сетевом контуре управления распределенных динамических объектов является космонавтика.

В рамках Государственной программы Российской Федерации "Космическая деятельность России на 2013—2020 годы" к приоритетным областям исследований отнесены различные направления применения робототехнических систем для сервисных операций на борту орбитальных станций и освоения поверхностей планет Солнечной системы. Поэтому одной из целей КЭ "Контур-2" является отработка технологий удаленного управления роботами применительно к ситуациям, когда оператор находится в условиях микрогравитации на борту МКС, а для передач команд управления напланетным роботом используют сетевые сегменты с различной канальной структурой. Схема космического эксперимента приведена на рис. 1, см. третью сторону обложки.

Особенностью КЭ "Контур 2", которая влияет на выбор архитектуры системы программного обеспечения, является тот факт, что объекты сетевого управления представлены двумя типами роботов, функционирующих в различных режимах, но управляемых с помощью одного устройства — задающего манипулятора. Это устройство оснащено электроприводами, формирующими регулируемые усилия (моменты) по двум ортогональным осям рукоятки управления. Для управления первым типом роботов используется синхронный канал связи, задержка в котором не превышает 15 мс, дисперсия задержки (джиттера) не более 10 % от значения параметра RTT. В этом случае оператор может использовать режим билатерáльного управления роботом [2, 3]. Такой режим основан на технологии телеприсутствия, которая позволяет в реальном масштабе времени "очувствлять" результаты выполнения целевых операций, анализируя для этого данные о значениях текущих координат или моментах, которые передаются в формате IP-пакетов (22 байта полезной нагрузки) с частотой 500 пакетов в секунду.

Для управления вторым типом роботов используется виртуальный канал, проходящий через сегмент сети Интернет. Неотъемлемым свойством такого канала являются высокие значения RTT и дисперсии RTT (рис. 2), что снижает эффективность использования режима телеприсутствия, но тем не менее позволяет оператору с помощью обратного усилия на ру-

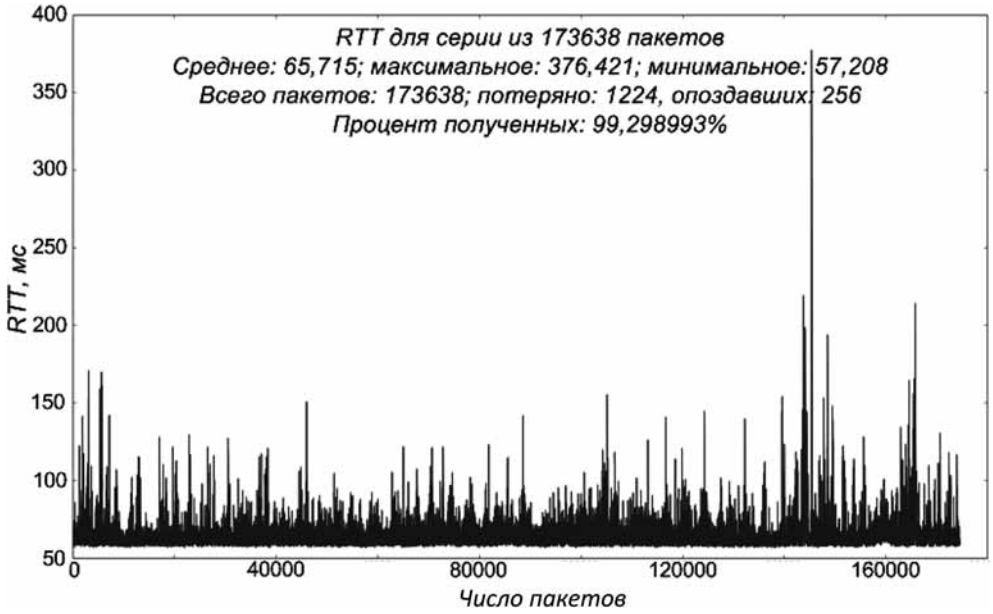


Рис. 2. Изменения значений параметра RTT в виртуальном канале управления, проходящем через сеть Интернет

коятке ЗМ оценивать свойства сетевой среды, в которой функционирует исполнительный элемент робота.

Такое повышение информативности управления за счет эффекта силомоментного очувствления задержек позволяет оператору корректировать скорость и направления перемещения робота, а с помощью воздействия на рукоятку джойстика — ощущать влияние сетевой среды и формировать оценку задержки данных в канале связи и сетевых сегментах, которые включают в себя:

- S-band-радиоканал, обеспечивающий передачу данных между бортом МКС и наземной приемной станцией, расположенной на территории Германии;
- цифровую синхронную линию связи между наземной приемной станцией и объектом управления (наземный сегмент № 1 в Германии);
- публичную сеть (Интернет), связывающую германский наземный сегмент № 1 с российским наземным сегментом № 2.

Особенности исследуемой сетевой среды характеризуют следующие факторы:

- S-band-радиоканал является асимметричным (256 кбит/с от сегмента № 1 к МКС и 4 Мбит/с от МКС к сегменту № 1);
- радиоканал обеспечивает дуплексную передачу данных только в течение 9—10 мин, когда пилотируемая орбитальная станция находится в зоне устойчивого приема;
- в сегменте № 2 требуется использование режима компенсации задержек, неупорядоченной доставки или потерь пакетов при прохождении данных через сеть Интернет, что позволяет реализовать интерактивное управление, учитывающее пространственные и временные ограничения.

Перечисленные особенности существенно повышают требования к архитектуре и составу системного и прикладного программного обеспечения ЗМ.

Организация интерактивного управления роботом с помощью двухконтурной системы

Учитывая особенности используемой сетевой среды для организации интерактивного управления, процессы информационного взаимодействия ЗМ и робота предлагается декомпозировать на два класса — процессы локальной отработки команд управления в режиме жесткого реального времени и процессы доставки команд управления через сетевую инфраструктуру с использованием модели взаимодействия типа "точка-точка" и протоколов из стека TCP/IP. Организованная таким образом система обеспечивает взаимодействие двух следующих контуров управления (рис. 3, см. третью сторону обложки):

1) локального контура, в котором программный модуль "контроллер ЗМ" (КЗМ) обеспечивает циклический опрос текущих координат ЗМ (вектор P), вычисление и отправку в ЗМ вектора силы (F), зависящего от текущего положения и скорости перемещения рукоятки, а также информации обратной связи (T'), получаемой от объекта управления (ОУ);

2) сетевого контура, в котором программные компоненты используют для организации процесса передачи векторов управления (C) и данных телеметрии (T) между КЗМ и ОУ.

Основой сетевого контура управления является программный модуль Транспортер, состоящий из сетевых модулей ЗМ и ОУ (СМЗМ и СМОУ соответственно, см. рис. 3), связанных виртуальным транспортным каналом, который построен на базе протокола UDP [6—8].

С конечными системами (КЗМ и ОУ) сетевые модули связаны через модули адаптации (МА) к свойствам среды передачи (МАЗМ и МАОУ, соответственно).

Очевидно, что для обеспечения плавности управления силового воздействия на рукоятку ЗМ частота циклов КЗМ должна быть достаточно высокой. Связку КЗМ-ЗМ при этом можно рассматривать как (почти) аналоговую систему, что упрощает математическое описание системы и анализ ее устойчивости. Также как аналоговую систему желательно рассматривать и ОУ со своим локальным контуром управления, обеспечивающим его движение под управлением приходящего задания (вектор C'). Обе эти конечные "почти аналоговые" системы объединяют в глобальный контур через сеть с пакетной коммутацией. Темп посылки пакетов по сети может быть значительно ниже частоты циклов КЗМ, поэтому связь по сети следует рассматривать как дискретную (по времени), причем вносящую существенное запаздывание.

Программный модуль Транспортер основан на протоколе UDP, обеспечивающем скорейшую, но не гарантированную и не упорядоченную доставку сообщений. Транспортер предназначен для обеспечения изохронной связи локальных контроллеров: отсчеты вектора управления, равномерно получаемые от локального контроллера ЗМ, должны также равномерно (но, естественно, с задержкой) передаваться локальному контроллеру ОУ. Аналогично во встречном направлении должны передаваться отсчеты вектора телеметрии. Таким образом, организуется "цифровая" связь "аналоговых" конечных систем с частотой дискретизации, равной частоте отправки сообщений. Однако задержка доставки, вносимая сетью, не является постоянной: часть отсчетов может теряться, порядок доставки может нарушаться. Задачей принимающей части модуля Транспортер является отслеживание правильного порядка и стимулирование своего модуля адаптации на восполнение пропущенных отсчетов. Транспортер также определяет текущее состояние среды передачи: задержку доставки (среднюю и вариации); вероятность пропадания сообщения.

Кроме изохронной доставки потоков отсчетов векторов, Транспортер обеспечивает и доставку асинхронных сообщений о событиях, которые имеют отношение к процессу телеуправления. Примером таких событий является нажатие кнопок, имеющихся на рукоятке ЗМ. Эти кнопки могут использоваться для управления режимом работы ОУ или выполнения ими каких-либо действий. Асинхронность события означает, что оно возникает эпизодически в произвольный момент времени. Однако оно должно гарантировано дойти до получателя, сохраняя привязку по времени к передаваемому изохронному потоку отсчетов.

В условиях управления с борта пилотируемой орбитальной станции в канале связи могут возникать случайные задержки, которые приводят к тому, что управляемое устройство может иметь неполную информацию о текущем состоянии управляющего устройства. С учетом этих обстоятельств в контур управления добавлены элементы, которые называют модулями адаптации. Задачами модулей адаптации является восполнение (при необходимости) пропущенных отсчетов, а также трансляция векторов управления и телеметрии (C, T) в сообщения, передаваемые по сети (CN, NT), с учетом текущих параметров среды передачи (и обратные преобразования на противоположной стороне). Модули адаптации предназначены для решения задачи обеспечения управляемого устройства дополнительной информацией. Такая информация в случае недостаточности данных позволит спрогнозировать и предсказать поведение оператора, не допустить перерывов в процессе передачи управляющих сигналов, и тем самым обеспечить плавность управления роботом.

В простейшем случае трансляция прозрачна: в сообщениях передаются непосредственно текущие (на момент отправки) значения соответствующих векторов (отсчеты), и на принимающей стороне эти векторы передаются в локальный контур управления. Более сложный вариант поведения модуля адаптации может быть основан на формировании функций, зависящих как от векторов управления и телеметрии (текущих значений, предыстории), так и от текущих параметров среды передачи данных.

Для визуализации модели робота, а также для "осциллографирования" процессов управления в целях оценки качества Транспортер передает на компьютер оператора потоки отсчетов векторов управления (текущая позиция ЗМ), силы, прикладываемой к рукоятке ЗМ, и телеметрии (информация от робота, отстающая на время доставки). Для отладочных целей в сетевые модули введена возможность отправки сообщений трассировки на внешний приемник (на рис. 3 обозначен как TeleSniffer).

Реализация модулей адаптации

По результатам анализа ряда работ по соответствующей тематике [1—4] была построена схема контура управления, позволяющего решить поставленную задачу. Данная схема изображена на рис. 4.

На схеме можно видеть разделение контура на три основные части:

- контур управляющего устройства (на схеме слева), выходом которого является вектор управления C, отображающий текущее положение рукоятки манипулятора p_m. Перемещение рукоятки определяется силой f_m, складывающейся из силы воздействия оператора f_h, силы f, формируемой регулятором ЗМ, и силы f_e', отображающей силу воздействия окружающей среды f_e на робота. Входом p регулятора ЗМ является рассогласование текущей позиции ЗМ p_m с отображением текущей позиции робота p_s. Параметры регулятора — вектор {A} — могут изменяться динамически.
- средства коммуникации (Транспортер), вносящие задержки доставки, и модули адаптации, парирующие действие этих задержек. В общем случае задержки в прямом (T_f) и обратном (T_b) каналах могут быть различными.

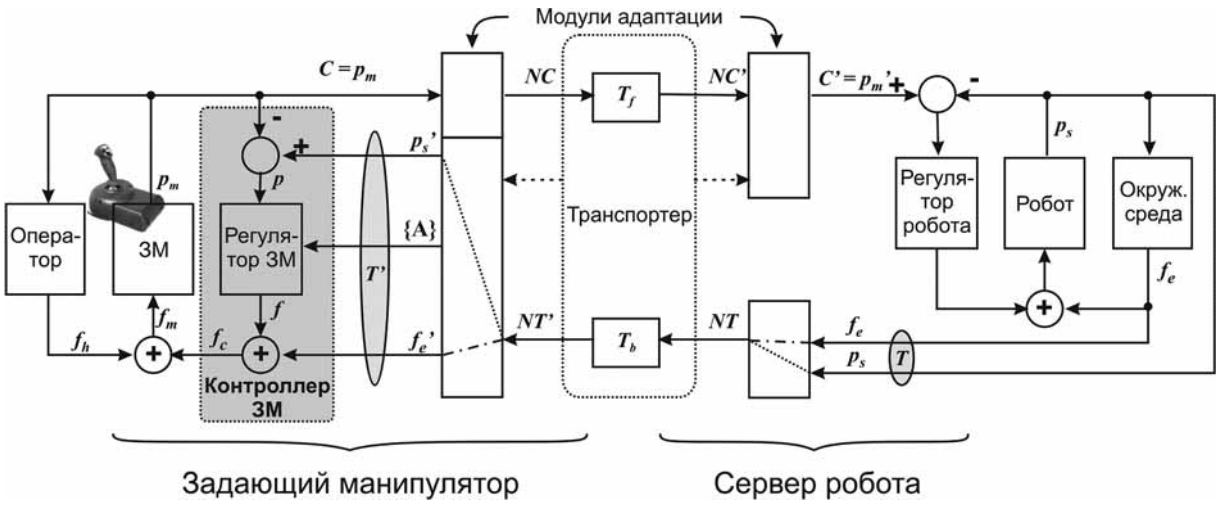


Рис. 4. Схема контура управления

• контур управляемого устройства (на схеме справа), вырабатывающего воздействие на робота по рассогласованию его текущей позиции p_s и отображения p_m' позиции задающего манипулятора. На робота также может воздействовать окружающая среда с силой f_e. В качестве обратной связи может использоваться вектор текущей позиции робота p_s и, при наличии соответствующих датчиков, вектор силы воздействия окружающей среды f_e.

В рамках исследования предлагается провести сравнение четырех способов организации удаленного управления с использованием модулей адаптации в целях изучения эффективности предлагаемых далее подходов:

1) управление без модулей адаптации;

2) управление с модулем адаптации принимаемых сигналов на стороне задающего манипулятора;

3) управление с модулем адаптации принимаемых сигналов на обеих сторонах;

4) управление с симметричными модулями адаптации, осуществляющими преобразования как принимаемых, так и отправляемых сигналов.

Подход 1 подразумевает процесс управления без попытки какого-либо прогнозирования поведения оператора или робота в процессе управления, а также учет влияния управляющих сигналов только в момент их непосредственного поступления. Подходы 2 и 3 реализуют стандартные способы прогнозирования поведения оператора и устройств на основе методов экстраполяции исторических данных, которые будут накапливаться по мере проведения эксперимента в памяти объекта управления. Для этих способов будет проведено исследование пригодности моделей прогнозирования на основе построения "скользящей средней" (Moving Average) или кривых более высоких порядков. Эти кривые позволят выявить основное направление движения джойстика или робота и осуществить необходимое прогнозирование дальнейшего движения в случае, когда информация от ЗМ не дошла до ОУ по каким-либо причинам. Подход 4 предполагает использование модулей адаптации не только при приеме управляющих сигналов, но и непосредственно при подготовке их к передаче по сети. Такой способ позволяет при передаче управляющих сигналов включить в передаваемое сообщение информацию о предыстории процесса (которая известна ЗМ) так, чтобы при выработке сигнала в модуле адаптации принимающего устройства полученные данные использовались для формирования воздействий, обеспечивающих устойчивость и подавление автоколебаний в замкнутой системе управления.

Модель силомоментного очувствления процесса управления

Организация процесса удаленного управления с силомоментным очувствлением требует того, чтобы в процессе управления оператор имел возможность "ощущать" текущее состояние как робота, так и сети передачи данных. Поэтому для описания конфигурационного пространства системы управления предлагается использовать модель виртуальной пружины, закрепленной одним концом в основание джойстика, начальное положение которой совпадает с текущим положением управляемого робота. В начальном положении робот никаким образом не воздействует на ЗМ. Однако если оператор начнет осуществлять управление и выводить джойстик из начального положения, то чем дальше будет отведен джойстик от начального положения, тем большие усилия операция "растяжения виртуальной пружины" потребует от оператора. На рис. 4 представлена схема, реализующая алгоритм расчета силомоментного воздействия, которое ощущает оператор в процессе управления. Сила упругости виртуальной пружины вычисляется по результатам анализа текущего положения джойстика с учетом пришедшей информации о положении робота, которая рассчитывается по формуле (1)

$$p = p_s' - p_m, \qquad (1)$$

где p_s' — вектор координат робота, который на данный момент доступен для обработки в модуле адаптации джойстика; p_m —текущее положение манипулятора. Расчет силового воздействия, ощущаемого оператором, проводится по формуле (2):

$$f(t) = A_0 \int_0^t p(\tau)d\tau + A_1 p + A_2 \dot{p} + A_3 \ddot{p}, \qquad (2)$$

где A_0 — астатизм системы, A_1 — жесткость виртуальной пружины, A_2 — вязкость среды, в которой происходит управление, A_3 — виртуальная масса рукоятки.

В процессе управления стартовое положение виртуальной пружины будет изменяться так, чтобы совпасть с текущим положением координат робота. Эти изменения оператор сможет ощутить в виде ослабевания силового давления виртуальной пружины и, как следствие, более легкого продвижения ЗМ в заданном направлении. Предложенная модель позволяет организовать процесс управления так, чтобы информация о состоянии канала связи вносила коррективы в процесс очувствления. Другими словами, жесткость виртуальной пружины должна увеличиваться не только в зависимости от расхождения положения робота и ЗМ, но и корректироваться с учетом задержки данных в канале связи, увеличивая таким образом инерционность сетевого контура управления.

Для реализации такого режима взаимодействия на управляющем устройстве потребуется обрабатывать дополнительный объем информации о состоянии канала связи, в качестве которой в ЗМ используют упорядоченную по времени последовательность отсчетов параметра задержки RTT и долю потерянных сетевых пакетов (ДПП) [6—8]. Таким образом, жесткость виртуальной пружины и скорость перемещения исполнительного органа робота будут изменяться пропорционально усредненным значениям RTT и ДПП, рассчитанным по формуле (3) за период времени, который соизмерим с постоянной времени всей замкнутой системы управления:

$$f(t) = (A_0 \int_0^t p(\tau)d\tau + A_1 p + A_2 \dot{p} + A_3 \ddot{p}) + \\ + A_4 \cdot \mathrm{RTT} \cdot \mathrm{ДПП} \frac{p}{\|p\|}, \qquad (3)$$

где A_4 — коэффициент влияния среды передачи данных на силомоментное очувствление. В результате при увеличении задержек силовое воздействие на рукоятку ЗМ не позволит оператору осуществлять быстрое изменение его положения. Это обстоятельство хотя и уменьшит скорость перемещения робота, но позволит оператору корректировать результаты отработки операций, анализируя данные об управляющих воздействиях, несмотря на то что эти данные станут доступны оператору с задержкой.

Особенности реализации

Рассмотренная организация системы телеуправления предназначена для наземной отработки алгоритмов и отладки программного обеспечения научной аппаратуры для космического эксперимента "Контур 2". На стенде параметры коммуникационной системы и программного модуля Транспортер [4] модифицированы с учетом ограничений, накладываемых S-band-каналом связи, пропускная способность которого оценивается исходя из необходимости выполнения условий телеприсутствия, что требует передавать каждые 2 мс IP-пакеты, содержащие 22 байта прикладных данных. Эти требования позволят модифицировать формат сообщений и учесть свойства, отражающие специфику сетевых процессов [5—7] в целях сокращения разрядности поля нумерации пакетов и меток времени с помощью разделения номеров передаваемых пакетов на четные и нечетные. Четные пакеты используют для передачи изохронного трафика (5 величин в 32-битном формате с плавающей точкой), нечетные — для передачи асинхронных сообщений и служебной информации, обеспечивающей определение параметров канала с реализацией сетевых протоколов и использованием "облегченного" стека LwIP.

Согласно требованиям билатерального управления, программное обеспечение двухконтурной системы телеуправления является симметричным, а именно структура программных модулей на стороне ЗМ и робота совпадают. Локальные контроллеры как ЗМ, так и робота реализованы в виде ПИД-регуляторов и отличаются друг от друга лишь параметрами настройки. На рис. 5 приведена структура разработанного программного обеспечения ЗМ, для реализации которого используется 32-разрядный ARM микроконтроллер.

Программный модуль контроллера ЗМ обеспечивает чтение текущих координат и состояния кнопок джойстика, а также формирование и подачу команд управления силовым воздействием. Модуль реализует функции ПИД-регулятора, для которого входное значение и коэффициенты в законе управления формируются программным модулем адаптации.

Модуль Транспортер осуществляет регулярный прием и передачу данных из сети, обращаясь к модулю адаптации для преобразования векторов управления и телеметрии, а также обработки текущей информации о задержках в канале передачи данных.

Модуль адаптации осуществляет преобразования, обеспечения устойчивости замкнутой системы и показатели качества управления. Системное программное обеспечение ARM микроконтроллера формирует среду исполнения и включает в себя следующие компоненты: USB-host, LwIP, Free RTOS, а также набор драйверов для настройки и использования сетевых интерфейсов.

Модуль Free RTOS используется для обеспечения совместного выполнения задач перечисленными далее модулями (см. рис 4, 5):

- сервер управления роботом, принимающий команды от ПКУ и формирующий встречный поток

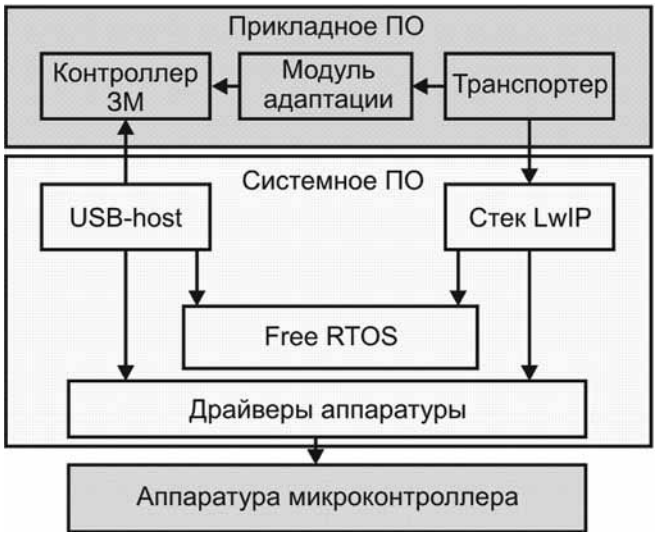


Рис. 5. Структура программного обеспечения задающего манипулятора

данных для управления 3D-моделью робота и регистрации (режим осциллографирования) параметров процессов управления;

- USB-host, обеспечивающий регулярный опрос координат джойстика и подачу команд управления силой;
- стек LwIP, осуществляющий первичную обработку IP-пакетов (контрольные суммы, подсчет статистики и пр.) и постановку их во входную очередь программного модуля Транспортер;
- Транспортер, обеспечивающий двусторонний обмен изохронными пакетами и асинхронными сообщениями.

Модуль Транспортер активизируется с удвоенной частотой обмена данными, что позволяет вызывать модуль адаптации, попеременно передавая ему на обработку то вектор телеметрии, полученный от робота, то вектор управления, полученный от контроллера джойстика. Результат работы модуля адаптации, соответственно, передается контроллеру джойстика или отправляется по сети объекту управления. Если в нужный момент времени во входной очереди сообщений нет, то в этом случае модуль адаптации формирует управляющее воздействие на основе предсказанного с использованием формулы (3) вектора значения жесткости виртуальной пружины и скорости перемещения исполнительного органа робота.

Заключение

Описана структура программно-алгоритмического обеспечения системы удаленного силомоментного управления роботом, находящимся на поверхности Земли, с борта пилотируемой орбитальной станции. Особенностью предложенного решения является использование двухконтурной системы, в которой "почти аналоговые" объекты (робот и задающий манипулятор со своими контурами управления) связаны сетевым каналом с пакетной коммутацией, вносящим переменную дискретность и существенные задержки доставки информации. Применение силомоментного очувствления позволяет оператору ощущать не только состояние управляемого устройства, но и получить информацию о канале передачи данных, что сигнализирует о возможном небезопасном дальнейшем управлении. Предложен метод управления и архитектура программного обеспечения для его реализации. Учитывая особенности рассматриваемой задачи, в качестве операционной системы для сетевых узлов выбрана свободно распространяемая система реального времени Free RTOS, позволяющая обеспечить высокую частоту циклов в локальных контурах управления и обеспечить сетевое интерактивное взаимодействие оператора с роботом. Программное обеспечение системы интерактивного телеуправления используется в билатеральном режиме как для задающего манипулятора, так и для объекта управления, что позволяет эффективно отрабатывать алгоритмы формирования сигналов силомоментной обратной связи для различных вариантов организации сетевой инфраструктуры.

Настоящая работа подготовлена по материалам договора № 13.G25.31.0026 между ОАО "РКК "Энергия" и Минобрнауки России, выполняемого с участием СПбГПУ и гранта РФФИ № 13-07-12106.

Список литературы

1. **Заборовский В. С., Кондратьев А. С., Силиненко А. В., Мулюха В. А., Ильяшенко А. С., Филиппов М. С.** Удаленное управление робототехническими объектами в космических экспериментах серии "Контур" // Научно-технические ведомости СПбГПУ. Информатика. Телекоммуникации. Управление. 2012. № 6 (162). С. 23—32.

2. **Artigas J., Jee-Hwan R., Preusche C. and Hirzinger G.** Network representation and passivity of delayed teleoperation systems // Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference. 25—30 September, 2011. San Francisco, CA. IEEE Computer Society, 2011. P. 177—183.

3. **Niemeyer G., Slotine J.-J. E.** Telemani pulation with time delays // Internacional Journal of Robotics Research. 2004. № 9 (23). P. 873—890.

4. **Гук М. Ю., Селезнев К. С., Балицкий В. И., Колесник А. М.** Реализация алгоритмов взаимодействия с удаленными робототехническими объектами через ненадежные каналы связи. Свидетельство о государственной регистрации программ для ЭВМ № 2012619599.

5. **Zaborovsky V., Lukashin A., Kupreenko S. and Mulukha V.** Dynamic Access Control in Cloud Services // International Transactions on Systems Science and Applications. 2011. Vol. 7, No. 3/4. P. 264—277.

6. **Заяц О. И. Заборовский В. С., Мулюха В. А., Вербенко А. С.** Управление пакетными коммутациями в телематических устройствах с ограниченным буфером при использовании абсолютного приоритета и вероятностного выталкивающего механизма. Часть 1 // Программная инженерия. 2012. № 2. С. 22—28.

7. **Заяц О. И. Заборовский В. С., Мулюха В. А., Вербенко А. С.** Управление пакетными коммутациями в телематических устройствах с ограниченным буфером при использовании абсолютного приоритета и вероятностного выталкивающего механизма. Часть 2 // Программная инженерия. 2012. № 3. С. 21—29.

8. **Zaborovsky V., Zayats O. and Mulukha V.** Priority Queueing With Finite Buffer Size and Randomized Push-out Mechanism // Proceedings of The Ninth International Conference on Networks (ICN 2010). 11—16 April, 2010. Menuires, The Three Valleys, French Alps. IEEE Computer Society, 2010. P. 316—320.

УДК 004.728.5 + 004.451.87 + 004.624

А. А. Санников, аспирант,
О. Ю. Богоявленская, канд. техн. наук, доц.,
Ю. А. Богоявленский, канд. техн. наук, доц., зав. каф., e-mail: ybgv@cs.karelia.ru,
ФГБОУ ВПО "Петрозаводский государственный университет"

Система анализа поведения и мониторинга производительности соединений транспортного уровня Интернет

Описана система, позволяющая проводить мониторинг активности сетевого узла на уровне ядра операционной системы Linux. Рассмотрены архитектура, подсистемы и механизмы работы. Представлены результаты тестирования системы на различных сетевых маршрутах. Экспериментально показано, что система вносит незначительные задержки в работу ядра ОС. Приведен пример интеграции системы в программный пакет Ganglia.

Ключевые слова: *мониторинг, ядро ОС Linux, производительность сетей передачи данных, соединения TCP*

Введение

Мониторинг производительности сетей передачи данных является одной из наиболее важных и актуальных проблем, поскольку данные мониторинга составляют основу для проектирования, разработки и управления сетями передачи данных. В таком исследовании ключевую роль играет пропускная способность маршрута на уровне точка-точка, так как в основном она формирует представление пользователя о качестве услуг.

В настоящей работе представлена система мониторинга GetTCP+, которая собирает информацию о сетевых соединениях транспортного уровня модели OSI [1] и получает метрики пропускной способности маршрута на уровне точка-точка, представляющие особый интерес для анализа и проектирования. Первичные данные собираются на уровне ядра ОС, что позволяет системе получать информацию, недоступную в пространстве пользователя (например, размер скользящего окна), и позволяет избежать искажения данных и/или проблем неверной интерпретации переменных.

Такая организация процесса мониторинга может обеспечить любую необходимую информацию о поведении соединения на транспортном уровне.

Транспортный уровень, а конкретно протокол TCP [2], выбран для мониторинга по той причине, что только он имеет механизм контроля потоков данных и отвечает за распределенное управление доступом соединений к сетевой инфраструктуре. В данной работе не рассматривается протокол UDP и его расширения, такие как RTP, так как они не реализуют механизмы контроля потоков данных и не контролируют их доставку. При этом отметим, что мониторинг поведения протокола TCP обеспечивает гораздо больший объем информации о пропускной способности сетевого маршрута.

Известны и широко используются системы мониторинга общего назначения IOS NetFlow [3], tcpdump [4], Iperf [5] и др., а также системы для обработки и анализа данных мониторинга, например, Flowtools [6], Nagios [7], Ganglia [8], tcptrace [9]. В отличие от перечисленных систем GetTCP+ получает данные непосредственно на уровне ядра ОС. Эти данные либо недоступны на более высоких уровнях ОС, либо не могут быть получены надежным образом средствами мониторинга общего назначения. Далее представлен пример, демонстрирующий ошибку оценки размера сегмента TCP, выдаваемую системой tcpdump, которая была обнаружена авторами и исправлена в системе GetTCP+.

Представлены общая характеристика системы GetTCP+, ее архитектура, подсистемы и механизмы, а также результаты тестирования и интеграция с системой Ganglia.

Общая характеристика системы GetTCP+

GetTCP+ базируется на идеях системы GetTCP [10], но предоставляет ряд принципиально новых механизмов, существенно расширяющих возможности анализа поведения процесса передачи данных.

Система получает как полные данные об отдельных сегментах соединения (временная метка, номер последовательности, размер скользящего окна — CWND, время двойного оборота — RTT и др.), так и характеристики соединения в целом (адреса источника и приемника, список неподтвержденных сегментов и др.).

По желанию пользователя данные о поведении отдельных сегментов могут быть записаны в файл для дальнейшего анализа, либо удалены после завершения соединения и вычисления метрик производительности. В текущей версии системы GetTCP+ используют следующие метрики: адреса источника и приемников; общий и полезный объемы переданных данных; длительность соединения; доля потерянных сегментов; максимальный размер сегмента — MSS; максимальное и среднее значение CWND; максимальный размер окна, объявленный получателем — RWS; среднее значение RTT. Эти метрики позволяют вычислить текущую или будущую пропускные способности наблюдаемых сетевых маршрутов, в том числе на основе математических моделей.

Возможности сжатия данных мониторинга обеспечивает новая система их хранения. Предусмотрена поддержка механизма выгрузки сегментирования — TSO [11] и реализован инструментарий фильтрации соединений по набору направлений. Для идентификации адресов узлов и подсетей приемников используется уровень IP (v. 4/6).

Для поддержки перечисленных выше функций в ядре ОС Linux реализованы несколько новых контрольных точек. Кроме того, система контрольных точек реорганизована на основе событий протокола TCP.

Новым также является унифицированный интерфейс доступа к данным мониторинга, позволяющий использовать GetTCP+ совместно с другими инструментальными средствами. В частности, продемонстрирована возможность интеграции системы с инструментальным комплексом Ganglia.

Система имеет механизм конфигурирования, позволяющий управлять ее работой без перезагрузки как ядра ОС, так и модуля ядра GetTCP+. Кроме того, был исправлен ряд программных ошибок и система GetTCP была перенесена на новые версии ядра Linux (2.2.38—3.1.10).

Высокоуровневая архитектура системы

Система GetTCP+ состоит из двух подсистем — подсистема сбора данных и подсистема их хранения. Высокоуровневая архитектура системы представлена на рис. 1.

Подсистема сбора данных

Подсистема сбора данных основана на модуле ядра GetTCP и библиотеке libgettcp [10]. Библиотека представляет инструментарий для управления наборами контрольных точек, интерфейс для взаимодействия с модулем ядра, а также интерфейс для передачи собранных данных из пространства ядра в пространство пользователя.

Используя механизмы libgettcp, подсистема сбора данных получает из ядра ОС информацию о текущих событиях и состояниях фильтруемых соединений, в том числе событий, связанных с каждым сегментом.

Подсистема состоит из двух частей: модуля ядра, который обеспечивает сбор необходимых данных, и интерфейса для передачи данных в пространстве пользователя. Обмен данными между адресным пространством пользователя и адресным пространством ядра ведется через высокопроизводительную виртуальную файловую систему RelayFS [12].

Когда происходит одно их событий, связанных с соединением или отдельным сегментом, вызывается обработчик соответствующей контрольной точки в модуле TCP ядра, порождая структуру данных, содержащую информацию об этом событии. Далее эти данные поступают в подсистему хранения данных, работающую в пространстве пользователя.

Рис. 1. Архитектура системы GetTCP+

Механизм контрольных точек

Включенные авторами в исходный код модуля TCP ядра контрольные точки реализованы с помощью механизма точек останова tracepoint [13]. Функции — обработчики контрольных точек, которые вызываются при передаче им управления, реализованы в виде дополнительных исходных модулей ядра.

Эти контрольные точки обнаруживают для соединений факты наступления событий, связанных с попаданием в определенные состояния конечного автомата протокола TCP [14].

Первые два события flow_start_event и flow_end_event связаны с конечным автоматом состояний протокола TCP, как это показано на рис. 2. Они соответствуют началу и завершению соединения и позволяют получить о нем общую информацию.

Когда конечный автомат переходит в состояние "Established", возникает событие flow_start_event. Обработчик этого события применяет установленные фильтры и если текущее соединение должно быть подвергнуто мониторингу, помечает его и передает данные об узле назначения и используемом сетевом устройстве. Когда конечный автомат покидает состояние "Established", возникает событие flow_end_event. В данном случае обработчик предоставляет информацию о длительности соединения, максимальном размере окна передачи данных и максимальном размере сегмента.

События flow_ack_event и flow_retr_event (рис. 3) позволяют регистрировать переданные и потерянные сегменты. Каждый отправляемый сегмент добавляется ядром в список неподтвержденных сегментов. Сегмент, не подтвержденный в течение времени таймера повторной передачи — RTO, предполагается потерянным.

В случае получения подтверждения сегмент удаляется из списка неподтвержденных. Каждый неподтвержденный сегмент рассматривается как потерянный. Данные о нем передаются в хранилище. Таким

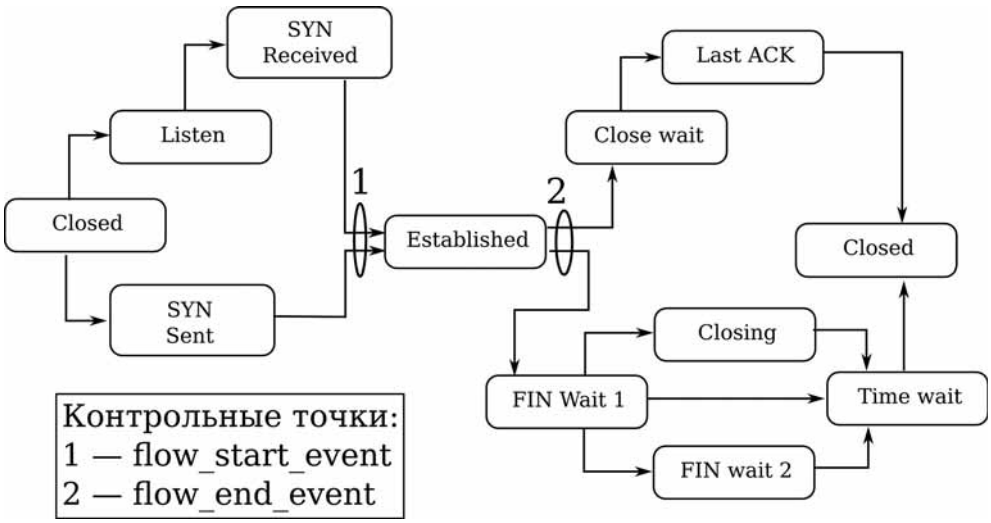


Рис. 2. Конечный автомат состояний TCP-соединения и события начала и конца передачи данных

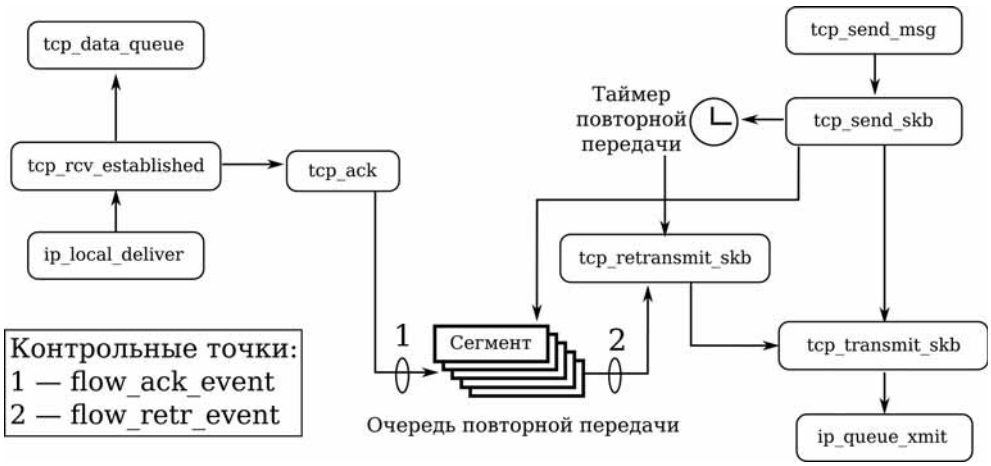


Рис. 3. Схема функций, реализующих механизм повторной передачи протокола TCP: контрольные точки связаны с передачей сегментов

образом, система сбора данных может оперировать полной информацией о каждом из сегментов. Информацию о подобных сегментах предоставляет обработчик события flow_retr_event.

Поддержка механизма ядра "выгрузка сегментирования"

Начиная с версии 2.6.13, ядро Linux реализует ряд механизмов для повышения производительности сетевого стека. Например, реализация алгоритма контроля потока TCP-CUBIC существенно отличается (как это показано в работе [15]) от оригинальной версии, описанной в работе [16]. Некоторые из этих изменений могут влиять на инструментальные средства мониторинга, искажая получаемые результаты. В частности, таким механизмом, влияющим на получаемые данные, является TSO — механизм выгрузки сегментирования протокола TCP.

Механизм TSO делегирует непосредственно сетевой интерфейсной карте задачу по разбиению больших блоков передаваемых данных на сегменты TCP, размер которых соответствует MTU, приемлемому в данной сети. Таким образом, модуль TCP получает возможность обрабатывать блоки данных, в несколько раз превосходящие по размеру доступный MTU, снижая нагрузку на центральный процессор. Данная технология особенно подходит для сетей с высокой пропускной способностью, например, для таких как, 1000BASE-T. Как было сказано в работе [11], TSO значительно повышает производительность сетевого стека. Внешне TSO выглядит как передача больших сегментов, в результате чего приложения пространства пользователя не могут оценить корректно размер передаваемых сегментов. Например, широко используемая утилита tcpdump выдает некорректные данные о размере передаваемых сегментов в случае использования механизма TSO. Детально этот эффект описан в работе [17].

Подсистема хранения данных

Данная подсистема состоит из следующих трех модулей: менеджер текущих данных, интерфейс хранилища, интерфейс систем анализа.

Менеджер текущих данных обрабатывает данные о незавершившихся соединениях. Запись о каждом переданном сегменте заносится в расположенный в оперативной памяти буфер соединения. При завершении соединения модуль обрабатывает содержимое буфера и записывает в хранилище метрики, описанные в разделе "Общая характеристика системы GetTCP+".

После вычисления метрик буферы соединения удаляются. При необходимости может быть задан режим сохранения полной информация о переданных сегментах.

Интерфейс хранилища обеспечивает взаимодействие между долговременным хранилищем данных и другими модулями подсистемы хранения. Внешнее хранилище располагается на жестком диске и позволяет накапливать историю характеристик соединений.

Внешнее хранилище реализовано с помощью файлов. Информация о каждой из подсетей хранится в отдельном каталоге (рис. 4).

Каждый каталог содержит файл списка соединений, в котором представлена основная информация о соединениях, связанных с наблюдаемыми узлами и подсетями.

При необходимости для каждого соединения может быть создан файл, содержащий полную последовательность записей о каждом отправленном сегменте. В этом файле каждая запись содержит временную метку, номер последовательности TCP, текущий размер окна передачи, RTT для данного сегмента. Следовательно, может быть восстановлена полная история поведения соединения.

Таким образом, данные могут записываться во внешнее хранилище в обычном и подробном режимах. В подробном режиме для каждого из соединений, кроме записи в файле — списке соединений, в хранилище также записываются файлы, содержащие буферы соединений.

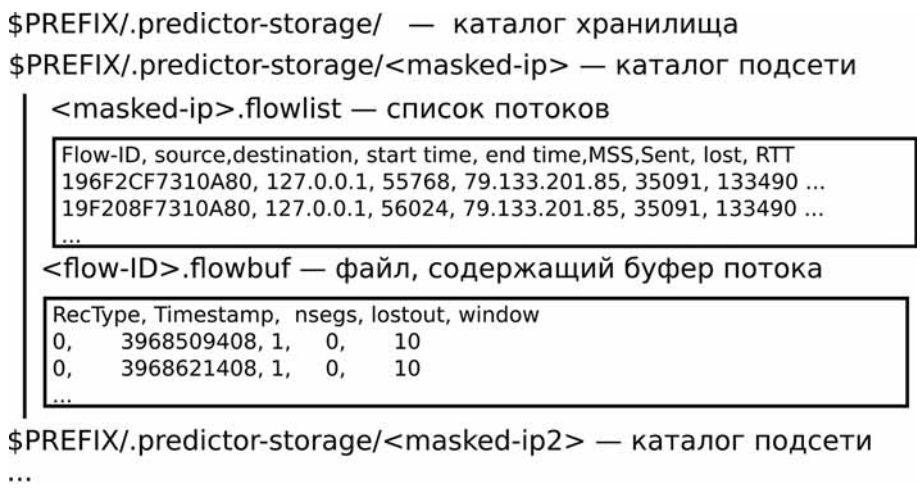


Рис. 4. Структура каталогов внешнего хранилища

Подробный режим существенно увеличивает объем хранимых данных. Так, например, объем файла буфера соединения, в ходе которого было передано 200 Мбайт, будет составлять 10 Мбайт. В обычном режиме запись в файле — списке соединений будет содержать 200—300 байт независимо от объема переданных данных.

Текущая реализация хранилища также обеспечивает три варианта хранения данных.

Первый вариант — формат журнала. В этом случае в обычном режиме одна строка файла представляет одно соединение, что увеличивает наглядность данных, но так же значительно увеличивает их размер. Этот вариант в основном предназначен для отладки и настройки системы.

Второй вариант — файлы формата CSV (*Comma-Separated Values*) предназначен для автоматической обработки данных мониторинга. Объем и наглядность при этом незначительно уменьшаются.

Третий вариант — бинарный формат, который обеспечивает существенное увеличение скорости обработки данных и уменьшение занимаемого ими объема.

Интерфейс внешних приложений обеспечивает доступ к данным мониторинга для других программных систем.

Система фильтрации

Фильтрация позволяет задать набор сетевых карт источника, где установлена система GetTCP, и набор адресов приемников (устройств или подсетей), соединения с которыми подлежат мониторингу. Реализация механизма фильтрации в пространстве ядра позволяет избежать передачи ненужных данных в пространство пользователя. Механизм фильтрации основан на списке сетевых карт источника и на списке узлов и подсетей. Для управления этими списками библиотека libgettcp предоставляет следующие две функции.

- Функция gettcp_conf_adddev(dname) — добавляет устройство с указанным именем в список устройств, подлежащих мониторингу. Мониторинг ведется только для тех устройств, которые указаны в данном списке. Поведение данного фильтра может быть инвертировано с помощью опции DEV_FLTR_EXCLUDED.
- Функция gettcp_conf_addadr(adr, mask) — добавляет подсеть или отдельный узел в список узлов, подлежащих мониторингу. Фильтрация в данном случае выполняется по адресу и указанной маске подсети. Поведение данного фильтра идентично поведению фильтра устройств. Оно может быть инвертировано с помощью опции ADDR_FLTR_EXCL.

Фильтрация соединений выполняется при обработке события flow_start_event. Необходимость мониторинга для каждого соединения отмечается в поле probed_sock структуры tcp_sock. Дальнейшее поведение системы в отношении этого соединения полностью определяется значением поля probed_sock. Если соединение не подлежит мониторингу, все связанные с ним события игнорируются, и данные о нем не сохраняются.

Тестирование системы

Система GetTCP+ была протестирована на ряде сетевых маршрутов, имеющих различные характеристики и структуру. К их числу относятся как маршруты с высокой пропускной способностью, так и с низкой пропускной способностью, высоким временем кругового оборота и уровнем потерь. Всего было проведено четыре серии экспериментов в сети, схема которой представлена на рис. 5. Монитор GetTCP+ был запущен на узле *A* с процессором Intel Celeron 2.20 GHz, 1 Гбайт RAM, подключенном к сетям Ethernet 1000Base-T и 3G (со скоростью 256 кбит/с). Первая серия экспериментов была проведена для маршрута с высокой пропускной способностью и малым временем кругового оборота. Данный маршрут представляет

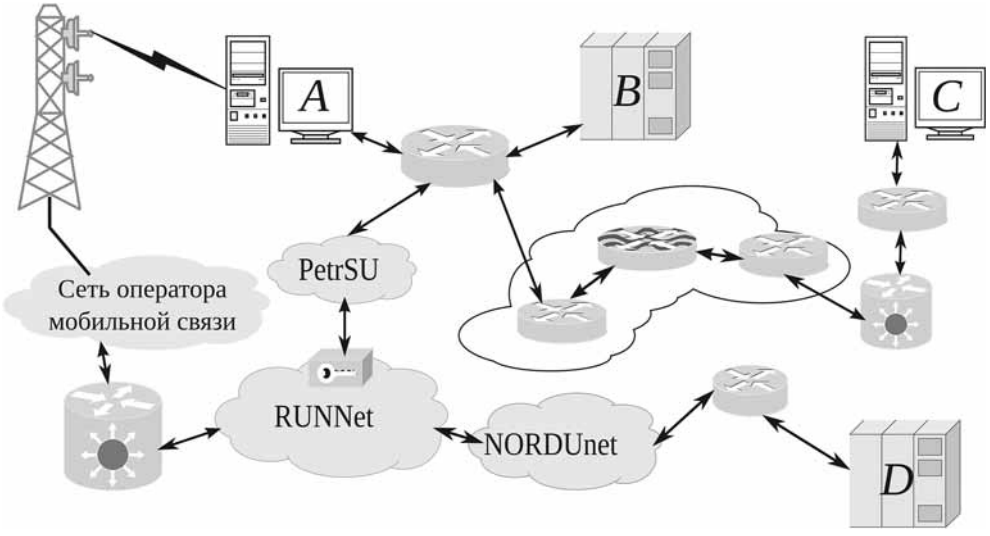


Рис. 5. Схема тестовой сети

Событие	Обработчик	Время выполнения, мкс
flow_start-event	tcp_start_event()	16,904
flow_ack_event	tcp_ack_event()	0,628
flow_retr_event	tcp_retr_event()	1,172
flow_end_event	tcp_end_event()	2,48

собой два узла, соединенные маршрутизатором (маршрут $A \rightarrow B$ на рис. 5). В данном случае проводилось нагрузочное тестирование системы.

Вторая и третья серии экспериментов проводились для сетевых маршрутов с различной пропускной способностью и временем кругового оборота (маршруты $A \rightarrow C$ и $A \rightarrow$ PetrSU $\rightarrow D$ на рис. 5). Четвертая серия экспериментов была проведена для маршрута с высокими временем кругового оборота и уровнем потерь и низкой пропускной способностью (маршрут $A \rightarrow$ Сеть оператора мобильной связи $\rightarrow D$ на рис. 5).

В рамках каждого из экспериментов с помощью утилиты iperf генерировались соединения TCP, в ходе которых передавалось по 200 Мбайт данных. Общая информация о соединениях, такая как длительность, объем переданных данных и средняя пропускная способность, полученная GetTCP+ и вычисленная утилитой iperf, совпали. Полученная последовательность размеров скользящего окна TCP — CWND также соответствует текущим реализациям алгоритмов контроля потока TCP NewReno. Время кругового оборота, полученное с помощью утилиты ping, также совпало со значением, полученным GetTCP+.

Кроме этого был проведен ряд тестов для оценивания задержек, вносимых системой в работу сетевого стека за счет выполнения обработчиков контрольных точек. Время выполнения обработчиков оценивалось с помощью инструмента Ftrace [18]. Это трассировщик функций, включенный в ядро Linux, начиная с версии 2.6.27. С его помощью также можно оценить и время выполнения отдельных функций [19]. В результате были получены значения, представленные в таблице.

Следует отметить, что обработчики tcp_start_event() и tcp_end_event() вызываются однократно для каждого соединения, а обработчик tcp_retr_event() — только для потерянных сегментов, доля которых в ходе эксперимента не превышала 5 % в худшем случае. Среднее время выполнения функции ядра tcp_transmit_skb(), внутри которой происходит вызов обработчика tcp_ack_event(), составляет 8,406 мкс, а функции ядра tcp_retransmit_skb() — 17,8 мкс. Время выполнения обработчика tcp_ack_event() меньше времени выполнения функции tcp_transmit_skb() примерно в 13 раз, и меньше времени выполнения функции tcp_retransmit_skb() примерно в 28 раз. Это означает, что система GetTCP+ не вносит существенных задержек в работу ядра Linux. Таким образом, GetTCP+ была протестирована в различных условиях и показала высокую стабильность и производительность.

Интеграция с системой Ganglia

Как было сказано ранее, унифицированный интерфейс доступа к данным мониторинга позволяет интегрировать систему в существующие средства мониторинга, что дает возможность использовать разработанную систему в комплексе с другими програм-

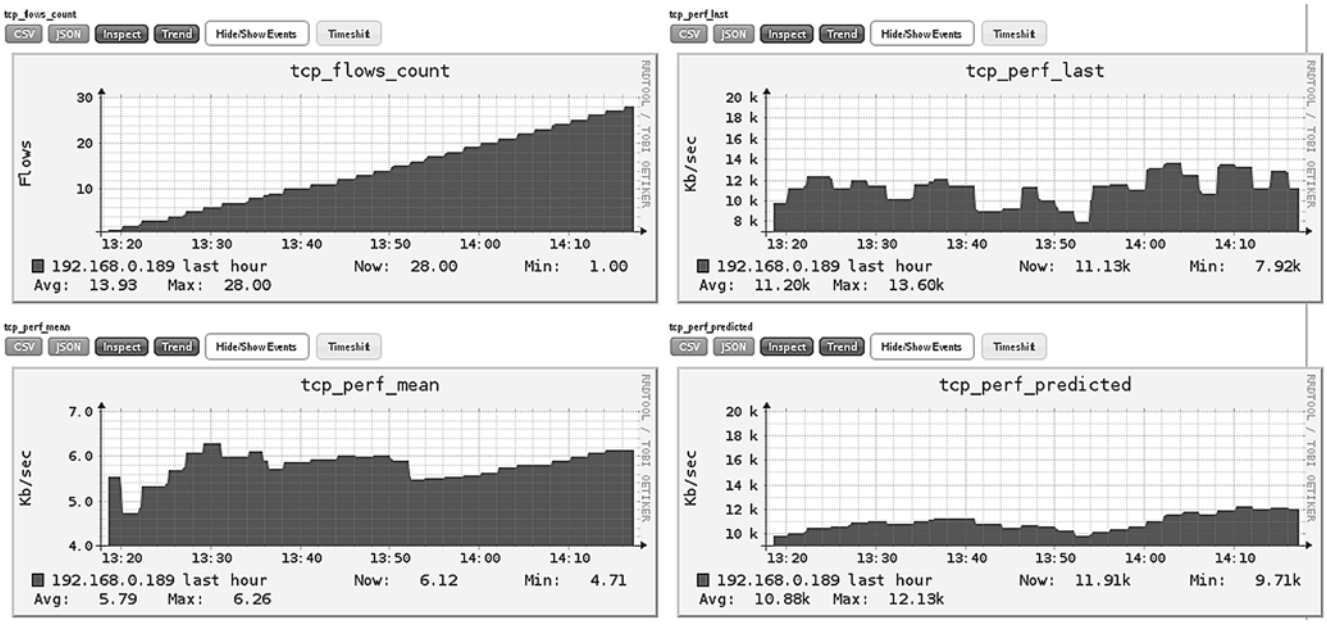


Рис. 6. Пример интеграции GetTCP+ с системой Ganglia

мными средствами. В частности, в ходе разработки система GetTCP+ была интегрирована с инструментальным комплексом для мониторинга в сети Ganglia в качестве подключаемого модуля. Для передачи в Ganglia данных, предоставляемых системой GetTCP+, был создан интерфейсный модуль, использующий прикладной программный интерфейс Ganglia для подключаемых дополнений [20], предоставляемый демоном сбора данных этой системы. При этом GetTCP+ являлась источником данных о сетевой активности узла для Ganglia.

Система GetTCP+ предоставляет информацию о сетевых соединениях и пропускной способности маршрутов между различными узлами распределенной вычислительной системы. В результате чего на основе данных можно получить более полную информацию о текущем состоянии распределенной системы. На рис. 6 представлены результаты мониторинга сетевых соединений между двумя узлами, отображаемые Ganglia, в частности, число соединений (tcp_flows_count), пропускная способность в ходе последнего соединения (tcp_perf_last) и средняя пропускная способность маршрута (tcp_perf_mean). Подобные данные могут быть получены для любых узлов и подсетей, как являющихся частью распределенной системы, так и внешних по отношению к ней.

Данные о сетевой активности извлекают непосредственно из внешнего хранилища GetTCP+. Анализ проводится только для тех узлов и подсетей, которые указаны в файле конфигурации GetTCP для текущего узла.

Таким образом, для получения полной информации о пропускной способности сети для всей распределенной системы необходимо на каждом узле, где запущен демон сбора метрик системы Ganglia, установить систему GetTCP+ с соответствующим интерфейсным модулем.

Заключение

Дано описание разработанной системы GetTCP+ для мониторинга пропускной способности соединений на уровне точка-точка. Система предоставляет общие и/или подробные данные о пропускной способности TCP-соединений. Система обеспечивает механизмы фильтрации, долговременного хранения, динамического конфигурирования, использует механизм контрольных точек.

В отличие от существующих аналогов, данные о поведении соединений поступают непосредственно из ядра операционной системы Linux, что обеспечивает их точность и полноту, устраняя искажения, которые могут быть внесены инструментами мониторинга, работающими в пространстве пользователя (например, tcpdump).

Система поддерживает важные особенности сетевого стека, такие как механизм выгрузки сегментирования. Интерфейсы, предоставляемые системой, дают возможность интеграции с другими инструментами анализа сетей.

В дальнейшем планируется разработка аналитической части приложения и ее интеграция в систему. Реализация внешних интерфейсов для системы хранения также позволит расширить возможности использования системы.

Список литературы

1. **International** Standard ISO/IEC 7498-1. 1996. 68 p.

2. **Allman M., Paxson V., Blanton E**. RFC 5681: TCP Congestion Control [Электронный ресурс]. URL: http://datatracker.ietf.org/doc/rfc5681/

3. **Cisco** IOS NetFlow [Электронный ресурс]. URL: http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html

4. **Tcpdump**/Libpcap public repository [Электронный ресурс]. URL: http://www.tcpdump.org/

5. **Iperf** [Электронный ресурс]. URL: http://iperf.sourceforge.net/

6. **Flow-tools** information [Электронный ресурс]. URL: http://www.splintered.net/sw/flow-tools/

7. **Nagios** Documentation [Электронный ресурс]. URL: http://www.nagios.org/documentation

8. **Ganglia** Monitoring System [Электронный ресурс]. URL: http://ganglia.info/

9. **Tcptrace** Homepage [Электронный ресурс]. URL: http://www.tcptrace.org/

10. **Пономарев В. А., Богоявленская О. Ю., Богоявленский Ю. А.** Конфигурируемая модульная система мониторинга поведения транспортного протокола на уровне ядра операционной системы // Информационные технологии. 2010. № 1. С. 54—59.

11. **Linux:** TCP Segmentation Offload (TSO) [Электронный ресурс]. URL: http://kerneltrap.org/node/397

12. **Relay** (formerly Relayfs) [Электронный ресурс]. URL: http://relayfs.sourceforge.net/

13. **Desnoyers M.** Using the Linux Kernel Tracepoints [Электронный ресурс]. URL: http://www.kernel.org/doc/Documentation/trace/tracepoints.txt

14. **RFC 793:** Transmission Control Protocol [Электронный ресурс]. URL: http://datatracker.ietf.org/doc/rfc793/

15. **Leith D. J., Shorten R. N., McCullagh G**. Experimental Evaluation of Cubic TCP // Extended version of paper presented at Proc. Protocols for Fast Long Distance Networks 2007, Los Angeles. [Электронный ресурс]. URL: http://www.hamilton.ie/net/pfldnet2007_cubic_final.pdf

16. **Ha S., Rhee I., Xu L.** CUBIC: A New TCP-Friendly Hight-Speed TCP Variant // ACM SIGOPS Operating Systems Review — Research and developments in the Linux kernel. 2008. V. 42, No 5. P. 64—74.

17. **Sannikov A. A., Bogoiavlenskaia O. I., Bogoiavlenskii I. A.** GetTCP+: Performance Monitoring System at Transport Layer // Internet of Things, Smart Spaces, and Next Generation Networking, Lecture Notes in Computer Science 8121. 2013. August. P. 236—246.

18. **Rostedt S.** Ftrace — function tracer [Электронный ресурс]. URL: http://git.kernel.org/cgit/linux/kernel/git/torvalds/linux.git/tree/Documentation/trace/ftrace.txt

19. **Bird T.** Measuring Function Duration with Ftrace // Proceedings of the Linux Symposium. 13—17 July, 2009. Monthreal, Canada. 2009. P. 47—54.

20. **Gmond** Python metric modules [Электронный ресурс]. URL: http://ganglia.wiki.sourceforge.net/ganglia_gmond_python_modules/

УДК 004.825

С. Д. Махортов, д-р физ.-мат. наук, доц., зав. каф., e-mail: sd@expert.vrn.ru,
А. Н. Шмарин, аспирант, Воронежский государственный университет

Нечеткий LP-вывод и его программная реализация

Предложена модификация LP-вывода, базирующаяся на идее динамического программирования — разбиении основной задачи на отдельные части (подзадачи) и решении каждой подзадачи только один раз. Приведены особенности "быстрого" расчета степеней истинности решений. Представлен способ улучшения качества решений в кластерно-релевантном LP-выводе, основанный на вычислении оценок возможных решений без построения прообразов. Описана программная реализация.

***Ключевые слова**: LP-вывод, И/ИЛИ-граф, динамическое программирование, нечеткие правила, программная реализация*

Введение

Теория LP-структур предоставляет эффективный аппарат для исследования процессов управления знаниями, верификации знаний, оптимизации логического вывода в различных разделах информатики [1—3]. В частности, в статье [1] представлены усовершенствованные алгоритмы обратного вывода, основанные на решении продукционно-логических уравнений в LP-структурах. Стратегия релевантного LP-вывода направлена на минимизацию числа обращений к внешним источникам информации.

В настоящей работе описана новая идея оптимизации LP-вывода. Ее преимущество обеспечивается построением дополнительной структуры данных для хранения найденных результатов, что позволяет экономить время при обходе графа базы знаний. Такой структурой является дерево достижимости ИЛИ-вершин, хранящее в своих узлах информацию для построения всех прообразов объекта экспертизы [1]. Еще одна особенность полученных в ходе настоящего исследования результатов — возможность обработки нечетких правил. Кроме того, впервые описана компьютерная реализация предложенных идей. Снижение сложности обхода достигается в силу того обстоятельства, что достаточно лишь один раз исследовать поддеревья базы знаний с корнем в вершине, выводимой более чем одним правилом, и использовать полученную информацию при обходе других связанных вершин. Таким образом, LP-вывод можно выполнять в два этапа:

- построение дерева достижимости ИЛИ-вершин;
- обход созданного дерева достижимости и построение прообразов.

Одним из важных результатов общей теории LP-структур является тот факт, что частное решение продукционно-логического уравнения на решетке существует тогда и только тогда, когда соответствующий слою отношения ориентированный граф не содержит циклов [1]. В силу данного обстоятельства, для корректной работы описываемого ниже метода необходимо исключить из рассмотрения все слои с циклами. Однако такое ограничение нетривиально и его практическая реализация требует отдельного исследования. Далее в работе будет подразумеваться, что все слои с циклами заранее исключены из рассмотрения.

1. Базовые определения

Используемая в настоящей работе терминология соответствует публикациям [1—5]. Введем несколько дополнительных понятий.

Утверждение об объекте базы знаний является *ИЛИ-вершиной*, если оно выводится более чем одним правилом.

Если более чем одно правило выводит одно и то же утверждение об объекте базы знаний, то будем считать, что такие правила *выводят ИЛИ-вершину*. Каждое такое правило будем называть *ИЛИ-связью*.

Деревом вывода утверждения называется поддерево базы знаний, корнем которого является данное утверждение.

Утверждение *участвует в выводе предпосылки правила*, если оно включено в предпосылку, либо включено в дерево вывода одного из утверждений предпосылки.

ИЛИ-вершина является *непосредственно достижимой* из ИЛИ-связи, если данная вершина участвует в выводе ИЛИ-связи и между ними существует путь, не включающий в себя другие ИЛИ-вершины. Такая ИЛИ-вершина будет являться *потомком* ИЛИ-связи. Соответственно ИЛИ-связь будет называться *предком* ИЛИ-вершины.

2. Построение дерева достижимости ИЛИ-вершин

Для создания дерева достижимости необходимо, чтобы вершины графа базы знаний удовлетворяли перечисленным ниже требованиям.

1. Каждой ИЛИ-связи должны быть поставлены в соответствие следующие структуры данных:

- LP-код [5] — для хранения информации о начальных утверждениях, участвующих в выводе предпосылок правила;
- список потомков — для хранения индексов ИЛИ-вершин, являющихся прямыми потомками.

2. Каждая ИЛИ-вершина должна иметь список предков — для хранения индексов ИЛИ-связей, являющихся прямыми предками.

3. Ребра должны создаваться между ИЛИ-связью и ее потомками — ИЛИ-вершинами.

Узлами дерева достижимости будут являться ИЛИ-вершины и выводящие их правила, а ребрами — отношения "предок/потомок". Корни дерева — это вершины, соответствующие целевым утверждениям. Ребрам, исходящим из корней, соответствуют правила, выводящие эти утверждения. Создание ребра между ИЛИ-связью и ИЛИ-вершиной выполняется следующим образом:

- в список потомков ИЛИ-связи добавляется индекс ИЛИ-вершины;
- в список предков ИЛИ-вершины добавляется индекс ИЛИ-связи.

При построении дерева достижимости необходимо использовать общий указатель для хранения индекса текущей ИЛИ-связи. Будем называть этот указатель *маркером верхней ИЛИ-связи*.

Алгоритм построения дерева достижимости ИЛИ-вершин

1. Выбрать очередное целевое утверждение.

2. Обойти все правила, выводящие текущее утверждение.

2.1. При выборе очередного правила сохранить предыдущее значение маркера верхней ИЛИ-связи и присвоить ему индекс выбранного правила.

2.2. Выполнить обход утверждений, входящих в предпосылку правила, и их деревьев вывода.

2.3. Если на очередном шаге обхода встретилось утверждение, выводящееся более чем одним правилом — создать в дереве достижимости ребро между данным утверждением и правилом, на которое указывает маркер верхней ИЛИ-связи. Если данное утверждение имеет статус "не открыто", присвоить ему статус "открыто" и выполнить для него Шаг 2.

2.4. Если достигнуто начальное утверждение, то его необходимо добавить к LP-коду правила, на которое указывает маркер верхней ИЛИ-связи.

2.5. Когда завершен обход предпосылок текущей ИЛИ-связи, необходимо восстановить сохраненное ранее значение маркера верхней ИЛИ-связи.

Очевидно, что вычислительная сложность построения дерева достижимости равна сложности обхода графа в глубину, т. е. $O(N)$, где N — число правил в базе знаний. Память, занимаемая деревом достижимости, не превышает объем памяти для графа базы знаний.

3. Вычисление степеней истинности

При обходе дерева нечеткой базы знаний не всегда можно однозначно рассчитать степень истинности утверждений [6]. Причина в том, что в дереве вывода могут встречаться ИЛИ-узлы. Правила, выводящие данные узлы, могут становиться невыполнимыми при конкретизации начальных объектов.

Вариантом решения отмеченной задачи может служить вычисление степеней истинности после построения прообразов. Однако такой подход приводит к увеличению вычислительной сложности и отсутствию возможности реализации кластерно-релевантного LP-вывода [1]. Более продуктивной оказывается реализация степеней истинности формулами, содержащими неизвестные переменные. Такие формулы могут быть представлены в виде дерева, узлами которого являются операции T-нормы и S-нормы [6].

Пусть сформулировано некоторое утверждение об объекте базы знаний, выводимое лишь одним правилом. Если все факты в предпосылке этого правила являются начальными, либо все соответствующие им деревья вывода не включают ИЛИ-вершин, то степень истинности исходного утверждения может быть

рассчитана в виде числового значения. Такую степень истинности будем называть *заданной явно.*

Дерево, моделирующее формулу расчета степеней истинности утверждения, будем называть *деревом истинности вершины.*

Замечание 1. Для корректности описываемого подхода необходимо, чтобы операция Т-нормы удовлетворяла условию $T(x, 0) = 0$. Данное условие выполняется для большинства применяемых на практике классов Т-норм.

Структура дерева истинности вершины. Каждый узел дерева соответствует структуре данных, которая включает:

- идентификатор типа операции, принимающий значения "Т-норма" либо "S-норма";
- числовое значение степени истинности;
- список аргументов, каждый элемент которого представляет ссылку на дерево истинности вершины, соответствующей аргументу.

Если узел моделирует Т-норму, то после заполнения списка аргументы с явно заданной степенью истинности необходимо удалить. Вместо них создается новый аргумент, для которого степень истинности равна результату применения операции Т-нормы к степеням истинности удаленных узлов. Данное преобразование возможно в силу того, что Т-норма коммутативна и ассоциативна по определению.

В худшем случае узел дерева истинности может иметь список аргументов, длина которого близка к числу вершин M графа базы знаний. Другими словами, верхняя оценка числа аргументов узла равна $O(M)$. При этом каждый узел дерева истинности в свою очередь ассоциирован с соответствующей вершиной графа базы знаний. Поэтому дерево истинности имеет верхнюю оценку объема используемой памяти $O(M^2)$. Поскольку построение дерева истинности заключается в последовательном построении всех его узлов, то оценка вычислительной сложности такого действия также равна $O(M^2)$.

Замечание 2. Если степень истинности задана явно, то дерево вырождается в узел, хранящий данное числовое значение.

Замечание 3. Если задано числовое значение степени истинности узла, то список аргументов узла считается пустым.

Описанный способ сокращения аргументов Т-нормы нельзя применять для операции S-нормы. Данное ограничение связано с тем, что моделирующие S-норму узлы создаются для ИЛИ-вершин, на основе которых могут быть получены различные прообразы.

Сохранение структурного представления степеней истинности необходимо для того, чтобы на этапе построения прообразов была возможность быстро (без обхода графа базы знаний) выполнить следующие действия:

- расчет степени истинности целевого утверждения для каждого прообраза;
- проверку условия, можно ли увеличить степень истинности выведенного целевого утверждения с помощью конкретизации начального объекта базы знаний;
- при выполнении кластерно-релевантного LP-вывода [1] — расчет оценки максимально возможной степени истинности для подграфа базы знаний, определяющего группу прообразов, и использование данного показателя для принятия решения о необходимости построения соответствующих прообразов.

Создание деревьев истинности вершин необходимо выполнять на этапе построения дерева достижимости. При этом кроме создания деревьев истинности для утверждений, необходимо строить деревья истинности для правил, выводящих ИЛИ-вершины.

4. Обход дерева достижимости ИЛИ-вершин и расчет показателей релевантности

Построение множества минимальных начальных прообразов позволяет выполнять начальный расчет и последующую модификацию коэффициентов, на основе которых выбирается наиболее подходящий для конкретизации (релевантный) объект экспертизы.

Основными коэффициентами (показателями релевантности) тестируемого объекта являются следующие параметры [1]:

- счетчик вхождений в прообразы;
- число вхождений в прообразы с малой мощностью;
- число вхождений в прообразы с высокой степенью истинности.

Однако построение всего множества минимальных начальных прообразов имеет экспоненциальную сложность и требует экспоненциального объема памяти для их хранения. Решение этой задачи заключается в том, чтобы не вычислять прообразы в явном виде. Это возможно в силу того обстоятельства, что слои с циклами исключены из рассмотрения. Дело в том, что дерево достижимости хранит всю необходимую информацию о прообразах, и каждый путь в нем соответствует слою. Так как слои с циклами исключены, оставшиеся слои всегда имеют решение. Для построения решения необходимо обойти соответствующий путь, объединить LP-коды и упростить дерево истинности до числового значения.

В силу того, что слой эквивалентен пути в дереве достижимости, следует ряд отмеченных далее важных фактов.

1. Задача подсчета числа вхождений каждого атома в прообразы может быть сведена к задаче вычисления числа всевозможных путей от листовых вершин дерева достижимости к корневым.

Из теории графов известно [7], что матрица смежности, возведенная в степень k, дает информацию обо всех путях длины k. Пусть $\mathbf{E}$ — матрица смежности дерева достижимости. Чтобы получить информацию о числе всевозможных путей длины от 1 до k между любыми двумя вершинами, достаточно вычислить суммарную матрицу $\mathbf{E} + \mathbf{E}^2 + \mathbf{E}^3 + ... + \mathbf{E}^k$. Самый длинный путь в графе без циклов не может превышать число вершин в графе, уменьшенное на единицу. Это значит, что для получения информации о всевозможных путях между любыми двумя вершинами в дереве достижимости, необходимо вычислить сумму $\mathbf{D} = \mathbf{E} + \mathbf{E}^2 + \mathbf{E}^3 + ... + \mathbf{E}^{S-1}$, где S — число вершин в дереве достижимости.

Простой алгоритм перемножения матриц имеет кубическую сложность, а матрица смежности имеет размер $S \times S$. Из приведенного выше следует, что оценка вычислительной сложности для задачи подсчета всевозможных путей в дереве достижимости равна $O(S^4)$, т. е. необходимо S раз найти произведение матрицы размером $S \times S$ на себя.

При подсчете всевозможных путей целесообразно вести стек LP-кодов, добавляя в него новый элемент при посещении очередной вершины. Новый LP-код должен вычисляться путем сложения (с контролем противоречивости) предыдущего элемента стека и LP-кода текущей ИЛИ-связи. Если при вычислении нового LP-кода возникло противоречие, необходимо завершить обход текущей ветви и вернуться назад.

Таким образом, общая оценка сложности этой задачи равна $O(S^4M_0)$, где M_0 — число начальных утверждений в базе знаний. Верхняя же оценка объема памяти стека LP-кодов будет равна $O(M_0S)$.

Замечание 4. При использовании быстрых алгоритмов перемножения матриц, асимптотическая оценка будет ниже.

2. Слои могут быть объединены в группы, соответствующие связным подграфам дерева достижимости. Нахождение прообразов минимальной мощности и максимальной степени истинности может быть выполнено для группы слоев. Таким образом, вычисление числа вхождений в прообразы с малой мощностью или высокой степенью истинности без построения самих прообразов можно выполнить следующим способом.

♦ В дереве достижимости для каждой ИЛИ-вершины необходимо определить правила (ИЛИ-связи), которые могли бы выводить данную вершину применением начальных утверждений, соответствующих атомам

— прообраза минимальной мощности;

— прообраза максимальной степени истинности.

Данное действие можно выполнить путем обхода дерева достижимости в глубину. С каждым правилом (ИЛИ-связью) при этом необходимо связать два значения:

- минимальную мощность прообраза, который мог бы быть найден для поддерева вывода предпосылок этого правила, если бы утверждение в заключении правила было целью;
- максимальную степень истинности, которая может быть получена для заключения правила.

♦ После выполнения обхода в глубину для целевых утверждений будут известны максимально возможная степень истинности и минимально возможная мощность прообраза, а также пути в графе, по которым можно достигнуть листовых вершин.

Для вычисления показателей релевантности будет достаточно выполнить обход от корня к листам по ребрам, для которых найдены наилучшие оценки (наибольшая степень истинности и наименьшая мощность). В процессе обхода необходимо вести стек LP-кодов и проверять элемент, добавляемый в стек, на противоречивость. Если обнаружено противоречие, обход текущей ветви завершается. Если же противоречий нет, нужно обновить показатели релевантности для объектов из LP-кода текущей ИЛИ-вершины.

Вычислительная сложность выполняемого обхода зависит от числа путей с наилучшими оценками. В худшем случае это будут почти все возможные пути в графе. Однако такое возможно лишь для реберно M-связных графов баз знаний, либо для графов, реберная степень связности которых близка к числу M (т. е. к числу вершин в графе базы знаний). Если же степень реберной связности графа базы знаний близка к единице, то число всевозможных путей с наилучшими оценками будет в худшем случае полиномиальным [7].

Таким образом, можно сделать следующие выводы о вычислительной сложности описанного метода.

- Если граф базы знаний имеет степень реберной связности, близкую к числу вершин в данном графе, то сложность описанного метода экспоненциальна.
- Если граф базы знаний имеет степень реберной связности, близкую к единице, то оценка вычислительной сложности описываемого метода будет равна $O(N) + O(M^2) + O(S^4M_0) + O(S^2P) = O(N + M^2 + S^4M_0 + S^2P)$, где N — число правил в базе знаний; M — число вершин в графе базы знаний; M_0 — число начальных утверждений в базе знаний; S — число вершин в дереве достижимости; P — число всевозможных путей в дереве достижимости, посещаемых при обходе ребер с наилучшими оценками.

5. Программная реализация

Описанный метод реализован в программной системе fLPExpert (fuzzy-LPExpert). Данный продукт является кроссплатформенным программным обеспечением, распространяемым по лицензии GPL 2.1 и работающим в операционных системах Windows и Linux. Получить последнюю версию исполняемых файлов, исходных кодов, дополнительную информацию о работе, архитектуре и методике развертывания системы можно в Интернете: http://sourceforge.net/projects/lpexpert.

Система реализована на языке C++ с использованием библиотеки Qt4. Описанный метод включен в систему в тестовом режиме и может быть активизирован через настройки проекта. На текущий момент полноценное практическое применение описанного метода представляет определенные трудности в силу того, что алгоритм корректного исключения из рассмотрения слоев с циклами еще не разработан.

Реализация описанных алгоритмов выполнена в классе LPGraph и состоит из следующих компонентов.

- Подготовка данных. Активизируется методом LPGraph::prepare(). Он удаляет из базы знаний правила, которые могут породить цикл. С точки зрения корректности системы в общем случае так поступать нецелесообразно. Данный подход применяется лишь для подготовки корректных тестовых данных.
- Построение дерева истинности и дерева достижимости. Запускается вызовом метода LPGraph::build(). Дерево достижимости представлено в виде класса ReachabilityThree, который представляет собой контейнер для списка смежности. Узлы дерева истинности являются экземплярами класса TruthNode и размещаются в контейнерах заключений правил — в элементах Trule::Perm. Данные функциональные возможности соответствуют действиям, описанным в разд. 2 и 3.
- Вычисление показателей релевантности и индекса объекта, наиболее подходящего для конкретизации. Активизируется методом LPGraph::getToAsk(). Он выполняет действия, описанные в разд. 4.
- Присваивание конкретного значения объекту базы знаний. Выполняется методом LPGraph::setValue (int object, const TValue & value). После конкретизации объекта выполняется прореживание пространства поиска путем обновления дерева достижимости и исключения из него всех подграфов, обход которых стал невозможен.
- Проверка наличия ответа. Реализуется методом LPGraph::hasAnswer(). Метод возвращает значение ответа либо 0, если информации для вывода ответа недостаточно. Результат −1 соответствует ответу "решений нет".

Заключение

В статье рассмотрены теоретические особенности оптимизированного алгоритма LP-вывода с поддержкой нечеткости на уровне коэффициентов доверия. Предложены алгоритмические решения для улучшения качества решений кластерно-релевантного LP-вывода. Получены оценки сложности рассматриваемых алгоритмов. Описана программная реализация представленных идей.

Эффективность и практическая значимость разрабатываемого подхода заключается в существенном снижении объема вычислений, что дает возможность применения метода LP-вывода на больших базах знаний, а также и в более сложных логических системах информатики [8].

Список литературы

1. **Болотова С. Ю., Махортов С. Д.** Алгоритмы релевантного обратного вывода, основанные на решении продукционно-логических уравнений // Искусственный интеллект и принятие решений. 2011. № 2. С. 40—50.
2. **Махортов С. Д.** LP-структуры для обоснования и автоматизации рефакторинга в объектно-ориентированном программировании // Программная инженерия. 2010. № 2. С. 15—21.
3. **Махортов С. Д.** Основанный на решетках подход к исследованию и оптимизации множества правил условной системы переписывания термов // Интеллектуальные системы. 2009. Т. 13. Вып 1—4. С. 51—68.
4. **Levi G., Sirovich F.** Generalized And/Or Graphs // Artificial Intelligence Journal. 1976. № 7. P. 243—259.
5. **Махортов С. Д.** Интегрированная среда логического программирования LPExpert // Информационные технологии. 2009. № 12. С. 65—66.
6. **Заде Л. А.** Понятие лингвистической переменной и его применение к принятию приближенных решений: пер. с англ. М.: Мир, 1976. 165 с.
7. **Харари Ф.** Теория графов: пер. с англ. М.: Едиториал УРСС, 2003. 296 с.
8. **Чечкин А. В.** Нейрокомпьютерная парадигма информатики // Нейрокомпьютеры: разработка, применение. 2011. № 7. С. 3—9.

УДК 004.023

П. В. Матренин, магистрант, e-mail: pavel.matrenin@gmail.com,
В. Г. Секаев, канд. техн. наук, доц., Новосибирский государственный технический университет

Системное описание алгоритмов роевого интеллекта

Рассмотрены алгоритмы роевого интеллекта с позиций системного подхода. Даны определения их основным понятиям в терминах теории систем и предложена единая схема описания для всех алгоритмов роевого интеллекта. Схема применена для описания алгоритмов роя частиц и колонии муравьев.

Ключевые слова: *алгоритм колонии муравьев, алгоритм роя частиц, глобальная оптимизация, роевой интеллект, системный подход*

Введение

Научная и практическая виды деятельности предполагают решение большого числа задач, которые сводятся к задачам дискретной или непрерывной глобальной оптимизации. При разработке крупных технологических, энергетических, аэрокосмических, информационных и других сложных комплексов возникают вопросы, обусловленные выбором оптимальной организации взаимодействия элементов, режимов их функционирования, что связано с необходимостью решения задач оптимизации [1]. Особенности таких задач [2]:

1) нелинейность;
2) недифференцируемость;
3) многоэкстремальность;
4) овражность;
5) отсутствие аналитического выражения;
6) высокая вычислительная сложность;
7) высокая размерность пространства поиска;
8) сложная топология области допустимых значений.

Найти решение точными методами, как правило, невозможно в силу свойств 1, 2, 5. Кроме того, свойства 6, 7, 8 приводят к очень большим затратам времени при использовании точных методов. По этой причине с середины 1980-х гг. для решения оптимизационных задач были предложены и начались исследования различных стохастических алгоритмов. К ним относятся: эволюционные алгоритмы; алгоритмы, использующие концепцию роевого интеллекта; алгоритмы, основанные на иных механизмах живой и неживой природы [2]. Поскольку все предложенные алгоритмы являются эвристическими, они используют принципы, которые трудно описать формально, что приводит к перечисленным далее трудностям.

- **Классификация**. Описанные методы можно разделить на группы по различным признакам, однако единой четкой классификации на настоящее время не существует, что приводит к путанице в терминах. Например, часто даже специалисты, знакомые с алгоритмами колонии муравьев, роя частиц, роя пчел, не знают значения объединяющего их термина "роевой интеллект". Хотя перечисленные примеры являются просто различными реализациями метода роевого интеллекта.
- **Терминологии**. Нет четкого разделения понятий "метод" и "алгоритм". В русскоязычной литературе принято использовать термины "метод роя частиц", но "генетический алгоритм", "муравьиный алгоритм" и одновременно "метод колонии муравьев". Кроме того, некоторые понятия, общие по своему смыслу для многих стохастических методов, такие как "фитнесс-функция", "миграция", используют только для некоторых отдельных алгоритмов, для других то же самое понятие обозначают другими терминами.
- **Объяснение** стохастических методов неспециалистам, обусловленное непривычными для технических наук терминами, взятыми из описания природы:

"феромон", "популяция", "мутация", "приспособленность", "гибель", "рой" и др.

Перечисленные трудности можно разрешить путем системного описания стохастических алгоритмов с введением четких общих обозначений, иерархии алгоритмов, выделением для каждого из них структуры и элементов. В настоящей статье предложенный подход применен к алгоритмам роевого интеллекта.

Концепция роевого интеллекта

Рассмотрим основные положения концепции роевого интеллекта. Этот термин (*Swarm Intelligence*) был введен Херардо Бени и Ван Цзином в 1989 г. [3].

Рой можно определить как децентрализованную систему, состоящую из множества простых однообразных элементов, косвенно взаимодействующих друг с другом и с окружающей средой для достижения предопределенной цели. Примерами таких систем могут служить колония муравьев, рой пчел, косяк рыб, стая птиц. Концепция роевого интеллекта построена на аддитивном, синергическом эффекте, который проявляется при объединении агентов в систему. Элементы принято называть агентами или "бойдами" (*boids* [3]).

Считая, что "метод" является понятием более высокого уровня абстракции, чем "алгоритм", который можно понимать как реализацию метода, можно считать роевой интеллект методом, а различные реализации его положений алгоритмами (алгоритм колонии муравьев, алгоритм роя частиц и т. д.). Такой подход используется в англоязычной литературе [2].

Перед рассмотрением алгоритмов роевого интеллекта как системы введем следующие обозначения:

$f(\mathbf{X})$ — скалярная целевая функция, для которой требуется найти максимальное или минимальное значение; для определенности дальнейшего описания будем полагать, что если это не оговорено особо, то требуется найти максимальное значение;

$\mathbf{X}$ — вектор варьируемых параметров, от которых зависит целевая функция;

D — область допустимых значений $\mathbf{X}$, $D \subset R^{|\mathbf{X}|}$ — пространство поиска решений;

$G(\mathbf{X})$ — функция, задающая ограничения на $\mathbf{X}$;

$|S|$ — число агентов роя;

$S = \{s_1, s_2, ..., s_{|S|}\}$ — множество всех агентов роя;

$\mathbf{X}_{ij}$ — вектор варьируемых параметров i-го агента на j-й итерации алгоритма, иными словами, положение агента, его позиция.

$\mathbf{X}_{ij}^{best}$ — наилучшее положение i-го агента от 1-й до j-й итерации алгоритма;

$\mathbf{X}^{opt}$ — наилучшее значение вектора варьируемых параметров (оптимальное решение);

f^{opt} — наилучшее значение целевой функции;

$\varphi(\mathbf{X}) = f(\mathbf{X})$ — значение фитнесс-функции в положении $\mathbf{X}$;

$\mathbf{X}_j^{best}$ — наилучшее значение вектора варьируемых параметров, которое было получено среди всех агентов от 1-й до j-й итерации алгоритма;

$\mathbf{X}_{final}^{best}$ — наилучшее значение вектора варьируемых параметров, найденное роем к моменту завершения работы (квазиоптимальное решение), использование эвристических методов не гарантирует равенство $\mathbf{X}_{final}^{best}$ и $\mathbf{X}^{opt}$;

$S_j^{best}(n)$ — n агентов роя, занимающих на j-й итерации наилучшие положения, т. е. таких, что $\varphi(\mathbf{X}_{ij}) \geqslant \geqslant \varphi(\mathbf{X}_{kj})$, $s_i \in S_j^{best}(n)$, $s_k \in S/S_j^{best}(n)$;

$\boldsymbol{C}_j = c(S_j) = c(\mathbf{X}_{1j}, \mathbf{X}_{2j}, ..., \mathbf{X}_{|s|j}, \varphi(\mathbf{X}_{1j}), \varphi(\mathbf{X}_{2j}), ..., \varphi(\mathbf{X}_{|s|j}))$ — точка "центра тяжести" роя, некоторая позиция X, полученная усреднением положений всех агентов с учетом их фитнесс-функций.

Рассматривается задача максимизации

$$f^{opt} = f(\mathbf{X}^{opt}) = \max_{\mathbf{X} \in D} f(\mathbf{X}). \quad (1)$$

Схема алгоритмов роевого интеллекта включает в себя следующие этапы.

1. **Генерация начальных состояний агентов**. Некоторым образом в пространстве поиска распределяются агенты. Номер итерации $j = 1$.

2. **Вычисление фитнесс-функции** для каждого из агентов. Для большинства алгоритмов роевого интеллекта для этого необходимо просто вычислить $\varphi(\mathbf{X}_{ij}) = = f(\mathbf{X}_{ij})$. Для других, например, колонии муравьев, вычисление целевой функции возможно только после реализации эвристического поведения агентов, для них шаг 2 при $j = 1$ пропускается.

3. **Миграция** (перемещение). На этом шаге реализуется главная особенность алгоритмов роевого интеллекта — выполнение каждым агентом своих действий на основании:

- своих собственных правил;
- правил, реализующих косвенный обмен с другими особями;
- стохастического поведения.

4. **Проверка условия завершения**. Если условие выполнено, процесс завершается, значение $\mathbf{X}_{final}^{best}$ будет конечным результатом, иначе происходит переход к шагу 2 (с увеличением номера итерации j на единицу).

Условиями завершения работы алгоритма могут быть: достижение заданного числа итераций; нахождение решения не хуже некоторого заранее заданного; стагнация процесса (когда $\mathbf{X}_j^{best}$ не улучшается на протяжении заданного числа итераций). Кроме того, при реализации данных алгоритмов следует предусмотреть возможность пользователя прерывать процесс

поиска в любой момент или продолжать его неограниченно долгое время.

Анализ шага 3 позволяет определить, является ли рассматриваемый алгоритм алгоритмом роевого интеллекта. Как правило, в формулах, определяющих поведение агентов, есть либо элемент, связанный одним ($\mathbf{X}_j^{best}$) или несколькими наилучшими положениями ($S_j^{best}(n)$), которые были найдены всем роем (косвенный обмен информацией о своих решениях), либо "центр тяжести" (C_j). Например, в алгоритме роя частиц для организации обмена информацией между агентами используется $\mathbf{X}_j^{best}$, в алгоритме роя пчел — $S_j^{best}(n)$, в обезьяньем поиске — C_j. Более сложными является муравьиный алгоритм, где для косвенного обмена опытом используется граф, веса дуг которого изменяются в зависимости от того, насколько хорошее решение было получено при движении по ним. В эволюционных же алгоритмах косвенный обмен отсутствует, он заменяется процессом отбора агентов с наилучшим значением фитнесс-функции. К таким алгоритмам относятся генетический алгоритм, сорняковый алгоритм, алгоритм растущих деревьев. Так, например, канонический алгоритм бактериальной оптимизации не относится к алгоритмам роевого интеллекта, но модификация, в которую введены правила притяжения и отталкивания бактерий друг от друга [3], является алгоритмом роевого интеллекта, о чем свидетельствует и название "алгоритм **роения** бактерий".

Системное описание роевого интеллекта

В определении роевого интеллекта указано, что рой является системой. Чтобы доказать это, рассмотрим некоторые определения системы из работы [1] с точки зрения соответствия им введенного понятия "рой".

- "Система есть нечто целое". Очевидно, что рой, разделенный на отдельных агентов, не сможет эффективно решать задачи оптимизации.
- "Система есть организованное множество". Рой является организованным множеством агентов, но его особенностью является самоорганизация, оператор организации в общем виде описан выше шагами 1—4 (алгоритм работы роя).
- "Система есть множество вещей, свойств и отношений". Под вещами можно понимать агентов, под свойствами — их характеристики, положения, под отношениями — алгоритм работы роя.
- "Система есть множество элементов, образующих структуру и обеспечивающих определенное поведение в условиях окружающей среды". По сути, отличие этого определения лишь во введении понятия окружающей среды. Для роя окружающей средой является целевая функция и пространство поиска решений.
- "Система есть множество входов, множество выходов, множество состояний, характеризуемых операторами переходов и выходов". На вход роя подается задача оптимизации, состояния роя — это состояния всех его агентов на j-й итерации, операторы переходов задаются алгоритмом работы роя, выходом является наилучшее найденное значение вектора варьируемых параметров $\mathbf{X}_{final}^{best}$.

Увеличение сложности определений соответствует углублению описания роевого интеллекта, поэтому понятие "рой" в концепции роевого интеллекта действительно соответствует понятию "система".

Выделим общие для всех алгоритмов роевого интеллекта понятия на основании системного описания, используя термины из работы [1].

1. Элемент. "Под элементом принято понимать простейшую неделимую часть системы" [1]. Элементом роя является агент (частица, муравей, бактерия, пчела и т. д.). Агент характеризуется положением, которое меняется от итерации к итерации, т. е. $\mathbf{X}_{ij}$ характеризуется значением фитнесс-функции $\varphi(\mathbf{X}_{ij})$.

2. Подсистема. "Система может быть разделена на элементы не сразу, а последовательным расчленением на подсистемы, которые представляют собой компоненты более крупные, чем элементы, и в то же время более детальные, чем система в целом" [1]. Концепция роевого интеллекта допускает выделение групп агентов. Возможны следующие ситуации:

- связи между агентами в группе сильнее, чем между агентами из различных групп (островковая модификация алгоритма роя частиц);
- выделяется группа агентов с особым поведением, например, не использующих информацию о предыдущем опыте, для предотвращения преждевременной сходимости алгоритма (разведчики в методе роя пчел).

3. "Структура — это совокупность элементов и связей между ними" [1]. Структура роевых систем задается правилами поведения агентов и обменом информацией. Особенностью таких систем является отсутствие центра, который управлял бы агентами напрямую.

4. Связь между агентами происходит путем косвенного обмена опытом (шаг 3 алгоритма работы роя).

5. Состояние. "Понятием "состояние" обычно характеризуют мгновенную фотографию, "срез" системы" [1]. Под состоянием понимается состояния всех агентов, состояние объекта для косвенного обмена опытом (наилучшее значение $\mathbf{X}_j^{best}$, веса дуг графа с феромоном и т. д.), текущее значение наилучшего

вектора варьируемых параметров и наилучшее найденное значение целевой функции.

6. Поведение. "Если система способна переходить из одного состояния в другое, то говорят, что она обладает поведением" [1]. Поведение роя задается рассмотренным алгоритмом работы роя, который переводит множество агентов из одного состояния в другое.

7. Внешняя среда. "Под внешней средой понимается множество элементов, которые не входят в систему, но изменение их состояния вызывает изменение поведения системы" [1]. В данном случае средой является решаемая задача, т. е. целевая функция и ограничения.

8. Равновесие, устойчивость и развитие. С точки зрения роевого интеллекта, эти понятия связаны со сходимостью алгоритмов. Сходимость алгоритмов роевого интеллекта является отдельной областью исследований. Алгоритмы роевого интеллекта должны обеспечивать развитие от начального случайного набора решений до оптимального или близкого к оптимальному решению. При этом высокая устойчивость может привести к быстрой сходимости поиска как к оптимальному решению, так и к некоторому локальному экстремуму, далекому от оптимального. Низкая устойчивость приводит к повышению вероятности попасть в область глобального экстремума, но при этом повышается вероятность и преждевременного выхода из этой области вследствие недостаточно подробного ее исследования.

9. Целью роя является нахождение такого значения вектора варьируемых параметров **X**, который обеспечил бы наилучшее значение целевой функции $f(\mathbf{X})$ (решение задачи (1)). Достижение цели не гарантируется, но эффективность стохастической оптимизации связана с нахождением близких к оптимальным решений за короткое время в тех задачах, где другие методы неприменимы или малоэффективны.

Примеры описания алгоритмов роевого интеллекта по предложенной схеме

На основании проведенного анализа можно дать формализованное определение алгоритмам роевого интеллекта. Алгоритм роевого интеллекта является системой вида

$$SI = \{S,\ M,\ A,\ P,\ I,\ O\}, \qquad (2)$$

где S — множество агентов; M — объект для обмена опытом между агентами, чаще всего некоторая матрица (может быть вектор), к которому имеют доступ по определенным правилам все агенты роя, и который используется в A; A — правила создания, поведения, модификации агентов; P — параметры (эвристические

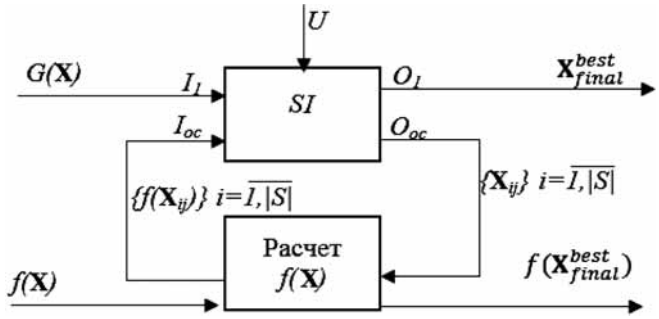


Схема связей роя с внешней средой и надсистемой

коэффициенты), использующиеся в формулах из A; $I = \{I_1, I_{oc}\}$; I_1 — вход системы, на который подается целевая функция, ограничения; I_{oc} — вход для обратной связи; $O = \{O_1, O_{oc}\}$; O_1 — выход (наилучшее найденное решение задачи); O_{oc} — выход для обратной связи.

Система SI взаимодействует с внешней средой (целевой функцией $f(\mathbf{X})$ и ограничениями $G(\mathbf{X})$). При реализации надсистемой для SI является чаще всего программное приложение, включающее в себя как минимум подсистемы ввода (чтения, загрузки) задач, ввода условий остановки, вывода (записи) результатов решения. Управлявшие воздействия на рой (запуск, остановка, задание параметров и числа агентов) обозначим U. Описанная схема приведена на рисунке.

Рассмотрим алгоритмы роя частиц и муравьиной колонии по предложенной схеме.

Алгоритм роя частиц

Алгоритм роя частиц (*Particle Swarm Optimization — PSO*) был изначально разработан для моделирования социального поведения и основан на поведении стай птиц [2]. В 1995 г. Дж. Кеннеди и Р. Эберхарт предложили этот метод для решения задач непрерывной глобальной оптимизации.

Стая птиц всегда действует скоординированно, каждая из птиц действует согласно простым правилам, следит за другими птицами и согласует свое движение с ними. Найдя источник пищи, птица сообщает о нем всей стае. Именно этот факт создает коллективное поведение и роевой интеллект. Источники пищи обычно расположены случайным образом и одной птице очень сложно быстро найти их. Только в том случае, если птицы будут обмениваться информацией, вся стая сможет выжить.

При переходе к модели, слово "птица" заменено на слово "частица" и введены следующие положения:

- частицы существуют в мире, где время дискретно;
- частицы оценивают свое положение с помощью фитнесс-функции;

- каждая частица знает позицию в пространстве, в которой она нашла наибольшее количество пищи (своя наилучшая позиция);
- каждая частица знает позицию в пространстве, в которой найдено наибольшее количество пищи среди всех позиций, в которых были все частицы (общая наилучшая позиция);
- частицы имеют тенденцию стремиться к лучшим позициям, в которых были сами и к общей наилучшей позиции;
- частицы случайным образом меняют свою скорость, так что описанная тенденция определяет лишь усредненное движение частиц;
- частицы обладают инерцией, поэтому их скорость в каждый момент времени зависит от скорости в предыдущий момент;
- частицы не могут покинуть заданную область поиска.

Основная идея алгоритма заключается в перемещении частиц в пространстве решений. Каждая частица "помнит" наилучшую точку в пространстве решений, в которой была, и стремится в нее вернуться, но подчиняется также закону инерции и имеет склонность к небольшому стохастическому изменению направления движения. Однако этих правил недостаточно для перехода к системе, так как не заданы связи между элементами. В качестве связи используется так называемая общая память, суть которой в том, что каждая частица знает координаты наилучшей точки среди всех, в которых была любая частица роя. Таким образом, наилучшее решение, найденное роем, в каждый момент времени известно всем его агентам. В итоге на движение частицы влияют стремление к своему наилучшему положению, стремление к наилучшему среди всех частиц положению, инерционность и случайные отклонения.

Согласно формуле (2), алгоритм роя частиц $PSO = \{S, M, A, P, I, O\}$.

Рассмотрим элементы системы подробнее.

Множество агентов (частиц) $S = \{s_1, s_2, ..., s_{|S|}\}$, $|S|$ — число частиц. На j-й итерации i-я частица характеризуется состоянием $s_{ij} = \{\mathbf{X}_{ij}, \mathbf{V}_{ij}, \mathbf{X}_{ij}^{best}\}$, где $\mathbf{X}_{ij} = \{x_{ij}^1, x_{ij}^2, ..., x_{ij}^l\}$ — вектор варьируемых параметров (положение частицы); $\mathbf{V}_{ij} = \{v_{ij}^1, v_{ij}^2, ..., v_{ij}^l\}$ — вектор скоростей частицы; $\mathbf{X}_{ij}^{best} = \{b_{ij}^1, b_{ij}^2, ..., x_{ij}^l\}$ — наилучшее по значению фитнесс-функции положение частицы среди всех положений, которые она занимала в процессе работы алгоритма от 1-й до j-й итераций, l — число варьируемых параметров.

Вектор $\mathbf{M} = \mathbf{X}_j^{best}$ — наилучшее значение вектора варьируемых параметров, которое было получено среди всех частиц от 1-й до j-й итерации алгоритма. Этот вектор обеспечивает косвенный обмен опытом между частицами.

Алгоритм A вписывает механизмы функционирования роя частиц. Существуют различные модификации этого алгоритма. Далее представлено описание базового алгоритма.

1. Генерация начальных положений и скоростей ($j = 1$):

$$\mathbf{X}_{i1} = \text{rand}(G(\mathbf{X})),\ i = \overline{1, |S|},$$

где $\text{rand}(G(\mathbf{X}))$ — вектор равномерно распределенных случайных величин, отвечающих ограничениям на область поиска;

$$\mathbf{V}_{i1} = \text{rand}(\mathbf{V}_{\min}, \mathbf{V}_{\max}),\ i = \overline{1, |S|},$$

где $\text{rand}(\mathbf{V}_{\min}, \mathbf{V}_{\max})$ — вектор равномерно распределенных случайных величин в диапазоне $(\mathbf{V}_{\min}, \mathbf{V}_{\max})$.

$$\mathbf{X}_{i1}^{best} = \mathbf{X}_{ij},\ i = \overline{1, |S|}.$$

Произвольно выбирается наилучшая позиция (при вычислении фитнесс-функций будет определена действительно наилучшая позиция):

$$\mathbf{X}_1^{best} = \mathbf{X}_{11}.$$

2. Вычисление фитнесс-функций и определение наилучшего положения:

$$\begin{gathered}\mathbf{X}_{ij}^{best} = \mathbf{X}_{ij},\ \varphi(\mathbf{X}_{ij}^{best}) < \varphi(\mathbf{X}_{ij}),\ i = \overline{1, |S|},\\ \mathbf{X}_i^{best} = \mathbf{X}_{ij},\ \varphi(\mathbf{X}_j^{best}) < \varphi(\mathbf{X}_{ij}),\ i = \overline{1, |S|}.\end{gathered} \tag{3}$$

Вычисление $\varphi(\mathbf{X}) = f(\mathbf{X})$ происходит во внешней среде, с помощью обмена данными по обратной связи (I_{oc}, O_{oc}) (см. рисунок).

3. Перемещения частиц:

$$\begin{gathered}\mathbf{V}_{ij+1} = \mathbf{V}_{ij}\omega + \alpha_1(\mathbf{X}_{ij}^{best} - \mathbf{X}_{ij})rnd_1 +\\ \alpha_2(\mathbf{M} - \mathbf{X}_{ij})\overline{rnd}_2,\ i = \overline{1, |S|},\end{gathered} \tag{4}$$

$$\mathbf{V}_{ij+1} = \begin{cases}\mathbf{V}_{ij+1},\ \mathbf{V}_{\min} \leqslant \mathbf{V}_{ij+1} \leqslant \mathbf{V}_{\max};\\ \mathbf{V}_{\min},\ \mathbf{V}_{ij+1} \leqslant \mathbf{V}_{\min};\\ \mathbf{V}_{\max},\ \mathbf{V}_{ij+1} \geqslant \mathbf{V}_{\max} = 0,\end{cases}$$

$$i = \overline{1, |S|},$$

$$\mathbf{X}_{ij+1} = \begin{cases}\mathbf{X}_{ij} + \mathbf{V}_{ij+1},\ G(\mathbf{X}_{ij} + \mathbf{V}_{ij+1}) = 1;\\ \mathbf{X}_{ij},\ G(\mathbf{X}_{ij} + \mathbf{V}_{ij+1}) = 0,\end{cases}$$

$$i = \overline{1, |S|},$$

где rnd_1 и rnd_2 — случайные числа, равномерно распределенные в интервале $[0,1)$; $G(\mathbf{X})$ здесь использу-

ется как предикат, который показывает, принадлежит ли **X** области допустимых значений *D*.

4. Если на *j*-й итерации выполнено условие остановки, то значение $\mathbf{X}_{final}^{best} = \mathbf{X}_{j}^{best}$ подается на выход O_1. Иначе происходит переход к шагу 2.

Вектор $\mathbf{P} = \{\alpha_1, \alpha_2, \omega\}$ — коэффициенты алгоритма *A*, которые используются в формуле (4) и влияют на перемещения частиц в пространстве поиска. Коэффициенты α_1 и α_2 определяют, соответственно, степень учета индивидуального и группового опыта агентов. Коэффициент ω характеризует инерционные свойства частиц.

Идентификаторы *I* и *O* — описанные выше вход и выход роя, которые не зависят от реализации алгоритмов роевого интеллекта.

Алгоритм колонии муравьев

Алгоритм колонии муравьев (*Ant Colony Optimization — ACO*) предложен в начале 1990-х гг. бельгийским ученым Марко Дориго [5]. Описание муравьиного алгоритма приводилось в журнале "Программная инженерия" [6]. Муравьи при движении выделяют особое вещество (феромон), отмечая свой путь. На коротких путях к пище феромона остается больше, чем на длинных, поэтому короткие пути привлекают большее число муравьев и поиск кратчайшего маршрута быстро сходится к квазиоптимальному пути.

В основе модели лежит косвенный обмен информацией между агентами через особую среду, представленную взвешенным графом. Для решения задач оптимизации алгоритмом колонии муравьев необходимо представить задачу в виде нахождения кратчайшего маршрута на взвешенном графе.

В отличие от алгоритма роя частиц, который описывается как алгоритм для нахождения экстремумов непрерывных функций, муравьиный алгоритм в классической формулировке решает комбинаторные задачи, например задачу коммивояжера. Поэтому вектором варьируемых параметров в муравьином алгоритме обычно является последовательность узлов в графе, оптимальный способ обхода которого нужно найти (в задаче коммивояжера — последовательность городов в искомом маршруте). Тем не менее представленная в формуле (2) структура полностью сохраняется.

Согласно формуле (2), алгоритм колонии муравьев $ACO = \{S, \mathbf{M}, A, P, I, O\}$. Для удобства будем считать, что решается задача минимизации (так как муравьи ищут наикратчайший путь).

Множество агентов (муравьев) $S = \{s_1, s_2, ..., s_{|S|}\}$, $|S|$ — число муравьев. На *j*-й итерации *i*-й муравей характеризуется состоянием $s_{ij} = \{\mathbf{X}_{ij}, \mathbf{T}_{ij}\}$, где $\mathbf{X}_{ij} = \{x_{ij}^1, x_{ij}^2, ..., x_{ij}^l\}$ — вектор варьируемых параметров (последовательность узлов графа); $\mathbf{T}_{ij} = \{t_{ij}^1, t_{ij}^2, ..., t_{ij}^l\}$ — вектор булевых переменных, которые показывают, был ли *l*-й узел посещен *i*-м муравьем на *j*-й итерации (в начале каждой итерации все его компоненты равны 0).

Граф $\mathbf{M} = \{\mathbf{F}, \mathbf{R}\}$ — граф, каждая дуга которого имеет два веса: переменный (количество феромона) и постоянный (характеризующий задачу; например, в задаче коммивояжера это расстояние между городами). Поэтому граф **M** можно представить в виде двух графов с одинаковыми структурами, но разными весами дуг (**F** и **R**):

$$\mathbf{F} = \begin{pmatrix} \tau_{11} & \cdots & \tau_{1l} \\ \vdots & \ddots & \vdots \\ \tau_{l1} & \cdots & \tau_{ll} \end{pmatrix}, \quad \mathbf{R} = \begin{pmatrix} r_{11} & \cdots & r_{1l} \\ \vdots & \ddots & \vdots \\ r_{l1} & \cdots & r_{ll} \end{pmatrix},$$

где $\tau_{k_1k_2}$ — количество феромона на дуге, соединяющей k_1-й и k_2-й узлы графа **F**, $r_{k_1k_2}$ — вес (длина) дуги, соединяющей k_1-й и k_2-й узлы графа **R**, при этом в общем случае $\tau_{k_1k_2} \neq \tau_{k_2k_1}$ и $r_{k_1k_2} \neq r_{k_2k_1}$.

Алгоритм *A* описывает механизмы функционирования колонии муравьев.

1. Генерация начальных положений. В зависимости от задачи, каждый муравей может быть создан в случайном узле графа или в заданном. Если муравей помещен в узел *k*, то

$$x_{i1}^1 = k, \quad t_{i1}^1 = 1.$$

На каждую дугу наносится некоторое ненулевое количество феромона

$$\tau_{ij} = \tau_{\min}.$$

2. В алгоритме колонии муравьев на первой итерации ($j = 1$) второй шаг пропускается, так как для вычисления фитнесс-функций необходимо выполнить перемещения агентов.

Для остальных шагов выполняются следующие действия:

- вычисление целевых функций:

$$\varphi(\mathbf{X}_{ij}) = f(\mathbf{X}_{ij}),\ i = i = \overline{1, |S|},$$

после каждого вычисления целевой функции происходит сравнение ее значения с $\varphi(\mathbf{X}_{final}^{best})$ аналогично формуле (3).

- определение количества феромона, которое нужно нанести на дугу, соединяющую узлы k_1 и k_2 (феромон для *i*-го муравья наносится только на те дуги, которые вошли в маршрут $\mathbf{X}_{ij}$):

$$\Delta\tau_{ij}^{k_1k_2} = \begin{cases} \dfrac{\gamma}{\varphi(x_{ij})}, \text{ в } \mathbf{X}_{ij}\ k_1 \text{ следует сразу за } k_2, \\ 0, \text{ иначе,} \end{cases}$$

$$i = \overline{1, |S|},\ k_1 = \overline{1, l},\ k_2 = \overline{1, l},$$

• пересчет количества феромона на всем графе с учетом испарения и ограничений:

$$\theta = \rho\tau_j^{k_1k_2} + \Delta\tau_{ij}^{k_1k_2}, \qquad k_1 = \overline{1, l},\ k_2 = \overline{1, l},$$

$$\tau_j^{k_1k_2} = \begin{cases} \theta, \ \theta \geqslant \tau_{\min}; \\ \tau_{\min}, \ \theta < \tau_{\min}. \end{cases}$$

3. Перемещения агентов. В каждом узле k муравей выбирает, в какой из еще не посещенных узлов перейти. При этом вероятность перехода в m-й узел равна (индексы i и j, определяющие агента и итерацию, здесь опущены)

$$P_m = \begin{cases} \dfrac{(\tau_{km})^{\alpha}(\eta(r_{km}))^{\beta}}{\sum\limits_{z=1,\, t_z=0}^{l} ((\tau_{kz})^{\alpha}(\eta(r_{kz}))^{\beta})}, & t_m = 0, \\ 0, & t_m = 1. \end{cases}$$

Здесь $\eta(r_{km})$ — некоторая функция от веса дуги, в простейшем случае $\eta(r_{km}) = r_{km}$.

После вычисления вероятностей с помощью розыгрыша по жребию происходит определение, в какой узел m должна переместиться частица. При этом $t_m = 1$, номер узла добавляется в маршрут частицы. После окончания обхода вектор **T** обнуляется.

4. Если на j-ой итерации выполнено условие остановки, то значение $\mathbf{X}_{final}^{best} = \mathbf{X}_j^{best}$ подается на выход O_1. Иначе происходит переход к шагу 2.

Вектор $\mathbf{P} = \{\alpha, \beta, \gamma, \rho\}$ — коэффициенты алгоритма A. Коэффициент α определяет степень влияния количества феромона на дуге на вероятность того, что муравей выберет эту дугу. Коэффициент β определяет степень влияния веса дуги графа на вероятность ее выбора. Коэффициент γ — коэффициент интенсивности выделения феромона. Коэффициент ρ влияет на испаряемость феромона, принимает значения от 0 (нет испарения) до 1 (испаряется до минимального уровня после каждой итерации).

Идентификаторы I и O — описанные выше вход и выход роя, которые не зависят от реализации алгоритмов роевого интеллекта.

Анализ результатов системного описания

Четкое описание алгоритмов роевого интеллекта позволило выделить общую схему алгоритмов, их системные свойства, определить терминологию. Отличительной чертой алгоритмов роевого интеллекта являются косвенный обмен информацией между агентами и децентрализованность (отсутствие центра, который управлял бы агентами, например, проводил бы их отбор для уничтожения или перевода в следующие итерации). На основании предложенного критерия можно выделить из множества стохастических алгоритмов те, которые относятся к роевому интеллекту (описания алгоритмов взяты из работ [2, 4, 5]):

- алгоритм гравитационного поиска;
- алгоритм динамики формирования рек;
- алгоритм колонии муравьев;
- алгоритм роения бактерий;
- алгоритм роя пчел;
- алгоритм роя частиц;
- алгоритм светлячков;
- гармонический поиск;
- интеллектуальные капли воды;
- обезьяний алгоритм;
- поиск косяком рыб;
- стохастический диффузионный поиск;
- тасующий алгоритм роя лягушек;
- улучшенный алгоритм поиска кукушки;
- электромагнитный поиск.

Заключение

В работе дано системное определение различным алгоритмам роевого интеллекта, описаны общие для них принципы, термины, схема работы. Выделены входы, выходы роевых систем, их элементы и подсистемы. Дана схема взаимодействия роя с внешней средой и надсистемой.

Введенное определение позволило сформулировать общую для всех алгоритмов роевого интеллекта схему описания и указать признаки, позволяющие выделять среди множества стохастических алгоритмов те, которые относятся к методу роевого интеллекта.

Предложенная схема описания может стандартизировать и упростить описание и программную реализацию алгоритмов роевого интеллекта, их объяснение и изучение.

Список литературы

1. **Красов А. В.** Теория информационных процессов и систем [Электронный ресурс] URL: http://www2.mts-sut.ru/kafedr/ibts/doc/tips.pdf.

2. **Карпенко А. П.** Популяционные алгоритмы глобальной оптимизации. Обзор новых и малоизвестных алгоритмов // Информационные технологии. Приложение. 2012. № 7. 32 с.

3. **Beni G., Wang J.** Swarm Intelligence in Cellular Robotic Systems [Электронный ресурс] // Proc. of NATO Advanced Workshop on Robots and Biological Systems. Tuscany, Italy. 26—30 June, 1989. URL: http://www-users.york.ac.uk/~jo115/paper-reviews/SwarmIntelligenceinCellularRoboticSystems.pdf.

4. **Kennedy J., Eberhart R.** Particle Swarm Optimization [Электронный ресурс] // Proc. of IEEE International Conference on Neural Network. Piscataway, NJ. 27 November — 01 December, 1995. P. 1942—1948. URL: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber = 488968.

5. **Dorigo M., Maniezzo V., Colorni A.** The Ant System: Optimization by a colony of cooperating agents [Электронный ресурс] // IEEE Transactions on Systems, Man, and Cybernetics — Part B. 1996. V. 26, No. 1. URL: ftp://iridia.ulb.ac.be/pub/mdorigo/journals/IJ.10-SMC96.pdf.

6. **Матренин П. В., Секаев В. Г.** Оптимизация адаптивного алгоритма муравьиной колонии на примере задачи календарного планирования // Программная инженерия. 2013. № 4. С. 34—40.

Указатель статей, опубликованных в журнале "Программная инженерия" в 2013 году

CONTENTS

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4

Дизайнер *Т. Н. Погорелова*. Технический редактор *Е. М. Патрушева*. *Корректор Е. В. Комиссарова*

Сдано в набор 03.10.2013 г. Подписано в печать 19.11.2013 г. Формат 60×88 1/8. Заказ PI1213
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
119071, г. Москва, Ленинский пр-т, д. 19, стр. 1.