ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

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

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

СОДЕРЖАНИЕ

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

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

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

Vol. 8
N 4
2017

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

CONTENTS

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

УДК 004.457+004.031.2+004.382.2 DOI: 10.17587/prin.8.147-160

Д. С. Николаев, науч. сотр., e-mail: dmitry.s.nikolaev@gmail.com, **В. В. Корнеев,** д-р техн. наук, проф., гл. науч. сотр., e-mail: korv@rdi-kvant.ru, ФГУП "Научно-исследовательский институт "Квант", Москва

Использование механизмов контейнерной виртуализации в высокопроизводительных вычислительных комплексах с системой планирования заданий SLURM

Рассмотрено использование механизма контейнерной виртуализации для изоляции параллельных заданий на бездисковых вычислительных узлах высокопроизводительных вычислительных комплексов под управлением системы планирования заданий. Предложен подход к созданию виртуальных пользовательских окружений с заданными требованиями к вычислительным ресурсам в многопользовательских гетерогенных высокопроизводительных вычислительных комплексах. Представлена оценка взаимного влияния изолированных заданий разного типа, разделяющих один вычислительный модуль.

Ключевые слова: *высокопроизводительные вычислительные комплексы, SLURM, CoreOS, Docker, MPI, InfiniBand, CUDA*

Введение

Архитектура современных высокопроизводительных вычислительных комплексов (ВВК) предполагает их построение из вычислительных модулей, объединенных высокоскоростной коммуникационной сетью. Модули состоят из большого числа универсальных и специализированных процессорных ядер: многоядерных универсальных процессоров и специализированных ускорителей. Например, модули включают в себя графические процессоры, программируемые логические интегральные схемы (ПЛИС), многопоточные ускорители типа Intel Xeon Phi. Высокопроизводительные вычислительные комплексы могут оснащаться вычислительными модулями различных типов и поколений, что делает структуру комплекса гетерогенной.

На ранних этапах развития ВВК вычислительные модули при планировании параллельных заданий не разделялись между несколькими заданиями. В настоящее время с учетом тенденции к увеличению общего состава и объемов вычислительных ресурсов модуля выделение только части универсальных и специализированных ядер одного вычислительного модуля одному параллельному заданию приводит к фрагментации ресурсов ВВК и, как следствие, к неизбежному простою значительной части тех ресурсов, которые не используются этим заданием. Если разделить ядра одного вычислительного модуля между несколькими параллельными заданиями, возникает вопрос об обеспечении изоляции одновременно выполняющихся на одном модуле параллельных заданий. Причина в том, что задания выполняются под управлением одной операционной системы (ОС) и, следовательно, могут конфликтовать при запросах к памяти, к внешним устройствам и др. По мере увеличения числа ядер в вычислительных модулях необходимость дефрагментации вычислительных ресурсов будет все более актуальной. В работе [1] количество простаивающих процессоров ВВК в силу отсутствия механизмов дефрагментации оценивается от 10 до 16 %.

Одним из подходов к обеспечению изоляции параллельных заданий является использование механизмов контейнерной виртуализации, которые позволяют запускать на одном вычислительном модуле изолированные друг от друга параллельные задания. Механизмы контейнерной виртуализации позволяют контролировать распределение вычислительных ресурсов ВВК между заданиями, появляются как возможность монопольного выделения части ресурсов одному заданию, так и механизмы разделения ресурсов между разными заданиями.

В программировании под изоляцией процессов подразумевается предотвращение доступа одного процесса к участку памяти другого. Концептуальные положения такой изоляции позволяют повысить безопасность ОС, обеспечивая различные уровни привилегий и ограничивая области памяти, доступные различным программам. Изоляция процессов является необходимым условием надежности и безопасности многозадачной многопользовательской системы. Один процесс не должен иметь возможности вмешаться в работу другого или получить доступ к его данным, ни по причине случайной ошибки,

ни намеренно. Этот факт означает, что процессы не должны ничего знать о существовании друг друга.

Под изоляцией заданий понимается как изоляция процессов их реализации, так и отсутствие возможности монопольного захвата одним заданием вычислительных ресурсов других, а также контроль конкуренции за вычислительные ресурсы между процессами заданий. В результате такой конкуренции производительность вычислений снижается, однако не выше некоторого наперед заданного допустимого значения. К вычислительным ресурсам можно отнести процессоры, память, порты коммуникационной среды, ускорители.

Несмотря на всю привлекательность механизмов контейнерной виртуализации, они пока не нашли широкого применения в области высокопроизводительных вычислений. В работе [2] показано, что накладные расходы на контейнерную виртуализацию при выполнении высокопроизводительных заданий практически отсутствуют. При этом время запуска подготовленных контейнеров на вычислительных модулях сопоставимо с временем запуска обычного процесса.

В настоящей статье предложен способ применения легковесной операционной системы CoreOS [3] с открытым исходным кодом на базе ядра Linux со встроенной технологией контейнерной виртуализации Docker [4] для запуска изолированных параллельных заданий на гетерогенных ВВК с бездисковыми вычислительными модулями. Описан макет вычислительного кластера с запуском распределенных задач в изолированных, преднастроенных пользователем окружениях с помощью системы планирования заданий SLURM [5].

1. Применяемые технологии и существующие решения

Контейнерная виртуализация или виртуализация на уровне ОС позволяет запускать изолированные и безопасные виртуальные машины на одном физическом модуле, но не позволяет запускать ОС с ядрами, отличными по типу от ядра базовой ОС. При виртуализации на уровне ОС не существует отдельного слоя гипервизора. Вместо этого сама базовая ОС отвечает за разделение аппаратных ресурсов между несколькими виртуальными машинами (контейнерами) и за поддержку механизмов, позволяющих обеспечить их независимость друг от друга.

В настоящее время осуществляются активные действия по внедрению технологии контейнерной виртуализации в область высокопроизводительных вычислений.

В США в Национальном научном вычислительном центре энергетических исследований (NERSC) ведут разработку проекта Shifter [6]. В основе этого проекта лежит использование заранее определенных пользователем образов (контейнеров) с настроенным программным обеспечением. В качестве этих образов могут выступать образы контейнеров из репозитория Docker. Изначально проект Shifter разрабатывали для суперкомпьютеров Cray XC-30 с системой планирования заданий ALPS, однако в настоящее время планируется его реализация под Linux-системы с различными системами планирования заданий. Непосредственно для запуска контейнеров Docker не используется. Вместо этого образы автоматически извлекаются из формата Docker и преобразовываются к общему формату. Далее с помощью программного обеспечения из состава системы Shifter настраивается окружение на вычислительных узлах и запускается задача пользователя. Применение общего формата для пользовательских образов позволило отказаться от адаптации Docker для платформы Cray и повысить универсальность системы за счет большего числа поддерживаемых форматов пользовательских образов. Большое внимание было также уделено безопасности системы, контейнер рассматривается как расширение приложения, контроль над которым пользователь уже имеет. Таким образом система Shifter упрощает и оптимизирует возможность создания и вызова пользовательских изолированных окружений. Однако она не добавляет дополнительных возможностей кроме тех, которые уже доступны пользователю. В настоящее время система Shifter не поддерживает запуска нескольких заданий пользовательских образов на одном узле, отсутствует поддержка многослойной файловой системы OverlayFS, а также поддержка запуска распределенных MPI-приложений. Реализация таких функций анонсирована в направлениях дальнейших исследований.

В Германии в исследовательском центре Немецкий электронный синхротрон (Deutsches Elektronen-Synchrotron — DESY) на вычислительном кластере Maxwell реализуют проект по запуску изолированных в Docker-контейнерах заданий с помощью системы планирования SLURM [7]. Предполагается, что задания могут быть предназначены как для работы в пределах одного вычислительного модуля, так и для случая распределенного по нескольким вычислительным модулям задания. Во втором случае в контейнерах требуется наличие библиотек MPI, стека InfiniBand и сервиса SSH-доступа. Для взаимодействия контейнеров распределенного MPI-приложения, расположенных на разных вычислительных модулях, используется сетевой стек, разделяемый с базовой системой. Такой подход ограничивает число одновременно работающих распределенных заданий одним экземпляром на один вычислительный модуль. Кроме того, работающие контейнеры задания разделяют пространство имен пользователей с базовой системой, благодаря чему контейнеры запускаются и работают с правами пользователя, запустившего задание. Однако такой подход требует наличия зарегистрированных учетных записей пользователей в базовой системе. В настоящее время проект находится в стадии бета-тестирования.

В России в Институте механики сплошных сред УрО РАН также исследуют возможности примене-

ния контейнерной виртуализации Docker для запуска задач на суперкомпьютерах. В работе [8] показана возможность использования технологии Docker для доставки, развертывания и запуска вычислительных задач на суперкомпьютере в изолированных контейнерах с использованием немодифицированной системы запуска задач. Описан способ интеграции контейнеров и системы SLURM. Ключевой особенностью решения является запуск в контейнерах только вычислительной задачи, в то время как инфраструктура запуска задач остается снаружи на вычислительных узлах. В отличие от предыдущего проекта, в данном проекте не разделяется пространство имен пользователей с базовой системой, а используются персонифицированные для каждого пользователя контейнеры. Этот факт означает, что в контейнерах с заданием создается пользователь, идентификаторы которого совпадают с идентификаторами пользователя, запускающего задание. Аналогично предыдущему проекту, в данном проекте для взаимодействия контейнеров распределенного MPI-приложения, расположенных на разных вычислительных модулях, используется сетевой стек, разделяемый с базовой системой. При этом сохраняются ограничения на возможность запуска нескольких контейнеров с распределенными заданиями на одном вычислительном модуле.

В качестве альтернативы технологии контейнерной виртуализации можно рассмотреть проект Дрезденского технического университета по запуску виртуальных машин на высокопроизводительном кластере с помощью SLURM. В работе [9] авторы используют гипервизор KVM для управления виртуальными машинами, которые создаются на вычислительных модулях гетерогенного суперкомпьютерного кластера под управлением системы планирования SLURM. Задачами в данном случае выступают виртуальные машины. В результате использования гипервизора удалось добиться полной изоляции пользовательского программного стека от системного. Однако при этом были отмечены повышенные накладные расходы по сравнению с контейнерной виртуализацией.

В перечисленных выше исследованиях не уделяется достаточного внимания повышению эффективности использования ресурсов ВВК за счет снижения их фрагментации. Однако технология контейнерной виртуализации позволяет в ряде случаев использовать простаивающие ресурсы вычислительных модулей без заметного взаимного влияния выполняющихся на них заданий. В настоящей работе рассмотрен способ запуска изолированных в контейнерах заданий на бездисковых вычислительных модулях. Большое внимание уделено возможности запуска нескольких изолированных заданий на одном вычислительном модуле, в том числе — с использованием ускорителей и ресурсов высокоскоростной сети. Рассмотрен способ запуска распределенных заданий с использованием виртуальных сетей для каждого задания.

2. Описание предлагаемой технологии

При разработке технологии запуска распределенных заданий в изолированных, преднастроенных пользователем окружениях на гетерогенном вычислительном кластере учитывались следующие требования:

- использование открытого свободно распространяемого программного обеспечения;
- работа совместно с существующими системами планирования заданий;
- масштабируемость вычислительного кластера и работа с различными аппаратными конфигурациями;
- разделение одного вычислительного модуля между несколькими изолированными заданиями.

Исходя из перечисленных требований, в качестве ОС для вычислительных модулей была выбрана ОС CoreOS. Выделение вычислительных ресурсов и управление заданиями осуществляется с помощью системы планирования заданий SLURM. Различные аппаратные конфигурации вычислительных модулей поддерживаются с помощью механизма модулей ядра для загрузки драйверов устройств. Для обеспечения одновременной работы нескольких распределенных изолированных заданий используются возможности создания виртуальных сетей для взаимодействия между контейнерами.

В данной работе приведена структура макета гетерогенного вычислительного кластера, дано описание организации общего хранилища, приведены требования к пользовательским контейнерам с заданиями, описан процесс подготовки вычислительных модулей к выполнению заданий, запуск заданий на выполнение и процесс приведения вычислительных модулей в исходное состояние.

Также приведены оценки производительности макета кластера при работе с изолированными заданиями разного типа, оценки безопасности использования механизма контейнерной виртуализации и дальнейшие направления исследований в данной области.

2.1. Доработка CoreOS для интеграции с SLURM и запуска на бездисковых узлах

Операционная система CoreOS является легковесной ОС с открытым исходным кодом на базе ядра Linux. Она предназначена для создания инфраструктуры компьютерных кластеров, особое внимание уделено автоматизации, упрощению внедрения приложений, обеспечению безопасности, надежности и масштабируемости. В качестве операционной системы CoreOS предоставляет минимальные функциональные возможности, необходимые для развертывания приложений внутри программных контейнеров, средства обнаружения сервисов и передачи настроек. В ОС CoreOS отсутствует пакетный менеджер. Подразумевается, что все приложения работают внутри собственных контейнеров, которые реализованы с помощью средств контейнерной

виртуализации Docker или rkt [10]. В качестве демона инициализации (init) в CoreOS используется системный менеджер systemd [11], тесно интегрированный с сервисами CoreOS. Эта ОС используется для построения легко и гибко масштабируемых кластеров.

В стандартной поставке CoreOS отсутствуют компоненты системы планирования заданий, необходимые для функционирования ВВК. Так как система планирования заданий должна управлять заданиями, изолированными в контейнерах, то ее собственные компоненты не могут быть размещены в контейнерах, как это подразумевается в методологии использования ОС CoreOS. В основе этой ОС лежит дистрибутив Gentoo Linux. Использование пакетной базы Gentoo позволяет добавлять необходимые функциональные возможности в ОС CoreOS.

С помощью комплекта средств разработки для CoreOS [12] были добавлены компоненты, необходимые для функционирования системы планирования заданий SLURM версии 16.05 и создан модифицированный образ для загрузки ОС по протоколу PXE. Конфигурирование вычислительных модулей осуществляется централизованно с использованием технологии начальной инициализации Ignition [13] и облачной конфигурации Cloud-Config [14]. Директивы Ignition исполняются на этапе загрузки ядра ОС вычислительного модуля и осуществляют первичную настройку программного обеспечения. Директивы Cloud-Config исполняются на этапе инициализации системы менеджером systemd, на котором загружаются необходимые модули ядра, подключаются общие каталоги, конфигурируются и запускаются системные службы.

В результате выполнения перечисленных выше действий был получен образ ОС CoreOS с возможностью реализации централизованного конфигурирования для PXE-загрузки вычислительных модулей с интегрированным программным обеспечением SLURM версии 16.05 и его программным окружением. Такое программное обеспечение является открытым, свободно распространяемым и широко используется в высокопроизводительных вычислительных комплексах. Открытые и хорошо документированные программные интерфейсы позволяют пользователю добавлять собственные компоненты для планирования, запуска и контроля заданий.

2.2. Реализация виртуальной сети контейнеров

Для выполнения изолированных параллельных вычислений в качестве механизма контейнерной виртуализации используется программное обеспечение Docker версии 1.11. Начиная с версии 1.9 в Docker появилась возможность использования виртуальных сетей для взаимодействия между контейнерами, в том числе расположенными на разных физических узлах. Для работы с виртуальными сетями необходима поддержка со стороны ядра базовой системы, которая реализована в ядрах Linux начиная с версии 3.16.

Для каждого задания пользователя создается своя виртуальная сеть, и все контейнеры, участвующие в выполнении этого задания, подключаются к ней. Все контейнеры в рамках одного задания могут взаимодействовать друг с другом, при этом контейнеры из разных заданий не имеют возможности взаимодействовать по сети (рис. 1, см. третью сторону обложки).

Для управления виртуальными сетями используется открытое программное обеспечение из проекта Calico, разрабатываемое компанией "Metaswitch Networks" [15]. В данном случае модули Calico используются в качестве подключаемых к Docker сетевых программных модулей. Модули Calico реализованы в виде контейнеров `calico-node`, реализующих механизм виртуальных сетей и механизм управления IP-адресами, которые, соответственно, взаимодействуют с сетевыми библиотеками Docker. Указанные контейнеры работают на каждом узле кластерного комплекса и обеспечивают сетевое взаимодействие между контейнерами.

Для управления конфигурационными параметрами кластера используется распределенное хранилище "ключ-значение". В качестве распределенного хранилища в ОС CoreOS используется сервис `etcd`, который запускается на каждой машине кластера CoreOS и обеспечивает общий доступ практически ко всем конфигурационным данным в масштабе всего кластера. Внутри `etcd` хранятся настройки служб, их текущие состояния, конфигурация самого кластера и т. д. Сервис `etcd` позволяет: хранить данные иерархически (хранилище древовидно), подписываться на изменения ключей или целых директорий; задавать для ключей и директорий ключей значения TTL; атомарно изменять или удалять ключи, упорядоченно хранить их. Параметры виртуальных сетей также находятся в этом хранилище. Демон `dockerd` подключается к данному хранилищу при старте.

Для создания виртуальных сетей в кластере первоначально добавляется пул адресов, из которого будут назначаться адреса контейнерам в указанной виртуальной сети, например:

```
calicoctl pool add 10.0.0.0/16
```

Таких пулов может быть несколько с различными настройками. После создания пула средствами Docker с использованием сетевых программных модулей Calico создаются виртуальные сети, например:

```
docker network create --driver calico --ipam-driver calico net1
docker network create --driver calico --ipam-driver calico net2
docker network create --driver calico --ipam-driver calico net3
```

В представленном примере создано три сети в одном пуле адресов. При этом контейнеры, созданные в разных сетях, не будут иметь возможности взаимо-

действовать по сети. При создании и запуске контейнера указывается сеть, к которой он принадлежит:

```
# Создание контейнеров на базовой системе 1
docker run --net net1 --name cont1 -tid centos-mpi
/bin/bash
docker run --net net2 --name cont2 -tid centos-mpi
/bin/bash
# Создание контейнеров на базовой системе 2
docker run --net net1 --name cont3 -tid centos-mpi
/bin/bash
docker run --net net2 --name cont4 -tid centos-mpi
/bin/bash
```

В данном примере контейнеры из сети **net1** могут обращаться друг к другу по именам **cont1.net1** и **cont3.net1** соответственно. Аналогичные обращения реализуются для сети **net2**. Схема полученных виртуальных сетей представлена на рис. 2.

Для каждого контейнера из его метаданных можно узнать назначенный ему IP-адрес, например:

```
docker inspect --format "{{
.NetworkSettings.Networks.net2.IPAddress }}" cont1
```

Взаимодействие между базовыми узлами кластера осуществляется по IP-адресам контейнеров, при этом контейнеры из разных виртуальных сетей не могут взаимодействовать друг с другом.

При запуске задания на выполнение, на этапе пролога создается виртуальная сеть с уникальным именем, например, именем сети может быть присвоенный идентификатор задания. Каждому создаваемому контейнеру назначается уникальное имя и указывается виртуальная сеть, с которой он будет взаимодействовать. На этапе создания контейнеров формируется файл со списком их адресов и числом выделенных процессорных ядер, формат данного файла соответствует формату `machinefile` библиотеки OpenMPI. В дальнейшем этот файл используется при запуске распределенной задачи пользователя в созданных контейнерах. Предполагается, что на одном вычислительном модуле будет располагаться не более одного контейнера одного задания, что позволит минимизировать межконтейнерное взаимодействие по виртуальным сетям.

При выполнении задания контейнеры могут использовать высокоскоростную сеть кластера, в данном макете кластера реализуется высокоскоростная сеть InfiniBand. Для использования высокоскоростной сети контейнерам предоставляется доступ к соответствующим устройствам и добавляется необходимое программное обеспечение стека высокоскоростной сети. Виртуальная сеть в данном случае используется для инициализации и управления выполнением задания.

Для подключения к выделенным заданию контейнерам на узле доступа для каждого задания создается управляющий контейнер, который подключен к виртуальной сети задания (рис. 1, см. третью сторону обложки). Для управления заданием используется протокол SSH с аутентификацией по имеющимся ключам пользователя.

2.3. Организация общего хранилища

Общее хранилище реализовано на основе протокола сетевого доступа к файловым системам NFS. Структура общего хранилища представлена на рис. 3.

Каталог `common` является общим для всех пользователей на всех вычислительных модулях. В каталоге `home` располагаются домашние каталоги пользователей. Они доступны на всех вычислительных модулях и используются для SSH-аутентификации пользователей в контейнерах изолированных заданий. В каталоге `docker` находятся образы и данные контейнеров Docker, которые расположены в отдельных каталогах для каждого вычислительного модуля. При загрузке они подключаются в корневую файловую систему вычислительного модуля по стандартному пути `/var/lib/docker`.

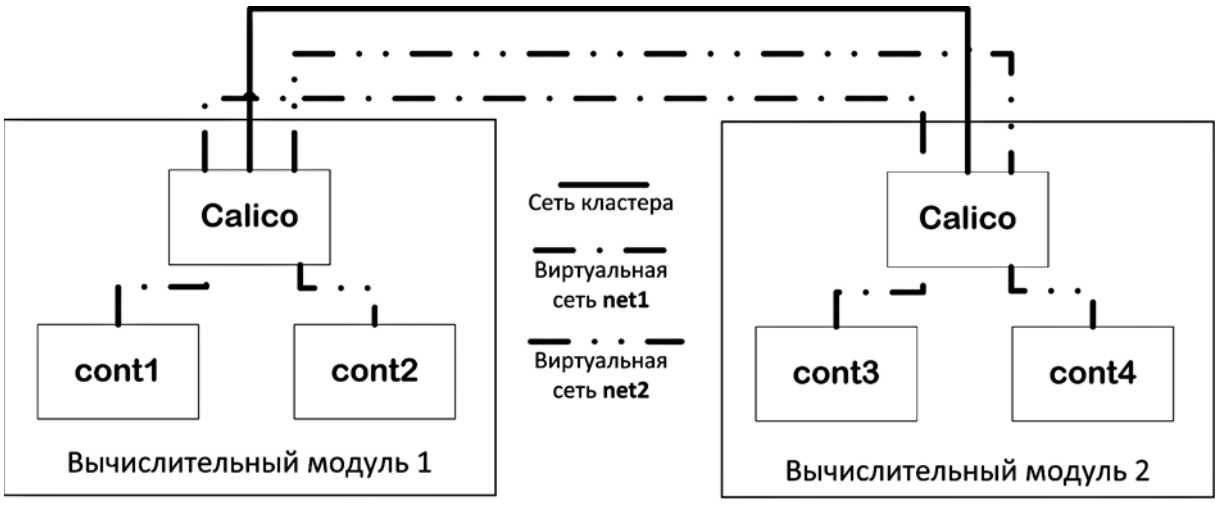


Рис. 2. Пример организации виртуальных сетей между контейнерами

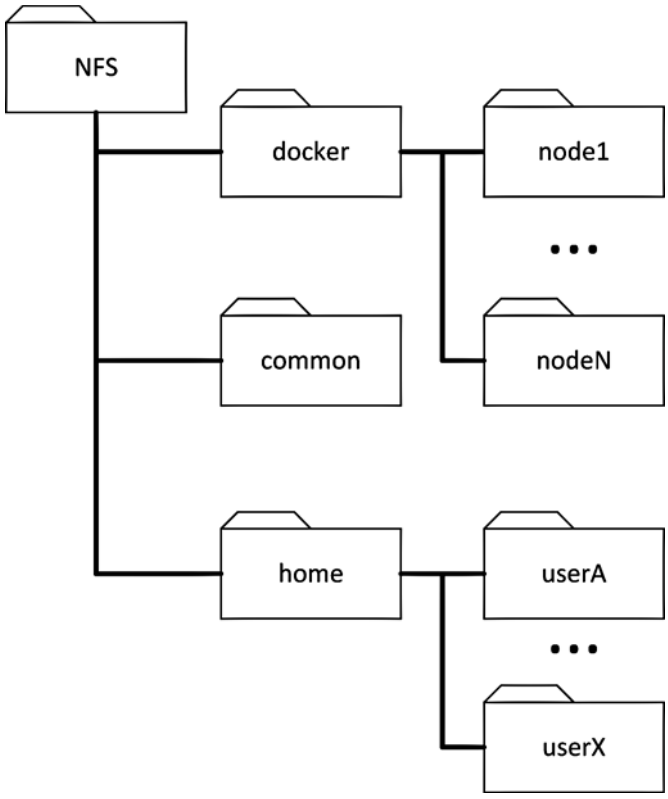


Рис. 3. Структура общего хранилища

2.4. Изоляция драйверов устройств в контейнерах

Одной из основных задач ОС является управление аппаратной частью. Ядро Linux ОС CoreOS содержит большое число встроенных драйверов и драйверов, представленных в виде модулей ядра. Модули ядра оформлены в виде отдельных файлов и для их подключения (на этапе загрузки или впоследствии) необходимо выполнить отдельную команду подключения модуля, после чего будет обеспечено управление соответствующим устройством. Если необходимость в использовании устройства исчезает, модуль можно выгрузить из памяти (отключить). Использование модулей обеспечивает большую гибкость, так как каждый такой драйвер может быть переконфигурирован без остановки системы.

Для функционирования гетерогенной ВВК с вычислительными модулями под управлением ОС CoreOS, оснащенными графическими ускорителями, потребуется установка специализированных драйверов устройств, отсутствующих в стандартном наборе драйверов. С помощью средства контейнерной виртуализации Docker существует возможность работы с так называемыми привилегированными контейнерами, т. е. контейнерами, имеющими полный доступ ко всем устройствам вычислительного модуля. После установки драйвера в таком контейнере соответствующие устройства будут отображены в каталоге `/dev/` вычислительного модуля и окажутся доступны для работы с ними из непривилегированных контейнеров с задачами пользователей. Пример создания и запуска контейнера для установки графического драйвера:

```
docker run --privileged=true -i -t -v /dev:/dev
ubuntu16.04-nvdrv-cuda /bin/bash
```

После загрузки соответствующего модуля ядра в этом контейнере, в каталоге `/dev/` вычислительного модуля будут доступны устройства графического ускорителя:

```
$ ls /dev/ | grep nv
nvidia-uvm
nvidia-uvm-tools
nvidia0
nvidia1
nvidia2
nvidia3
nvidiactl
```

Приложения пользователя, которым требуются ресурсы графических ускорителей, могут быть запущены в непривилегированных контейнерах с указанием требуемых устройств, например:

```
docker run -ti --name cuda0 --device
/dev/nvidia0:/dev/nvidia0 --device
/dev/nvidiactl:/dev/nvidiactl --device /dev/nvidia-
uvm:/dev/nvidia-uvm ubuntu16.04-nvdrv-cuda /bin/bash
```

Аналогичным образом настраиваются и выделяются ресурсы высокоскоростной коммуникационной среды. Далее приведен пример выделения ресурсов RDMA для изолированного в непривилегированном контейнере MPI-приложения пользователя:

```
docker run --name cont1 --
device = /dev/infiniband/uverbs0 --
device = /dev/infiniband/rdma _ cm --ulimit memlock=-1
centos _ ssh _ ompi _ ib /usr/sbin/sshd –D
```

Таким образом, использование привилегированных контейнеров для размещения в них драйверов устройств в виде подключаемых модулей ядра позволяет использовать общий унифицированный образ ОС для вычислительных модулей с различной конфигурацией оборудования в гетерогенном ВВК. Для обновления изолированного в контейнере драйвера нет необходимости модифицировать образ ОС. Достаточно отключить имеющийся в контейнере драйвер, провести обновление и подключить его снова.

2.5. Требования к контейнеру с заданием пользователя

В контейнерах пользователь может использовать произвольный дистрибутив ОС Linux, совместимый с установленным на базовом узле ядром Linux. Выбор используемого в контейнерах программного обеспечения средств разработки, библиотек MPI, допол-

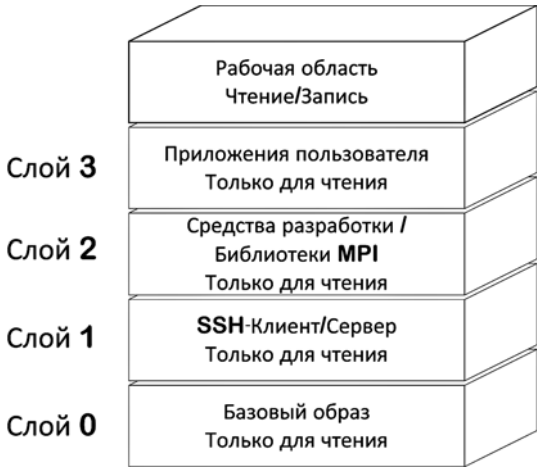


Рис. 4. Структура контейнера

нительных и вспомогательных библиотек остается за пользователем. При создании контейнера для решения задачи пользователь может использовать либо преднастроенный базовый контейнер из репозитория, либо создать собственный контейнер. Примерная структура контейнера представлена на рис. 4. Пользователь может создавать контейнеры как вручную, так и с использованием директив сценарного файла Dockerfile. В целях повышения производительности файловой системы рекомендуется создавать контейнеры с как можно меньшим числом слоев.

Подключение к контейнерам осуществляется по протоколу SSH с использованием авторизации по открытым ключам. Соответственно, в каждом контейнере должен быть установлен и настроен сервер и клиент SSH. Пользовательские ключи SSH разделяются между домашним каталогом пользователя на узле доступа и всеми контейнерами пользователя путем подключения домашнего каталога пользователя в качестве его домашнего каталога в контейнере. Пример создания и запуска контейнера с подключением каталога пользователя:

```
docker run --name cont1 -tid -v
/var/nfshome/user:/home/user centos-mpi /bin/bash
```

Если в контейнере уже существует каталог, путь которого совпадает с подключаемым (монтируемым), то в соответствии с правилами команды **mount** по этому пути будут располагаться данные из подключенного каталога.

В целях соблюдения рекомендуемого принципа "один контейнер — один процесс" демон SSH запускается сразу при старте контейнера, например:

```
docker run --name cont1 -tid -v
/var/nfshome/user:/home/user centos-mpi1 /usr/sbin/sshd
-D
```

Также команда на запуск демона SSH может быть указана в файле сценария (Dockerfile) на этапе создания контейнера:

```
CMD [„/usr/sbin/sshd", „-D"]
```

После запуска контейнера в нем выполняется команда создания пользователя, соответствующего пользователю, поставившему в очередь это задание:

```
docker exec cont1 useradd -u {UID} -g {GIU} {LOGIN}
```

Для корректной работы контейнера с общими сетевыми ресурсами от имени пользователя, запустившего задание, необходимо обеспечить соответствие логина и идентификаторов UID и GID на узле доступа и внутри контейнеров.

Ключи SSH, как правило, располагаются в домашнем каталоге пользователя **~/.ssh/** в файлах:

- **authorized _ keys** — список ключей, разрешенных к подключению;
- **id _ rsa** — закрытый ключ пользователя;
- **id _ rsa.pub** — открытый ключ пользователя;
- **known _ hosts** — список разрешенных к подключению узлов.

Для обеспечения взаимной беспарольной аутентификации между всеми контейнерами пользователя, его открытый ключ добавляется в список ключей, разрешенных к подключению. В список разрешенных к подключению узлов либо добавляются адреса всех контейнеров пользователя, либо разрешаются все подключения из заданной подсети.

Все действия по созданию, запуску и настройке контейнеров выполняются согласно "сценарному" файлу пролога задания, исполнение которого инициируется системой планирования заданий.

После настройки контейнеров взаимодействие с ними осуществляется по протоколу SSH от имени пользователя, запустившего задачу. Соответственно, у пользователя отсутствуют права суперпользователя при запуске задания внутри контейнера.

Таким образом, основным ограничением, накладываемым на пользовательский контейнер, является наличие в нем серверной и клиентской частей протокола SSH.

2.6. Составные компоненты

Рассмотрим применение представленной технологии, использующей механизмы контейнерной виртуализации для изоляции выполняющихся заданий, для создания макета гетерогенного вычислительного кластера под управлением системы планирования заданий. Предполагается обеспечить размещение нескольких независимых изолированных заданий на незадействованных вычислительных ресурсах модулей и оценить взаимное влияние заданий разного типа, разделяющих один вычислительный модуль.

Рассматриваемый макет вычислительного кластера (рис. 5) состоит из бездисковых вычислительных модулей, узла доступа, узла общего хранилища, узла удаленной загрузки, управляющего узла и высокоскоростной коммуникационной среды. В качестве основной ОС для узлов кластера выбрана система

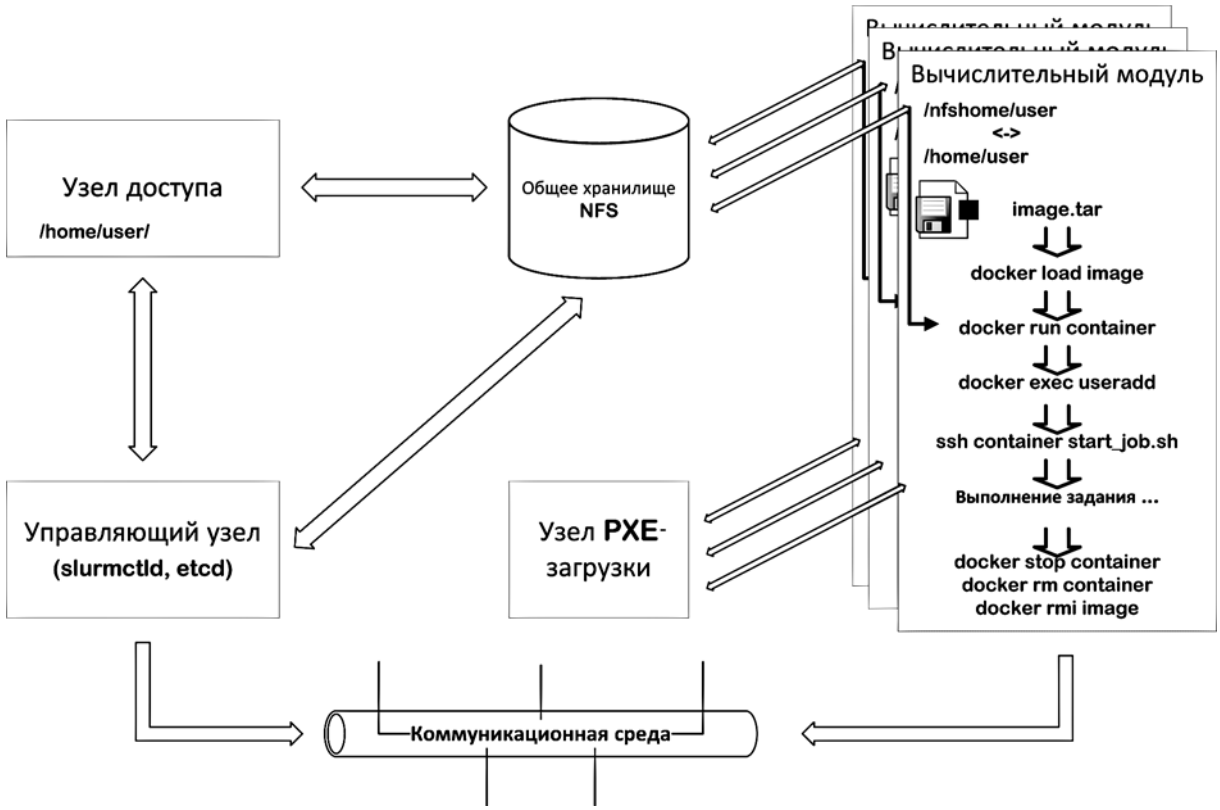


Рис. 5. Основные составные компоненты макета кластера

CentOS 7.1, представляющая собой свободно распространяемый аналог системы Red Hat Linux. Бездисковые вычислительные модули работают под управлением модифицированной версии ОС CoreOS выпуска 1185 с ядром Linux версии 4.7.3.

Бездисковые модули загружаются с помощью общего узла удаленной загрузки и обеспечивают общую конфигурацию программного обеспечения. Общая конфигурация программного обеспечения снижает затраты, связанные с администрированием и техническим обслуживанием системы. Использование бездисковых вычислительных модулей ведет к уменьшению общей стоимости системы, снижению энергозатрат, повышению надежности, которая связана с меньшим числом точек отказа. Оправданным является использование бездисковых вычислительных модулей в тех случаях, когда информация на них не должна храниться постоянно.

Узел доступа предназначен для подготовки, настройки и постановки в очередь на выполнение контейнеров с заданиями пользователей. Общее хранилище доступно на каждом вычислительном узле кластера и служит для хранения домашних каталогов пользователей, а также содержит репозиторий преднастроенных образов контейнеров.

Управляющий узел обеспечивает работу служебных компонентов: системы планирования заданий SLURM, распределенного хранилища "ключ—значение" etcd, системы удаленной PXE-загрузки бездисковых вычислительных модулей кластера. На вычислительных модулях кроме служебных компонентов механизма контейнерной виртуализации Docker работают клиентские части служебных компонентов управляющего узла, а также контейнеры, обеспечивающие функционирование виртуальных сетей для заданий пользователя. Наличие обладающего спецификой программного обеспечения для выполнения заданий пользователя на вычислительных узлах не требуется. Все программные зависимости задания должны быть упакованы в контейнер.

В ходе исследований был проведен ряд экспериментов, основной целью которых являлась оценка возможности применения средств контейнерной изоляции совместно с системой планирования заданий на гетерогенном вычислительном кластере с бездисковыми вычислительными модулями. Эксперименты проводили на испытательном стенде, состоящем из узла доступа, объединяющего в себе сервер удаленной загрузки и сервер сетевой файловой системы, двух вычислительных модулей, объединенных высокоскоростной сетью InfiniBand, и вычислительного модуля с двумя двухпроцессорными графическими ускорителями (рис. 6). Сетевое взаимодействие узлов макета кластера при этом осуществляется по стандарту Gigabit Ethernet. Модули, объединенные высокоскоростной сетью InfiniBand, содержат по два 4-ядерных центральных процессорных устройств (ЦПУ) Intel Xeon CPU X5472 3.00 GHz, 32 Гбайта

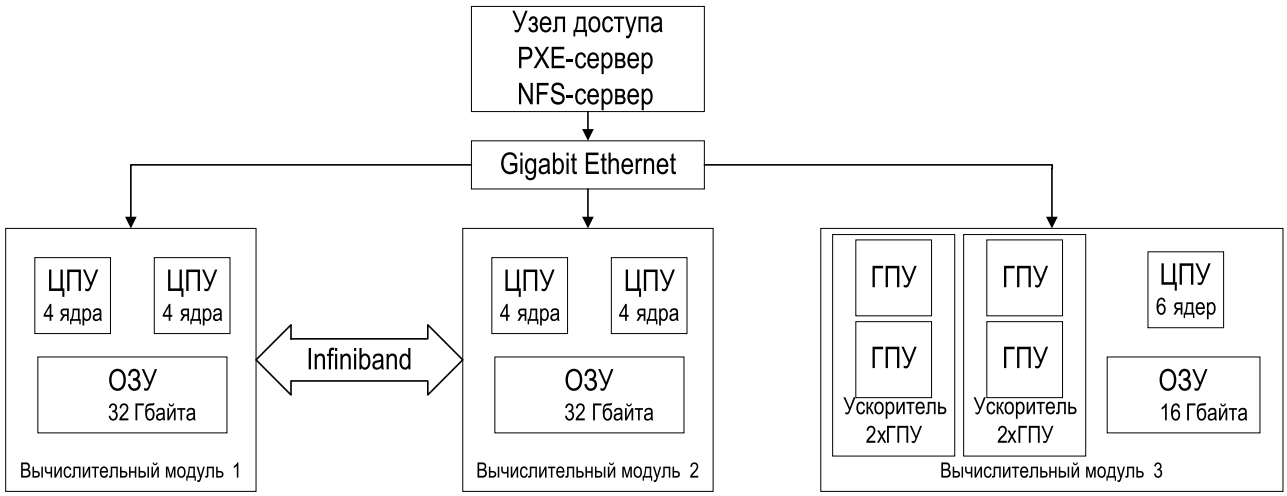


Рис. 6. Схема испытательного стенда

оперативной памяти, два сетевых адаптера Gigabit Ethernet и адаптер высокоскоростной сети InfiniBand Mellanox MT25204. Модуль с высокопроизводительными графическими процессорными устройствами (ГПУ) содержит ЦПУ Intel Core i7-3930K 3.20 GHz, 16 Гбайта оперативной памяти, два сетевых адаптера Gigabit Ethernet и два двухпроцессорных графических ускорителя NVIDIA GeForce GTX 690.

Для хранения контейнеров и их образов на вычислительных модулях используется подключаемый из общего сетевого хранилища каталог `/var/lib/docker`. Для каждого вычислительного модуля данный каталог должен быть уникальным, так как в нем сохраняется состояние контейнеров, работающих на каждом из модулей. Таким образом, на общем сетевом хранилище размещаются данные всех контейнеров и их образов для каждого вычислительного модуля. На общем сетевом хранилище эти данные располагаются по пути `/var/dockercommon/<имя _ узла>/` (см. рис. 3).

Домашние каталоги пользователей подключаются в каталог `/var/nfshome/` на каждый вычислительный узел. В дальнейшем планируется, что они могут быть подключены в пользовательский контейнер по стандартному пути, например, при создании контейнера с помощью опции `-v /var/nfshome/user:/home/user`. Наличие учетных записей пользователей на вычислительных модулях не требуется. Схема размещения домашнего каталога пользователя, расположенного на общем хранилище, приведена на рис. 7. Размещение домашнего каталога пользователя в контейнере с заданием будет соответствовать его размещению на узле доступа.

Коммуникационная среда состоит из сетей Ethernet и высокоскоростной сети InfiniBand. Сеть Ethernet используется для управления работой кластера, размещения и управления заданиями. Высокоскоростная сеть InfiniBand используется для межпроцессного взаимодействия выполняемых заданий пользователей.

2.7. Подготовка вычислительного узла

По умолчанию система планирования заданий SLURM сконфигурирована на выделение ресурсов заданиям с точностью до целого вычислительного узла. Такая конфигурация означает, что не используемые заданием ресурсы на выделенном ему узле будут простаивать. Как следствие, уменьшается эффективность использования ресурсов ВВК и повышается их фрагментация. К потребляемым ресурсам можно отнести процессоры (ядра), память, внешние ускорители и т. п.

На рассматриваемом испытательном стенде система планирования заданий SLURM сконфигурирована с использованием встраиваемого модуля для выделения потребляемых ресурсов `select/cons _ res`. Такой модуль

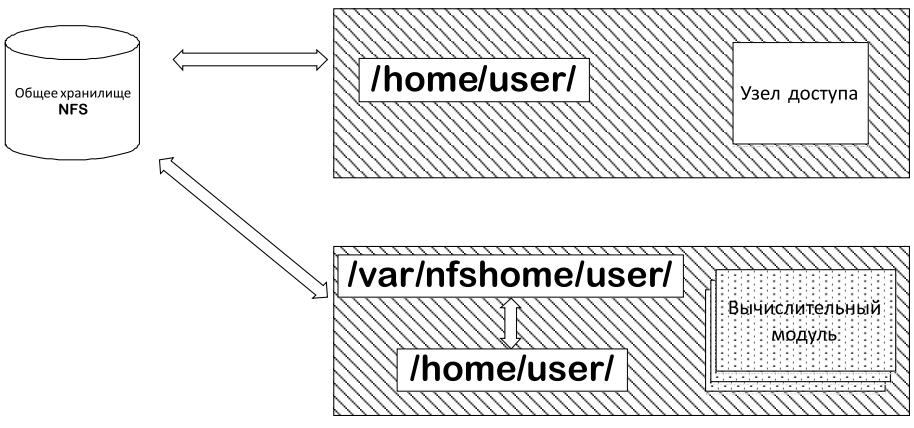


Рис. 7. Пример размещения домашнего каталога пользователя

позволяет более точно выделять потребляемые ресурсы. Для проведения экспериментов в качестве потребляемых ресурсов были выбраны вычислительные ядра центрального процессора, параметр `CR _ Core` для модуля выделения потребляемых ресурсов.

Системой планирования заданий выделяется востребованное по запросам число вычислительных узлов. Для этих целей используется команда **salloc**, служащая для выделения таких ресурсов, выполнения указанного в команде сценария и освобождения ресурсов по завершению, например:

```
salloc –n12 start _ job.sh image.tar
```

В данном случае запрашивается выделение 12 ядер для выполнения задания в контейнерах из образа **image.tar**. Директивы сценарного файла **start _ job.sh** осуществляют подготовку, запуск и последующее удаление контейнеров на каждом вычислительном узле. Подготовка осуществляется в рамках пролога задания.

Подготовка вычислительного узла к запуску контейнера на выполнение заключается в загрузке требуемого образа в локальное хранилище:

```
docker load < /home/user/images/image.tar
```

Далее создается виртуальная сеть для выполняемого задания:

```
docker network create --driver calico --ipam-driver
calico user _ net1
```

В представленном примере задается имя сети **user _ net1**. Посредством распределенного хранилища etcd параметры данной сети будут доступны на всех вычислительных узлах кластера. С помощью утилиты `scontrol` из состава программного обеспечения SLURM определяется, какие вычислительные модули и ядра на них были выделены для отдельного задания. Список выделенных ядер ЦПУ помещается в переменную окружения `CORES`.

Далее на каждом из выделенных вычислительных узлов создаются и запускаются контейнеры с заданием с указанием процессорных ядер, назначенных системой планирования:

```
docker run --name user _ cont1 --net user _ net1 --
cpuset-cpus="$CORES" -tid -v /home/user:/home/user
image /usr/sbin/sshd -D
```

В каждый создаваемый контейнер подключается домашний каталог пользователя. В качестве точки входа указывается запуск демона SSH. При создании контейнера ему присваивается адрес в указанной виртуальной сети. Этот адрес сохраняется в домашнем каталоге пользователя на вычислительном узле, в файле с именем контейнера. В дальнейшем этот файл используется для формирования списка контейнеров, участвующих в выполнении параллельного задания:

```
docker inspect --format "{{
.NetworkSettings.Networks.user _ net.IPAddress }}"
user _ cont1 > user _ cont1.addr
```

Вычисляется также число ядер, выделенных на данном вычислительном модуле. После создания и запуска всех контейнеров будет сформирован файл **image _ hosts** со списком IP-адресов всех контейнеров, созданных для исполнения задания, и с указанием числа ядер, выделенных каждому контейнеру:

```
user _ cont1 slots=8
user _ cont2 slots=4
```

Формат данного файла соответствует формату `machinefile` библиотеки OpenMPI.

В созданном и запущенном контейнере с помощью команды из сценарного файла, запускаемого системой планирования заданий, создается пользователь с именем и идентификаторами, совпадающими с именем и идентификаторами пользователя, поставившего задание в очередь, например:

```
docker exec user _ cont1 useradd -u 1001 -g 1001 user
```

На узле доступа создается головной контейнер, принадлежащий к назначенной заданию виртуальной сети, для управления изолированным заданием, например:

```
docker run --name user _ cont0 --net user _ net1 -tid
-v /home/user:/home/user image /usr/sbin/sshd –D
```

Адрес головного контейнера заносится в переменную окружения `HEAD _ IP` и в дальнейшем используется для управления заданием.

На данном этапе подготовка контейнеров к выполнению задания на выделенных вычислительных узлах завершена. Подключившись к созданному системой планирования головному контейнеру, пользователь будет иметь интерактивный доступ к востребованным им по запросу ресурсам ВВК с преднастроенным им самим программным окружением. Если интерактивный доступ при этом не требуется, то запуск задания на исполнение будет осуществлен.

2.8. Запуск задачи на выполнение

После проверки того факта, что на всех выделенных вычислительных узлах запущены все требуемые контейнеры и сформирован файл **image _ hosts** со списком адресов всех участвующих в выполнении задания контейнеров, можно выполнять запуск задания пользователя. Файл **image _ hosts** располагается в домашнем каталоге пользователя на вычислительном узле и доступен во всех его запущенных контейнерах.

Запуск задания в контейнерах происходит путем выполнения в головном контейнере соответству-

ющей команды по протоколу SSH. Задание будет запущено и выполнено от имени пользователя, поставившего его в очередь:

```
ssh $HEAD_IP "mpirun --machinefile image_hosts user_mpi_task"
```

Таким образом происходит выполнение распределенного задания в изолированных контейнерах, использующих востребованные по запросу ресурсы ВВК.

2.9. Приведение вычислительного узла в исходное состояние

При успешном завершении задания результаты его работы следует сохранять в домашнем каталоге, в этом случае они будут доступны для анализа после остановки и удаления контейнеров. Если при выполнении задания происходит какой-либо сбой, выполняется принудительная остановка контейнеров пользователя и очистка вычислительного узла.

Приведение вычислительного узла в исходное состояние происходит в рамках эпилога задания и заключается:

- в остановке работающих контейнеров

```
docker stop user_cont1;
```

- в удалении остановленных контейнеров на вычислительном узле

```
docker rm user_cont1;
```

- в удалении образа контейнера из локального репозитория вычислительного узла

```
docker rmi image.
```

После остановки и удаления всех контейнеров, участвовавших в выполнении задания, удаляется виртуальная сеть данного задания:

```
docker network rm user_net1
```

Средствами распределенного хранилища параметры данной сети будут удалены на всех вычислительных модулях кластера.

На данном этапе все выделенные вычислительные модули возвращены в исходное состояние. Выполнение директив сценарного файла, инициировавшего выполнение задания, завершается, и все ресурсы, выделенные командой **salloc** системы планирования заданий, возвращаются.

3. Аспекты безопасности

При использовании контейнерной виртуализации практически отсутствуют дополнительные риски с точки зрения информационной безопасности. Контейнер можно рассматривать как обычную задачу пользователя, запускаемую от его имени с помощью системы планирования заданий. В рассматриваемом макете вычислительного кластера задания в контейнерах выполняются от имени запустившего их пользователя, без прав суперпользователя. Однако в случае запуска контейнеров, созданных пользователем, существует возможность повышения привилегий пользователя внутри контейнера. Это не должно представлять угрозу безопасности для базовой системы, так как данный суперпользователь остается полностью изолированным в своем контейнере. Его привилегии на базовом узле при этом остаются без изменений.

При работе с общими сетевыми ресурсами в контейнерах пользователю доступны только те ресурсы, которые изначально были подключены к контейнеру при его создании системой планирования заданий. Как следствие, изменение привилегий пользователя в своем контейнере никак не скажется на угрозах для общих сетевых ресурсов других пользователей.

Наибольшую опасность представляют контейнеры, запущенные в привилегированном режиме, т. е. с доступом ко всем устройствам ядра базовой системы. В этом случае, повысив привилегии в своем контейнере, пользователь может нарушить работу базовой системы, например, перезагрузить ее. Это действие представляет серьезную угрозу в том случае, если на одном вычислительном модуле выполняются контейнеры с задачами разных пользователей.

4. Оценка производительности

На рассматриваемом в настоящей статье макете гетерогенного ВВК проведены эксперименты по оценке взаимного влияния различного типа приложений, выполняющихся в отдельных контейнерах, разделяющих один вычислительный модуль.

По результатам ранее проведенных экспериментов [2] было выявлено, что наиболее заметно взаимное влияние приложений, интенсивно использующих ресурсы высокоскоростной сети. Одним из таких приложений является тест CG из пакета NAS Parallel Benchmark [16] версии 3.3. Ключевым вопросом теста CG является оценка скорости передачи данных при отсутствии какой-либо регулярности. Примером приложения, слабо зависимого от ресурсов высокоскоростной сети, является тест EP из этого же пакета. Ключевыми вопросами теста EP являются оценка максимальной производительности кластера при операциях с плавающей точкой и минимальные межпроцессорные взаимодействия.

Тестовые задания размещались в контейнерах. Каждому контейнеру назначалось по 4 ядра ЦПУ. Контейнеры одного параллельного задания размещались на разных вычислительных узлах. В ходе экспериментов были смоделированы следующие ситуации (рис. 8):

1) одно параллельное задание CG в контейнерах на двух вычислительных модулях;

2) два параллельных задания CG в контейнерах на двух вычислительных модулях одновременно;

3) одно параллельное задание EP в контейнерах на двух вычислительных модулях;

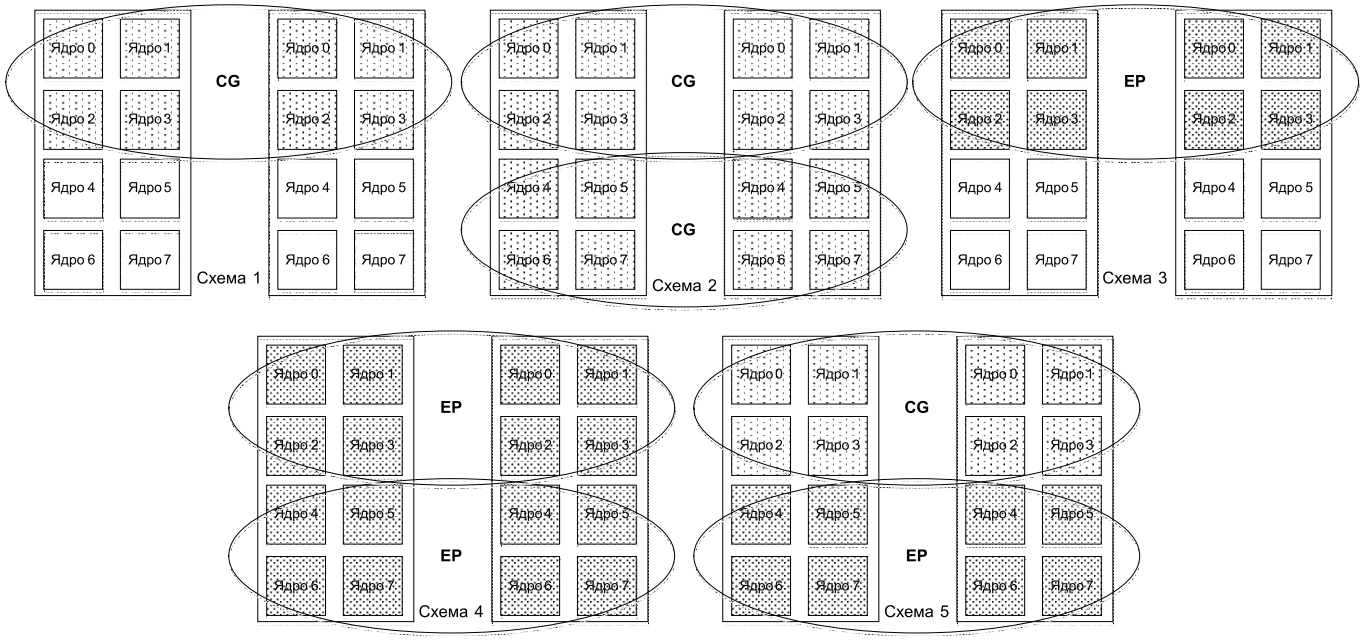


Рис. 8. Схемы размещения заданий на ядрах вычислительных модулей

4) два параллельных задания EP в контейнерах на двух вычислительных модулях одновременно;

5) одно параллельное задание CG и одно параллельное задание EP в контейнерах на двух вычислительных модулях одновременно.

На рис. 9 представлены результаты экспериментов. По данным рис. 9 видно, что два одновременно выполняющихся параллельных задания CG оказывают заметное взаимное влияние. Это обстоятельство связано с конкуренцией за ресурсы высокоскоростной сети. В то же время два одновременно выполняющихся параллельных задания EP не оказывают взаимного влияния. Таким образом, разместив в контейнерах на вычислительных модулях одно параллельное задание с высокими требованиями к ресурсам высокоскоростной сети и одно параллельное задание с низкими требованиями к ресурсам высокоскоростной сети, можно получить изолированные параллельные задания с незначительным взаимным влиянием.

Рис. 9. Время выполнения тестовых заданий с различными схемами размещения

Эксперименты также показали возможность использования высокоскоростной коммуникационной сети InfiniBand MPI-приложениями, изолированными в непривилегированных контейнерах. Это означает, что контейнерам с заданиями не предоставляется полный доступ ко всем устройствам вычислительного модуля, а выделяются только необходимые для выполнения задания устройства и ресурсы.

5. Дальнейшие направления исследований

В качестве развития представленного выше макета вычислительного кластера предполагается размещение общего хранилища, домашних каталогов пользователей и репозитория образов контейнеров на распределенной файловой системе массового параллелизма, например, Lustre [17]. Такая модификация позволит ускорить распространение готовых к выполнению контейнеров по вычислительным модулям, а также даст возможность снизить нагрузку на коммуникационную среду в случае кластера большого масштаба.

Представляется также необходимым рассмотреть способы оптимизации планирования заданий в контейнерах с поддержкой внешних ускорителей, таких как графические платы, ПЛИС, многоядерные процессорные платы.

Заключение

В настоящей работе представлен способ использования механизмов контейнерной виртуализации для создания изолированных сред с востребованным по запросу числом вычислительных ресурсов. По сравнению с цитированными ранее работами [6—9], в представленном макете гетерогенного вычислительного кластера существует возможность разделения одного вычислительного модуля между несколькими независимыми изолированными заданиями. Учет системой планирования заданий требований к вычислительным ресурсам задачи позволяет минимизировать взаимное влияние изолированных заданий, разделяющих один вычислительный модуль.

Контейнерная виртуализация может быть использована при организации высокопроизводительных вычислений как средство развертывания специализированных программных платформ, в том числе сформированных пользователем. Накладные расходы на контейнерную виртуализацию при развертывании программной платформы существенно меньше, чем на виртуализацию с помощью гипервизора.

В результате выполненных исследований показана возможность использования ОС CoreOS с технологией Docker для доставки, развертывания и запуска параллельных заданий в изолированных контейнерах на гетерогенных ВВК.

Список литературы

1. **Liu F., Weissman J.** Elastic Job Bundling: An Adaptive Resource Request Strategy for LargeScale Parallel Applications. Department of Computer Science and Engineering University of Minnesota, Minneapolis, Technical Report, 2015.

2. **Баранов А. В., Николаев Д. С.** Использование контейнерной виртуализации в организации высокопроизводительных вычислений // Программные системы: теория и приложения. 2016. № 1. С. 117—134.

3. **About** CoreOS, Inc. URL: https://coreos.com/about/ (дата обращения 15.12.2016).

4. **WHAT** IS DOCKER? URL: https://www.docker.com/what-docker (дата обращения 17.03.2016).

5. **SLURM** Workload Manager. URL: https://slurm.schedmd.com/ (дата обращения 15.12.2016).

6. **Jacobsen D. M., Canon R. S.** Contain This, Unleashing Docker for HPC. URL: http://www.nersc.gov/assets/Uploads/cug2015udi.pdf (дата обращения 15.12.2016).

7. **Yakubov S.** Using Docker container virtualization in DESY HPC environment. URL: https://indico.cern.ch/event/466991/contributions/1143628/attachments/1261303/1864379/Docker_HPC.pdf (дата обращения 23.11.2016).

8. **Щапов В. А., Денисов А. В., Латыпов С. Р.** Применение контейнерной виртуализации Docker для запуска задач на суперкомпьютере // Суперкомпьютерные дни в России. 2016. URL: http://russianscdays.org/files/pdf16/505.pdf (дата обращения 29.11.2016).

9. **Markwardt U.** Running Virtual Machines in a Slurm Batch System. URL: https://slurm.schedmd.com/SLUG15/SlurmVM.pdf (дата обращения 15.12.2016).

10. **A security-minded**, standards-based container engine URL: https://coreos.com/rkt/ (дата обращения 15.12.2016).

11. **systemd** System and Service Manager. URL: https://wiki.freedesktop.org/www/Software/systemd/ (дата обращения 15.12.2016).

12. **CoreOS** developer SDK guide. URL: https://coreos.com/os/docs/latest/sdk-modifying-coreos.html (дата обращения 15.12.2016).

13. **Ignition** 0.12.1 Documentation. URL: https://coreos.com/ignition/docs/latest/ (дата обращения 15.12.2016).

14. **Using** Cloud-Config. URL: https://coreos.com/os/docs/latest/cloud-config.html (дата обращения 15.12.2016).

15. **Project** Calico Documentation. URL: http://docs.projectcalico.org/v2.0/introduction/ (дата обращения 15.12.2016).

16. **NAS** Parallel Benchmarks. URL: https://www.nas.nasa.gov/publications/npb.html (дата обращения 15.12.2016).

17. **About** the Lustre File System. URL: http://lustre.org/about/ (дата обращения 15.12.2016).

Using Container Virtualization in High Performance Computing with SLURM Scheduler

D. S. Nikolaev, dmitry.s.nikolaev@gmail.com, **V. V. Korneev**, korv@rdi-kvant, Scientific Research Institute "Kvant", Moscow, 125438, Russian Federation

Corresponding author:

Korneev Victor V., Principal Researcher, Moscow, 125438, Russian Federation
E-mail: korv@rdi-kvant

Received on January 24, 2017
Accepted on February 06, 2017

The topic of this paper is using container virtualization technic to isolate parallel tasks running on diskless compute nodes of clustered heterogeneous HPC with SLURM job scheduler. The author proposes an approach of creating user defined virtual environments with certain computing resources requirements and suggests a heterogeneous HPC cluster model. A method of using modified CoreOS operation system with integrated SLURM scheduler is described. Diskless compute nodes boot CoreOS via PXE. Initial configuration of various OS-level items and mounts performs with Ignition and Cloud-Config declarations. Cluster shared storage is based on a NFS distributed file system protocol. Users' home catalogs are shared across cluster nodes and each node has its own container storage on shared file system. Different isolated tasks share one compute node and use a combination of local Linux bridges and VXLAN

to overlay container-to-container communications over physical network infrastructure. Using of privileged Docker containers allows user to load drivers via kernel modules to support different compute nodes hardware configurations. Isolated tasks can make use of InfiniBand communications or CUDA if appropriate devices assigned to their containers. There are some security concerns of using privileged containers; otherwise, container virtualization does not increase additional security risks. In conclusion, the author also provides performance measurements to illustrate how various tasks interfere with each other and further research topics.

Keywords: *container virtualization, HPC, cluster, CoreOS, Docker, SLURM, MPI, InfiniBand, CUDA*

For citation:

Nikolaev D. S., **Korneev V. V.** Using Container Virtualization in High Performance Computing with SLURM Scheduler, *Programmnaya Ingeneria*, 2017, vol. 8, no. 4, 147—160.

DOI: 10.17587/prin.8.147-160

References

1. **Liu F., Weissman J.** Elastic Job Bundling: An Adaptive Resource Request Strategy for LargeScale Parallel Applications, Department of Computer Science and Engineering University of Minnesota, Minneapolis, Technical Report 2015.

2. **Baranov A. V., Nikolaev D. S.** Ispol'zovanie kontejnernoj virtualizacii v organizacii vysokoproizvoditel'nyh vychislenij (Using container virtualisation in high performance computing), *Programmnye sistemy: teorija i prilozhenija*, 2016, no. 1, pp. 117—134 (in Russian).

3. **About** CoreOS, available at: https://coreos.com/about/ (reference date 15.12.2016).

4. **WHAT IS** DOCKER? available at: https://www.docker.com/what-docker (date of access 17.3.2016).

5. **SLURM** Workload Manager, available at:URL: https://slurm.schedmd.com/ (date of access 15.12.2016).

6. **Jacobsen D. M., Canon R. S.** *Contain This, Unleashing Docker for HPC*, available at:http://www.nersc.gov/assets/Uploads/cug2015udi.pdf (date of access 15.12.2016).

7. **Yakubov S.** *Using Docker container virtualization in DESY HPC environment*, available at: https://indico.cern.ch/event/466991/contributions/1143628/attachments/1261303/1864379/Docker_HPC.pdf (date of access 23.11.2016).

8. **Shchapov V. A., Denisov A. V., Latypov S. R**. Using Docker container virtualization to run tasks on supercomputer nodes, *Russian Supercomputing Days*, 2016, available at:http://russianscdays.org/files/pdf16/505.pdf (date of access 29.11.2016).

9. **Markwardt U.** *Running Virtual Machines in a Slurm Batch System*, available at: https://slurm.schedmd.com/SLUG15/SlurmVM.pdf (date of access 15.12.2016).

10. **A security-minded**, standards-based container engine, available at:https://coreos.com/rkt/ (date of access 15.12.2016).

11. **systemd** System and Service Manager, available at: https://wiki.freedesktop.org/www/Software/systemd/ (date of access 15.12.2016).

12. **CoreOS** developer SDK guide, available at: https://coreos.com/os/docs/latest/sdk-modifying-coreos.html (date of access 15.12.2016).

13. **Ignition 0.12.1** Documentation, available at:https://coreos.com/ignition/docs/latest/ (date of access 15.12.2016).

14. **Using** Cloud-Config, available at:https://coreos.com/os/docs/latest/cloud-config.html (date of access 15.12.2016).

15. **Project** Calico Documentation, available at: http://docs.projectcalico.org/v2.0/introduction/ (date of access 15.12.2016).

16. **NAS** Parallel Benchmarks, available at:https://www.nas.nasa.gov/publications/npb.html (date of access 15.12.2016).

17. **About** the Lustre File, available at: http://lustre.org/about/ (date of access 15.12.2016).

ИНФОРМАЦИЯ

24 мая 2017 г. в отеле "Holiday Inn Suschevsky", Москва
информационно-аналитическое агентство TelecomDaily проводит Всероссийскую конференцию

"Интернет вещей в жилищно-коммунальном хозяйстве – IoT в ЖКХ 2017"

На предстоящей конференции будут обсуждены следующие вопросы.

- Создание и перспективные направления дальнейшего развития ГИС ЖКХ
- Современные технологии и практический опыт их применения в сфере жилищно-коммунального хозяйства. Поэтапное внедрение ГИС ЖКХ
- ГИС ЖКХ как платформа для развития технологий Интернет вещей в коммунальном хозяйстве
- Драйверы и барьеры развития экосистемы IoT в коммунальном хозяйстве
- **Интернет вещей в строительстве и ЖКХ — наступившая доподлинная реальность?**
- Автоматизированные системы управления, самообучающиеся системы и искусственный интеллект в коммунальном хозяйстве
- Развитие и внедрение сетей четвертого и пятого поколения и их влияние на экосистему IoT и услуг M2M в ЖКХ
- Перспективы сетей на технологии LPWAN в России
- Использование навигационных систем ГЛОНАСС/GPS в IoT/M2M-системах ЖКХ
- Аналитика IoT: новые методы и инструменты анализа данных
- Smart City и интеллектуальный дом
- Концепция интеллектуального дома в умном городе
- Умный дом: новейшие технологии для реальной жизни
- Основные подходы к созданию концепции "Безопасный город"
- Интеллектуальные системы поддержки принятия решений с использованием IoT в чрезвычайных и кризисных ситуациях

Подробности: http://www.tmtconferences.ru/iot2017.html

УДК 004.738 DOI: 10.17587/prin.8.161-169

Р. Э. Асратян, канд. техн. наук, вед. науч. сотр., e-mail: rea@ipu.ru,
Институт проблем управления им. В. А. Трапезникова РАН, Москва

Интернет-служба защищенной мультисерверной обработки информационных запросов в распределенных системах

Рассмотрены принципы организации сетевой службы, предназначенной для реализации защищенной обработки информационных запросов в распределенных информационных системах, ориентированных на работу в сложных мультисетевых средах со многими обрабатывающими серверами. Отличительными особенностями службы являются тесная интеграция функций информационной защиты данных с функциями информационного взаимодействия и маршрутизации запросов в условиях мультисерверной обработки.

Ключевые слова: *распределенные системы, web-технологии, интернет-технологии, информационное взаимодействие, информационная безопасность*

Введение

Несмотря на признанные достоинства современных сетевых технологий и, в первую очередь, технологии web-сервисов на базе архитектуры .NET (высокая гибкость в определении сервисных функций, высокое быстродействие, эффективная поддержка в целом ряде современных систем программирования и т. п. [1, 2]), разработчики распределенных информационных систем нередко испытывают трудности с обеспечением защиты данных в сети. Эти трудности чаще всего связаны с отсутствием встроенных средств защиты и аутентификации сетевых сообщений в применяемых сетевых технологиях и более всего проявляются при создании систем, предназначенных для работы в сложных, мультисерверных и мультисетевых средах [3] в условиях высоких требований к информационной безопасности [4, 5].

В работе [6] описана новая сетевая служба PMS (*Protected Message Service*), разработанная с целью преодоления отмеченного выше недостатка. Суть подхода заключается в тесной интеграции функций сетевого информационного обмена с функциями защиты и аутентификации данных. Внешне эта интеграция проявляется в том, что отмеченные функции входят в набор методов главного класса службы — класса "защищенное сообщение", отображающего электронный документ (информационный запрос или ответ), снабженный одной или несколькими удостоверяющими электронными цифровыми подписями (ЭЦП). В отличие, например, от технологии web-сервисов, описываемая служба опирается не на модель вызова методов удаленных объектов, а на модель обмена сообщениями. В данном случае это означает, что все сервисные обрабатывающие функции (методы) имеют одинаковую, жесткую спецификацию: они получают объект класса "защищенное сообщение" в качестве параметра и возвращают объект того же класса. Эти обрабатывающие функции группируются в одну или несколько динамических библиотек, которые подключаются к серверу PMS в момент его запуска (каждую библиотеку можно рассматривать как отдаленный аналог web-сервиса в сетевой архитектуре .NET) и становятся доступными для клиентских компонентов. Web-технологии (HTTP, HTTPS, SOAP, WSDL и т. п.), также как и web-серверы вообще не используются.

Важным преимуществом сетевых служб, построенных на модели обмена сообщениями, является принципиальная возможность организации "программного конвейера": обработки информационного запроса не одной сервисной функцией, но цепочкой функций таким образом, что защищенное сообщение, возвращенное каждой сервисной функцией, или передается непосредственно на вход следующей функции в цепочке (если она имеется), или возвращается клиенту. Разумеется, это прямая аналогия с известным еще с первых версий операционной системы UNIX механизмом трубопровода (*pipeline*), основанном на последовательном соединении стандартных выводов и вводов у нескольких процессов в компьютере. Однако поскольку в данном случае речь идет о сетевой службе, сетевых сообщениях и о распределенных системах, наибольший интерес представляет мультисерверная организация конвейера, в которой сервисные функции, задействованные в обработке информационного запроса, могут выполняться на разных серверах, в том числе на серверах, размещенных в разных сетях.

В данной статье рассмотрен подход к расширению возможностей PMS за счет организации мультисерверной обработки информационных запросов (PMS-конвейер). При этом основной акцент сделан на средствах межсерверного взаимодействия и маршрутизации защищенных сообщений в сложных мультисетевых средах.

Краткие сведения о PMS

Как и всякая сетевая служба, основанная на базовом сетевом протоколе TCP/IP [7], PMS поддержана клиентским и серверным программным обеспечением. Сервер PMS представляет собой постоянно активную программу, обслуживающую запросы на обработку от клиентов, по умолчанию используется порт 8132. Клиентское программное обеспечение представляет собой библиотеку функций PmsBase.dll, реализующих прикладной программный интерфейс (API) к PMS. Этот интерфейс является "лицом" PMS с точки зрения пользователя.

На рис. 1 представлен фрагмент кода в нотации C#, иллюстрирующий простое обращение к обрабатывающей функции с применением средств защиты данных. Две первые строки кода определяют две переменные типа PmsMessage, представляющего собой главный класс PMS (защищенное сообщение). Первой переменной (Request) присваивается значение — объект класса PmsMessage, инициированный символьной строкой (например, содержащей XML-документ информационного запроса). Вторая переменная (Reply) предназначена для хранения результата обработки. В третьей строке определяется и инициируется переменная MyConn класса PmsConnection, предназначенного для создания и прекращения сетевого соединения с сервером.

В четвертой и пятой строках кода выполняется формирование подписей в запросе. Сначала определяется переменная SenderCerts класса PmsCertList. Этот класс предназначен для хранения в памяти списков сертификатов с открытыми ключами в стандарте X509 и содержит несколько конструкторов для загрузки сертификатов из файлов или из системных хранилищ с поиском по имени владельца или по серийному номеру. В переменную SenderCerts загружаются два сертификата, соответствующих именам владельцев "Иванов" и "Петров" для последующего формирования двух ЭЦП в запросе. Такой способ использования означает, что с этими сертификатами обязательно должны быть связаны парные им закрытые ключи, иначе формирование ЭЦП закончится неудачно. Вызов метода AddSignatures позволяет сформировать две ЭЦП в сообщении Request.

```
PmsMessage Request =  new PmsMessage("<request> … </request>");
PmsMessage Reply;
PmsConnection MyConn = new PmsConnection();
PmsCertList  SenderCerts= PmsCertList( new string [] {"Иванов", "Петров"});
Request.AddSignatures(SenderCerts);

MyConn.Connect("MyServer");
PmsCertList  ReceiverCert = MyConn.GetServerCertificate();

Reply=Request.Process (MyConn,"MyLib.MyFunc  param", ReceiverCert);
if(Reply != null)
{
     string Signer;
     for(int i=0; Reply.Signatures.Length; i++)
        if(Reply.VerifySignature(i, out Signer) >= 0)
              Console.WriteLine ("Ответ подписал: " + Signer);
     Console.WriteLine (Reply.GetString());
}
else
     Console.WriteLine ("Ошибка: " +MyConn.ErrMsg);
MyConn.Disconnect();
```

Рис. 1. Пример использования PMS

Собственно вызов обрабатывающей функции начинается с вызова метода Connect, устанавливающего сетевое соединение с сервером с указанным сетевым именем или адресом. Далее выполняется первая сетевая операция — запрос сертификата сервера (метод GetServerCertificate) с занесением результата в переменную ReceiverCert уже знакомого класса PmsCertList для последующего шифрования информационного запроса. Сразу же после успешного получения сертификата сервера выполняется вызов удаленной функции с помощью применения метода Process к переменной Request с использованием все того же соединения и полученного сертификата. Предполагается, что к серверу PMS подключена библиотека MyLib.dll, содержащая код функции MyFunc. При запуске функции на сервере ей передаются значение переменной Request и опциональный строковый параметр param в качестве фактических параметров. Результат обработки заносится в переменную Reply. Подчеркнем, что шифрование запроса и дешифрование ответа выполняются автоматически в методе Process.

Последующие строки обеспечивают последовательную проверку всех ЭЦП в полученном ответе сервера (вызов метода VeifySignature) и запись в стандартный вывод сведений о подписантах.

Пример заканчивается записью результата обращения в стандартный вывод (предполагается, что результат, как и запрос, имеет форму символьной строки) и закрытием соединения с сервером с помощью метода Disconnect, так как в данном примере оно больше не нужно.

Необходимо отметить, что из обоих фрагментов кода намеренно удалены операторы обработки исключений.

Организация сервера PMS проиллюстрирована на рис. 2. Все множество сервисных обрабатывающих функций реализуется в форме одной или нескольких динамических библиотек. Никакие специальные конструкции типа web-сервисов не используются. Поиск и загрузка всех динамических библиотек выполняются при запуске сервера PMS из каталога, указанного в конфигурации сервера.

В каждой найденной библиотеке проводятся поиск и подключение всех функций, имеющих строго определенную спецификацию:

PmsMessage *имя_функции*
(PmsMessage Inpt, string [] Conf, ref string Msg),

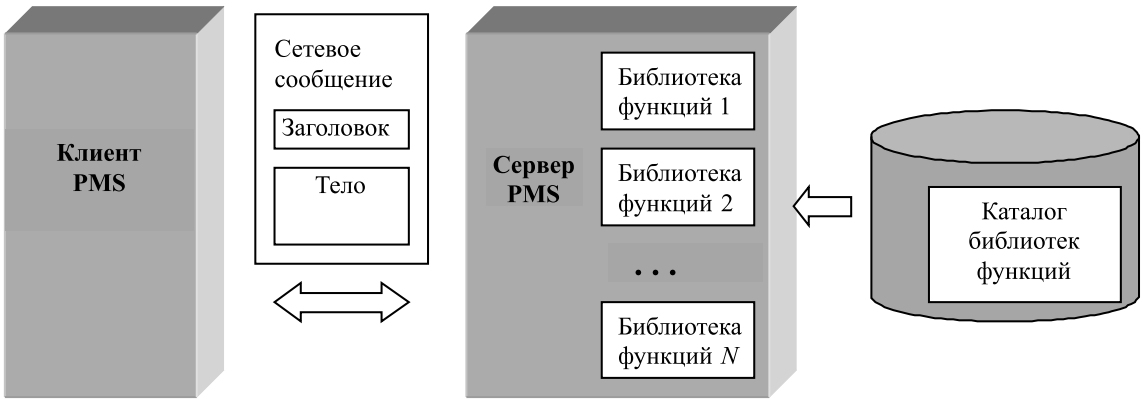


Рис. 2. Принципы организации сервера PMS

в которой параметр Inpt — входное сообщение (запрос); параметр Conf — конфигурационные данные в форме массива строк вида "имя=значение"; строковый параметр Msg на входе принимает любое значение, а возвращает диагностическое сообщение функции. С этого момента все библиотечные функции, соответствующие данной спецификации, начинают играть роль сервисных функций, доступных для клиентских программ. Все остальные функции попросту игнорируются.

Служба PMS в полной мере использует двоичную природу TCP/IP [7, 8]. Взаимодействие между клиентом и сервером PMS осуществляется по специальному, достаточно простому PMS-протоколу, ориентированному на передачу двоичных сетевых сообщений (PMS-сообщений) в обоих направлениях. Каждое такое сообщение в общем случае содержит два массива байтов: заголовок сообщения и тело сообщения (см. рис. 2). Первые четыре байта заголовка или тела сообщения содержат целое число — его длину. При передаче запроса от клиента к серверу в заголовок сетевого сообщения помещается строка, содержащая полное имя вызываемой функции, а в тело сообщения упаковывается структура PmsMessage в открытой или зашифрованной форме, содержащая информационный запрос. Строка заголовка используется сервером для организации вызова соответствующей обрабатывающей функции. При передаче результата обработки от сервера к клиенту в заголовок сетевого сообщения помещается строка диагностического сообщения (значение параметра Msg, сформированное обрабатывающей функцией), а в тело сообщения упаковывается структура PmsMessage, содержащая ответ сервера в открытой или зашифрованной форме. Полученное от сервера диагностическое сообщение автоматически присваивается члену ErrMsg объекта класса PmsConnection на стороне клиента (см. рис. 1).

Принципы организации мультисерверного PMS-конвейера

Как уже отмечалось, главное свойство сервисных функций заключается в том, что они получают объект класса PmsMessage (защищенное сообщение) в качестве основного параметра и возвращают объект того же класса в качестве результата. Это создает принципиальную возможность организации конвейерной обработки сообщений на сервере PMS, т. е. последовательной обработки входного сообщения цепочкой сервисных функций.

Предположим, например, что в библиотеке Flib имеются три сервисные функции:

- функция List получает во входном сообщении строку, содержащую спецификацию папки на сервере и возвращает построчный список имен файлов в папке;
- функция Filter получает построчный список имен файлов и оставляет в нем только файлы с указанным расширением имени;
- функция Sort сортирует список файлов.

Рассмотрим следующее обращение к серверу PMS, в котором параметр метода Process содержит несколько имен сервисных функций, разделенных запятыми (конвейерный список):

```
PmsMessage Request=PmsMessage("c:\\Datafiles");
Request.AddSignatures(SenderCerts);
MyConn.Connect("alpha");
...
PmsMessage Reply=Request.Process(MyConn,
"Flib.List, Flib.Filter jpg, Flib.Sort", ReceiverCert);
```

Здесь переменные MyConn, SenderCerts и ReceiverCert имеют тот же смысл, что и на рис. 1. Результатом данного обращения является отсортированный список имен файлов с расширением .jpg, содержащихся в папке c:\Datafiles на сервере PMS. Предполагается, что функция Filter получает требуемое значение расширения .jpg через строковый формальный параметр Msg. Процесс обработки для данного примера проиллюстрирован на рис. 3. Легко заметить, что функции криптозащиты (формирования ЭЦП, шифрования, дешифрования и проверки ЭЦП) в данном случае должны применяться только при передаче запроса серверу и возврате результата клиенту. Если бы клиентская программа вызывала сервисные функции поочередно, то все это пришлось бы повторять при каждом вызове. Именно в этом заключается положи-

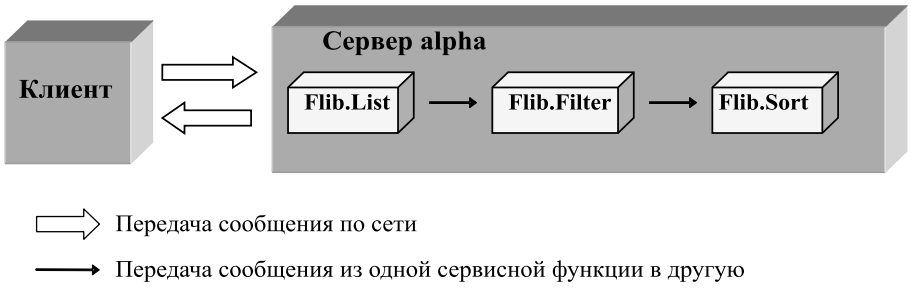


Рис. 3. Пример конвейерной обработки

тельный эффект применения PMS-конвейера с точки зрения защиты данных.

Применение PMS-конвейера для распределенной, мультисерверной обработки реализуется по следующим правилам.

- Кроме имен сервисных функций в конвейерном списке может использоваться еще один элемент: вызов удаленного сервера. Этот элемент имеет следующий формат:

"имя_сервера/(имя$_1$, имя$_2$, ..., имя$_N$)",

где каждое "имя" может, в свою очередь, представлять собой имя функции или же вложенный вызов удаленного сервера. Если $N = 1$, то скобки можно отбросить:

"имя_сервера/имя$_1$".

- Как и всякая многоканальная служба, сервер PMS создает отдельный обрабатывающий программный поток (нить) для обслуживания каждого запроса на сетевое соединение от клиента. Этот поток оснащается собственной структурой данных, обеспечивающей выполнение не только серверных, но и клиентских функций с использованием классов PmsConnection и PmsMessage. Обработка каждого вложенного вызова удаленного сервера, содержащегося в конвейерном списке, выливается в обращение к другому серверу с передачей ему на обработку соответствующего фрагмента конвейерного списка.

На рис. 4 проиллюстрирована обработка достаточно сложного обращения к серверу PMS с именем alpha, в которой задействованы еще три дополнительных сервера (beta, delta и gamma) в соответствии со следующим вызовом метода Process:

```
MyConn.Connect("alpha");
...
PmsMessage Reply=Request.Process(MyConn, "L1.A,
L1.B, beta/L2.E, L1.C, delta/(L3.F, gamma/L4.H,
L3.G), L1.D", ReceiverCert);
```

Здесь переменные MyConn, Request, Reply и ReceiverCert имеют тот же смысл, что и на рис. 1, L1, L2, L3 и L4 — это имена использованных библиотек, а латинские буквы от A до H — имена серверных функций.

Выполнение клиентского метода Process в сервере PMS включает прием сетевого сообщения по TCP-соединению с клиентом и его обработку по приведенному ниже алгоритму. Предполагается, что переменная InMsg типа PmsMessage используется для хранения входного сообщения при вызове сервисных библиотечных функций; переменная CertList типа

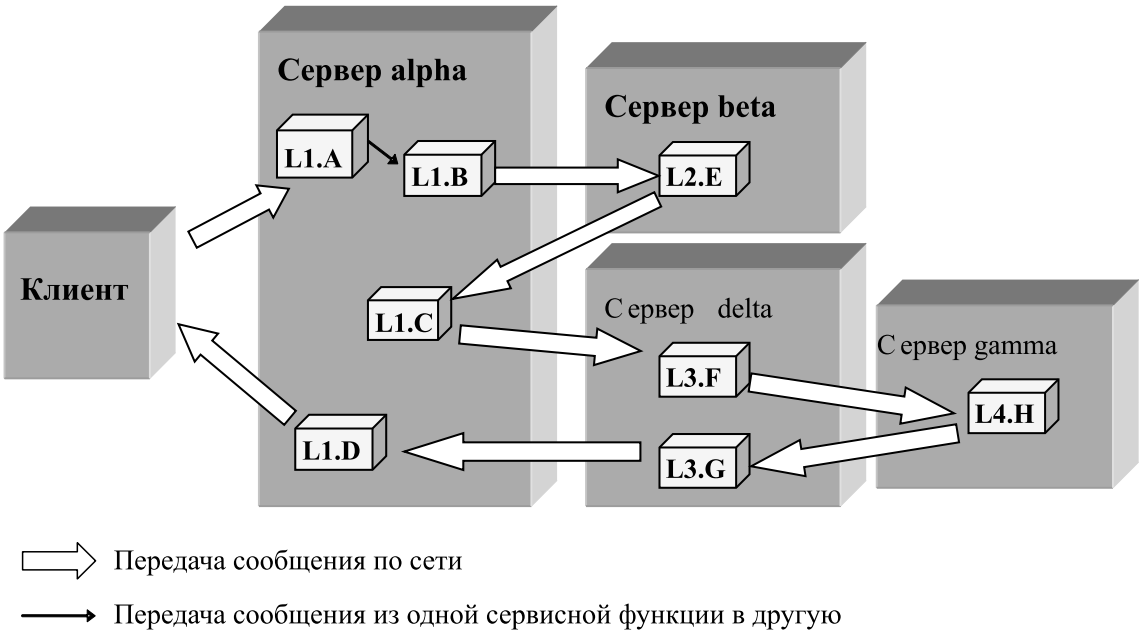


Рис. 4. Пример мультисерверной конвейерной обработки

PmsCertList предназначена для хранения списков сертификатов с открытыми ключами в стандарте X509, извлеченных из всех ЭЦП первичного входного сообщения; а строчная переменная DiagStr используется для формирования итогового диагностического сообщения клиенту (инициируется пустой строкой).

Шаг 1. Извлечение конвейерного списка из заголовка сетевого сообщения. Извлечение зашифрованного объекта типа PMSMessage в форме массива байтов из тела сетевого сообщения. Дешифрование объекта и занесение его в переменную InMsg. Проверка корректности найденных в нем ЭЦП. Если хотя бы одна проверка закончилась неудачно, то формирование соответствующего диагностического сообщения в переменной DiagStr и переход к шагу 7. В противном случае — извлечение всех сертификатов из ЭЦП и занесение их в переменную CertList.

Шаг 2. Если конвейерный список пуст, переход к шагу 7. В противном случае начало последовательного разбора полученного конвейерного списка, начиная с первого элемента.

Шаг 3. Извлечение очередного элемента из конвейерного списка. Если этот элемент представляет собой вызов удаленного сервера, то переход к шагу 5, в противном случае — к шагу 4.

Шаг 4. Организация вызова серверной функции с именем, содержащимся в обрабатываемом элементе конвейерного списка. В качестве фактических аргументов задаются защищенное сообщение, содержащееся в переменной InMsg, и строковый параметр, если он имеется в обрабатываемом элементе. После завершения работы функции возвращенное защищенное сообщение заносится в переменную InMsg, а сформированное диагностическое сообщение добавляется к значению переменной DiagMsg отдельной строкой. Если InMsg имеет пустое значение (null), то — переход к шагу 7, в противном случае — к шагу 6.

Шаг 5. Организация обращения к удаленному серверу, включающая формирование собственной ЭЦП сервера для защищенного сообщения в переменной InMsg (с помощью метода AddSignatures), соединение с удаленным сервером (с помощью метода Connect класса PmsConnection) и получение сертификата удаленного сервера с помощью метода GetServerCertificate. В случае неудачного завершения любой операции — добавление соответствующего диагностического сообщения к значению переменной DiagMsg отдельной строкой и переход к шагу 7. В противном случае — извлечение новой конвейерной строки для удаленного сервера из обрабатываемого элемента и отправка защищенного сообщения из переменной InMsg на обработку с помощью метода Process с использованием установленного соединения и новой конвейерной строки. После завершения обработки возвращенное защищенное сообщение заносится в переменную InMsg, а сформированное диагностическое сообщение извлекается из использованного объекта класса PmsConnection и добавляется к значению переменной DiagMsg отдельной строкой. Если InMsg имеет пустое значение (null), то — переход к шагу 7. Если проверка ЭЦП в защищенном сообщении закончилась неудачно, добавление соответствующего диагностического сообщения к переменной DiagStr и переход к шагу 7, в противном случае — к шагу 6.

Шаг 6. Если в конвейерном списке еще имеются необработанные элементы, то переход к шагу 3. В противном случае выполняется формирование собственной ЭЦП сервера в итоговом защищенном сообщении, содержащемся в переменной InMsg, и шифрование его всеми открытыми ключами, содержащимися в сертификатах из списка в переменной CertList (т. е. ответное сообщение шифруется открытыми ключами подписантов входного сообщения). Формирование сетевого сообщения с заголовком, содержащим строку из переменной DiagStr, и телом, содержащим зашифрованное ответное сообщение, и отправка его клиенту через TCP-соединение. Завершение обработки.

Шаг 7. Аварийное завершение. Формирование сетевого сообщения с заголовком, содержащим строку из переменной DiagStr, и пустым телом, и отправка его клиенту через TCP-соединение. Завершение обработки.

Важно подчеркнуть, что проверка ЭЦП на сервере PMS может включать не только аутентификацию входящего защищенного сообщения, но и выборочную проверку реквизитов подписантов (имя, организация, должность и т. п.) по специальным спискам доступа, содержащимся в конфигурационных данных сервера.

Формирование собственной ЭЦП сервера PMS осуществляется с помощью сертификата, указанного в его конфигурационных данных: серийный номер сертификата в одном из системных хранилищ или имя файла, содержащего сертификат (разумеется, с этим сертификатом должен быть связан закрытый ключ, иначе формирование ЭЦП окажется невозможным). Функции проверки ЭЦП у входящих сообщений, а также подписания и шифрования исходящих являются опциональными и могут выборочно отключаться и включаться через конфигурационные данные сервера.

Как видно из приведенного алгоритма, если клиент установил соединение с сервером alpha, а в качестве параметра метода Process задал строку "beta/MyLib.MyFunc", то обработка запроса будет фактически выполнена на сервере beta, а серверу alpha достанется лишь роль посредника (прокси-сервера). Тем не менее эта роль может заключаться не только в простом перенаправлении клиентского запроса в сервер beta, но и в выполнении функций информационной защиты. Предположим, например, что сервер beta размещен в частной локальной сети предприятия и недоступен клиентам из глобальной сети для прямого соединения, а сервер alpha занимает положение "пограничного" сервера, подключенного к обеим сетям — и к частной, и к глобальной. В этой ситуации сосредоточение функций информационной защиты в сервере alpha могло бы защитить всю частную сеть предприятия (включая сервер beta и,

возможно, другие серверы) от проникновения в нее несанкционированных запросов из глобальной сети. Особенности применения PMS в сложных сетевых структурах рассмотрены в следующем разделе.

Маршрутизация защищенных сообщений в мультисетевой среде

Важную часть сети Интернет составляют частные локальные сети организаций, соединенные глобальной сетью. Статус частной сети предполагает высокую степень изолированности от остальной среды: прямое сетевое соединение между программами, одна из которых работает в частной сети, а другая — вне нее, невозможно без применения специальных средств, например, трансляторов IP-адресов в датаграммах или серверов-посредников. Желание администраторов "спрятать" информационные ресурсы в частных сетях и возрастающее недоверие к низкоуровневым (т. е. основанным на IP-адресах и портах) средствам ограничения доступа — межсетевым экранам — можно признать важной тенденцией последних лет.

Если компоненты распределенной системы разнесены по разным частным локальным сетям предприятий, то возникает проблемный вопрос организации безопасного взаимодействия между ними через глобальную сеть. Обычный способ его решения заключается в применении технологии VPN (*Virtual Private Network*) [8] — технологии построения защищенных каналов взаимодействия через общедоступную глобальную сеть, позволяющей связать две или более удаленных локальных сетей в одну территориально-распределенную частную сеть с единым жестким администрированием, гарантирующим уникальность IP-адресов в пределах всей распределенной сети. Технология основана на передаче IP-датаграмм "поверх" другого протокола, оснащенного средствами информационной защиты (например, PPP или IPSec). Однако при наличии большого числа частных сетей, принадлежащих различным организациям, требование жесткого централизованного администрирования становится практически нереализуемым.

Как видно из предыдущего раздела, в функциональность сервера PMS заложено "поведение" прокси-сервера, обеспечивающего защиту данных. Для решения вопроса организации безопасного взаимодействия между удаленными частными сетями эта функциональность должна быть дополнена еще одним важным механизмом — средствами маршрутизации информационных запросов в мультисетевой среде. Эта маршрутизация является составной частью обработки элемента "вызов удаленного сервера" в конвейерном списке и осуществляется по следующим правилам.

- При обработке вызова удаленного сервера ("шаг 5" в приведенном выше алгоритме) сервер PMS сравнивает имя адресуемого сервера с собственным именем, содержащимся в его конфигурационных данных. Если эти имена совпадают, сервер PMS самостоятельно обрабатывает соответствующую этому вызову вложенную конвейерную строку. В противном случае, весь элемент "вызов удаленного сервера" (вместе с именем сервера и вложенным конвейерным списком) передается на обработку другому серверу.
- Перед соединением с удаленным сервером сервер PMS выполняет собственно процедуру маршрутизации с использованием таблицы маршрутизации, которая является опциональной частью его конфигурационных данных. Используя имя удаленного сервера в качестве ключа, сервер PMS осуществляет поиск строки таблицы маршрутизации, соответствующей этому ключу, и извлекает из найденной строки интернет-имя или IP-адрес удаленного сервера для выполнения сетевого соединения. Если же поиск строки заканчивается неудачно, то исходное имя удаленного сервера интерпретируется как его интернет-имя.
- После соединения с удаленным сервером и получения от него сертификата для шифрования защищенного сообщения, сервер PMS выполняет не только обычную проверку корректности полученного сертификата, но и выборочную проверку реквизитов его владельца (имя, организация, должность и т. п.) для защиты от фальсифицированного удаленного сервера. Поэтому, кроме интернет-имени или IP-адреса удаленного сервера, строка таблицы маршрутизации может содержать и список контрольных значений реквизитов владельца для выполнения такой проверки. Если этот список отсутствует в найденной строке таблицы, то запрос сертификата у удаленного сервера и шифрование сообщения попросту не выполняются.
- Даже если используются всего два сервера PMS, при неудачной подготовке таблиц маршрутизации возможна ситуация "вечного зацикливания" сообщения между ними. Для его предотвращения в заголовок сетевого сообщения добавляется специальное поле — однобайтовый обратный счетчик межсерверных передач. Это поле инициируется максимальным значением (конфигурируемый параметр) и декрементируется всякий раз, когда один сервер направляет защищенное сообщение другому без какой-либо обработки (аналог поля TTL в заголовке IP-пакета [7]). При обнулении счетчика обработка сообщения прекращается, а клиенту возвращается соответствующее диагностическое сообщение.

На рис. 5 проиллюстрирована маршрутизация информационного запроса в мультисетевой среде. Как видно на рис. 5, рассматриваемая среда включает три частных локальных сети (LAN_1, LAN_2 и LAN_3), соединенных с глобальной сетью посредством "пограничных" серверов PMS (обозначены темными параллелепипедами). Предполагается, что каждый из "пограничных" серверов PMS оснащен двумя сетевыми интерфейсами, один из которых (внутренний) подключен к локальной сети, а другой (внешний) — к глобальной. Предположим также, что собственные имена этих серверов в их конфигурационных данных — Pms1.ru, Pms2.ru и Pms3.ru, соответственно, и таковы же интернет-имена их внешних интерфейсов

в глобальной сети. В данном примере на эти серверы возлагаются исключительно служебные функции, связанные с маршрутизацией и защитой информационных запросов. Рядом с каждым из этих серверов в прямоугольной рамке показан фрагмент его таблицы маршрутизации, имеющий отношение к примеру. Кроме того, в LAN_2 и LAN_3 имеются обрабатывающие серверы PMS с именами Filial2.1 и Filial3.1 и одинаковыми IP-адресами (например, 192.168.0.3), в которые загружены библиотеки сервисных функций Lib2 и Lib3 соответственно.

Рассмотрим обработку следующего клиентского обращения к серверу Pms1.ru:

MyConn.Connect("*внутренний_адрес_Pms1*");
...
PmsMessage Reply=Request.Process(MyConn, "Filial2.1/Lib2.MyFunc", null);

Здесь переменные MyConn, Request и Reply имеют тот же смысл, что и на рис. 1, *внутренний_адрес_Pms1* — сетевой адрес внутреннего сетевого интерфейса сервера Pms1.ru, а MyFunc — имя сервисной функции.

Используя свою таблицу маршрутизации, сервер Pms1.ru определит, что любой запрос к серверу, имя которого начинается на Filial2, должен быть перенаправлен на сервер Pms2.ru. Предварительно сервер проверит ЭЦП, содержащиеся во входящем сообщении, опционально добавит к нему свою собственную подпись, установит соединение с сервером Pms2.ru, запросит сертификат у этого сервера для шифрования сообщения, проверит этот сертификат на корректность и проведет выборочную проверку реквизитов его владельца. В данном случае на эти реквизиты накладывается единственное ограничение: название организации владельца должно начинаться со строки "Филиал2", за которой может следовать все что угодно (отдельные реквизиты обозначаются одной или двумя латинскими буквами точно так же, как у сертификатов в стандарте X509). Если проверка ЭЦП, контроль корректности сертификата или реквизитов владельца закончатся неуспешно, то обработка запроса будет прекращена, а клиенту будет направлен ответ с соответствующим диагностическим сообщением.

Обработка запроса в сервере Pms2.ru будет выполнена аналогично, с перенаправлением запроса на сервер PMS с IP-адресом 192.168.0.3, но с одним отличием: так как в строке таблицы маршрутизации отсутствуют контрольные значения реквизитов владельца сертификата, запрос сертификата у этого сервера и шифрование защищенного сообщения в данном случае выполняться не будут (в данном примере предполагается, что частная локальная сеть является безопасной зоной).

Собственно обработка сообщения будет выполнена на сервере Filial2.1 с помощью сервисной функции Lib2.MyFunc. Ответное сообщение будет доставлено клиенту через ту же цепочку серверов с соблюдением аналогичных мер защиты.

На рис. 5 передача информационного запроса между узлами сети обозначена белыми стрелками. Легко видеть, что если бы в клиентском обращении был задан сервер Filial3.1, а не Filial2.1, то обработка

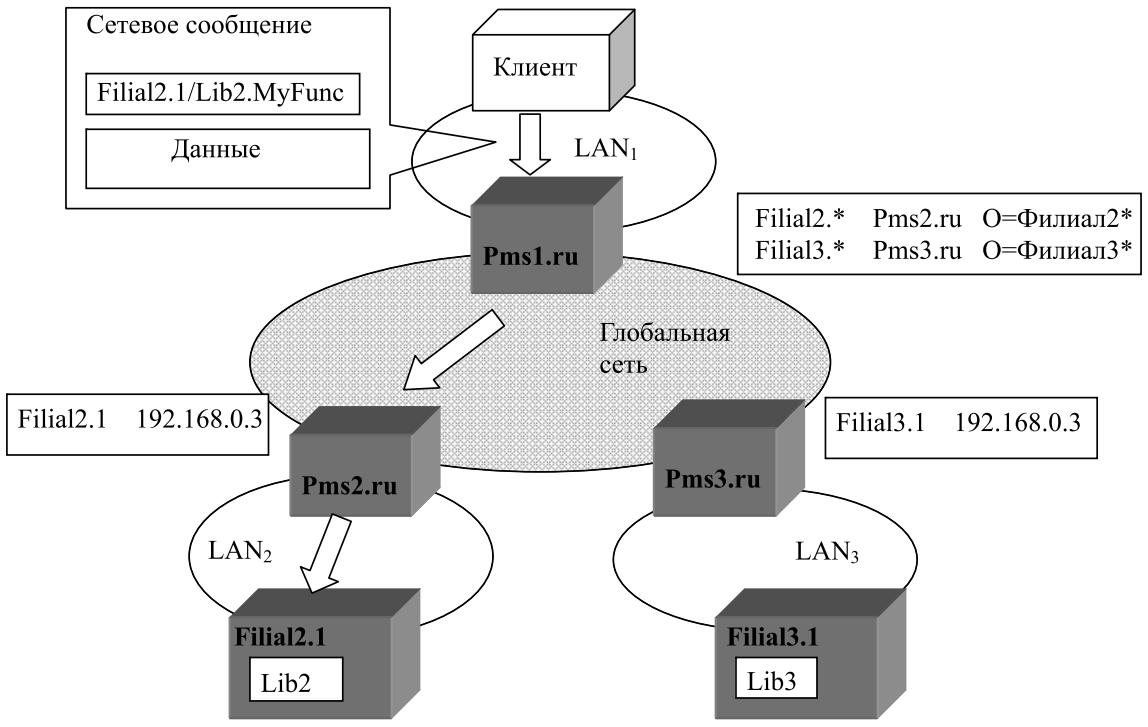


Рис. 5. Маршрутизация защищенного сообщения в мультисетевой среде

была бы выполнена совершенно аналогично, а совпадение IP-адресов у этих серверов не создало бы никаких трудностей.

Как видно из приведенного примера, серверы PMS могут быть использованы как для функций обработки (Filial2.1, Filial3.1), так и для выполнения служебных функций, связанных с информационной безопасностью и маршрутизацией данных. В данном случае "пограничные" серверы (Pms1.ru, Pms2.ru и Pms3.ru) обеспечивают защиту и маршрутизацию сообщений в глобальной сети, а также защиту частных сетей предприятий от проникновения несанкционированных информационных запросов.

Заключение

Организация PMS заключается в отказе от гибкой модели организации взаимодействий на основе вызова методов удаленных объектов, характерной для электронных сервисов в архитектуре .NET, в пользу более жесткой модели обмена сетевыми сообщениями при фиксированной спецификации сервисных функций. В данной работе автор попытался показать, что этот подход позволяет предложить ряд готовых решений таких важных для разработчиков задач, как защита и маршрутизация информационных запросов в сложных мультисетевых средах и организация конвейерной обработки запросов в мультисерверной среде.

Разумеется, мультисерверная обработка запросов может быть организована и на основе технологии web-сервисов. Например, в процессе обработки запроса сервис *A* может вызвать сервис *B*, а тот, в свою очередь, сервис *C* и т. п. Однако эти вызовы должны быть явно заложены в программы сервисов. Идеология PMS-конвейера основана на том, что задействованные сервисные функции не вызывают друг друга, но могут быть связаны между собой в разных сочетаниях, описываемых конвейерными списками PMS.

Описанный подход был реализован в форме двух дополняющих друг друга программных продуктов: сервера PMS (в форме постоянно активной Windows-службы) и библиотеки функций PMSBase.dll для клиентских приложений и библиотечных сервисных функций. Оба продукта реализованы на языке C# в среде Microsoft Visual Studio для среды MS Framework 4.0 с использованием сертифицированной криптосистемы "КриптоПро CSP" версии 3.6. Проведенные эксперименты с новой сетевой службой показали ее достаточно высокое быстродействие [6] и удобство применения для защиты данных в распределенных системах.

Список литературы

1. **Шапошников И. В.** Web-сервисы Microsoft .NET. СПб.: БХВ-Петербург, 2002. 336 с.
2. **Мак-Дональд М., Шпушта М.** Microsoft ASP .NET 3.5 с примерами на C# 2008 и Silverlight 2 для профессионалов. М.: Вильямс, 2009. 1408 с.
3. **Асратян Р. Э., Лебедев В. Н.** Защита электронных сообщений в мультисетевой среде // Управление большими системами. 2013. № 45. С. 95—111.
4. **Згоба А. И., Маркелов Д. В., Смирнов П. И.** Кибербезопасность: угрозы, вызовы, решения // Вопросы кибербезопасности. 2014. № 5. С. 30—38.
5. **Щеглов А. В.** Защита компьютерной информации от несанкционированного доступа. СПб.: Наука и техника, 2004. 384 с.
6. **Асратян Р. Э.** Интернет-служба защищенной обработки информационных запросов в распределенных системах // Программная инженерия. 2016. № 11. С. 490—497.
7. **Снейдер Й.** Эффективное программирование TCP / IP. Библиотека программиста. СПб.: Символ-Плюс, 2002. 320 с.
8. **Хант К.** TCP/IP. Сетевое администрирование. СПб.: Питер, 2007. 816 с.

Internet Service for Protected Multi-server Processing of Information Queries in Distributed Systems

R. E. Asratian, e-mail: rea@ipu.ru, V. A. Trapeznikov Institute of Control Sciences of Russian Academy of Sciences, Moscow, 117997, Russian Federation

Corresponding author:

Asratian Ruben E., Leading Researcher, V. A. Trapeznikov Institute of Control Sciences of Russian Academy of Sciences, 117997, Moscow, Russian Federation
E-mail: rea@ipu.ru

Received on January 18, 2017
Accepted on January 24, 2017

The principles of the organization of the new network service — Protected Message Service (PMS) — intended for protected queries processing in the distributed information systems oriented on operation in complex multi-server and multi-network structures are considered. Distinctive features of service are tight integration of functions of information data protection with functions of information exchange and information requests routing for multi-server processing.

From the client point of view the service architecture is based on two main program classes: "Protected message" and "Network Connection". This classes offer necessary functionality not only for creating and protecting messages, but also for transferring them to remote server via established network connections for processing. Contrary to web-services, based on remote function call model, PMS-service is based on message processing model: all service functions receive object of "Protected message" class as a parameter and return another object of the same class as a result of processing. This approach opens a basic possibility for the organization of pipeline processing of one protected message by the sequence of service functions in a manner when the output of each function is forwarded to the input of the following one, and the result of last function returns to the client. The principles of the organization of pipeline processing of queries in a chain of servers are considered. The emphasis on routing of queries in the complex multi-network structures including a set of private local area networks is especially placed.

Keywords: *distributed systems, web-technologies, Internet-technologies, network interactions, data security*

For citation:

Asratian R. E. Internet Service for Protected Multi-server Processing of Information Queries in Distributed Systems, *Programmnaya Ingeneria*, 2017, vol. 8, no. 4, pp. 161—169.

DOI: 10.17587/prin.8.161-169

References

1. **Shaposhnikov I. V.** *Web-servisy Microsoft .NET* (Web-services Microsoft .NET), Saint-Petersburg, BHV-Peterburg, 2002, 336 p. (in Russian).

2. **Mak-Donal'd M., Szpuszta M.** *Microsoft ASP .NET 3.5 s primerami na C# 2008 i Silverlight 2 dlja professionalov* (Pro ASP .NET 3.5 in C#2008 Includes Silverlight 2), Moscow, Williams, 2009, 1408 p. (in Russian).

3. **Asratian R. E., Lebedev V. N.** Zashhita jelektronnyh soobshhenij v mul'ti-setevoj srede (Message protection in multi-network environment), *Upravlenie bol'shimi sistemami*, 2013, no. 45, pp. 95—111 (in Russian).

4. **Zgoba A. I., Markelov D. V., Smirnov P. I.** Kiberbezopasnost': ugrozy, vyzovy, reshenija (Cyber security: threats, challenges, decisions), *Voprosy kiberbezopasnosti*, 2014, no. 5, pp. 30—38 (in Russian).

5. **Shheglov A. V.** *Zashhita komp'juternoj informacii ot nesankcionirovannogo dostupa* (Protection of computer information against illegal access), Saint-Petersburg, Nauka i tehnika, 2004, 384 p. (in Russian).

6. **Asratian R. E.**, Internet-sluzhba zashhishhennoj obrabotki informacionnyh zaprosov v raspredelennyh sistemah (Internet service for protected information queries processing in distributed systems), *Programmnaya Ingeneria*, 2016, vol. 7, no. 11, pp. 490—497 (in Russian).

7. **Snader J.** *Effektivnoe programmirovanie TCP/IP* (Effective TCP/IP programming), Biblioteka programmista, Saint-Petersburg, Simvol-Pljus, 2002, 320 p. (in Russian).

8. **Hant K.** *TCP/IP. Setevoe administrirovanie* (TCP/IP. Network administration), Saint-Petersburg, Piter, 2007, 816 p. (in Russian).

УДК 004.855.5 DOI: 10.17587/prin.8.170-176

В. А. Харахинов, аспирант, e-mail: tes4obse@mail.ru,
С. С. Сосинская, канд. техн. наук, доц., проф., e-mail: sosinskaya@mail.ru,
Иркутский национальный исследовательский технический университет

Исследование способов кластеризации деталей машиностроения на основе нейронных сетей

Изложены подходы к кластеризации деталей машиностроения, проведенной на основе известных методов с использованием самоорганизующихся сетей. Обсуждены различные способы визуализации результатов кластеризации. Один из них основан на построении графика Эндрюса, который для рассматриваемой выборки деталей дает только приблизительное графическое представление о совокупности экземпляров деталей. Еще один способ визуализации использует факторный анализ для понижения размерности многомерного описания деталей в целях их отображения в пространстве координат факторов. Рассмотрена возможность проведения кластерного анализа на основе описания деталей в координатах факторов в целях сокращения затрат времени. Проведено сравнение результатов кластерного анализа деталей машиностроения на основе самоорганизующихся сетей с результатами, полученными при использовании других методов кластеризации.

Ключевые слова: *кластеризация, детали машиностроения, сети Кохонена, график Эндрюса, факторный анализ*

Введение

Совершенствование технологической подготовки машиностроительного производства за счет его автоматизации является в настоящее время одним из важных направлений управления предприятием этой отрасли. Повсеместное применение ЭВМ для решения технологических задач обусловлено высокими темпами роста промышленного производства и ужесточением требований к качеству выпускаемой продукции. Современные машины становятся все более сложными, точными, надежными в работе и долговечными, растут объем и сложность работ на всех этапах технической подготовки их производства.

Для сокращения объемов технической, главным образом, технологической подготовки важную роль играет групповой метод — метод унификации. При его применении для групп однородной по тем или иным конструктивно-технологическим признакам продукции устанавливают однотипные высокопроизводительные методы обработки с использованием однородных и быстро переналаживаемых станков. Чем выше уровень унификации технологии на базе группового метода, тем проще и рациональнее организационные формы производства. В основе унификации технологии лежит классификация продукции.

В статье рассмотрены нейронные сети и карты Кохонена [1] как инструментальные средства для проведения кластерного анализа деталей машиностроения, которые описываются определенным набором признаков. Дано сравнение полученных результатов с другими методами кластеризации. Обсуждены возможности наглядного отображения результатов и проведения кластеризации по факторным значениям.

Комплексная деталь

Несмотря на многообразие и различие конструкций, детали машин и приборов имеют большое число конструктивных, размерных, точностных, технологических и подобных перечисленным родственных признаков. При построении групповых процессов механической обработки за основу берут комплексную деталь.

Под комплексной деталью понимают реальную или условную (искусственно созданную) деталь, содержащую в своей конструкции все основные элементы (поверхности), характерные для деталей данной группы, и являющуюся ее конструктивно-технологическим представителем. Естественно предположить, что технологический процесс обработки, составленный на комплексную деталь, с небольшой дополнительной переналадкой оборудования должен быть применим при изготовлении любой другой детали данной группы [2]. В качестве данных для проведения экспериментов, подтверждающих это предположение, были взяты 34 втулки, обрабатываемые на револьверных станках [2]. Такая выборка является примером комплексной детали. Получить выборку большей размерности не представляется возможным

в силу ограниченного доступа к промышленным базам данных. Признаки, описывающие каждую деталь: максимальный диаметр; высота; число внешних диаметров; число внутренних диаметров; число конических поверхностей; число резьб.

Классификация и кластеризация

Для автоматизированного разбиения объектов на классы существует большое число методов [3]. Наиболее полная и точная классификация таких методов представлена в работе [4]. Эти методы можно отнести к одному из двух типов: классификации и кластеризации.

Под классификацией понимают способы отнесения объектов (наблюдений, событий) к одному из заранее известных классов. Классификация относится к стратегии обучения с учителем, которое также именуют контролируемым или управляемым обучением.

Задача кластеризации сходна с задачей классификации, является ее логическим продолжением. Отличие задачи кластеризации состоит в том, что классы изучаемого набора данных заранее не предопределены. Другими словами, кластеризация (обучение без учителя) отличается от классификации (обучения с учителем) тем, что принадлежность к классу исходных объектов изначально не задана.

В самой общей постановке задача кластеризации выглядит следующим образом.

Пусть X — множество объектов, Y — множество номеров (имен, меток) кластеров. Задана функция расстояния между объектами $p(x, x')$. Имеется конечная обучающая выборка объектов $X^m = \{x_1, ..., x_m\} \in X$. Требуется разбить выборку на непересекающиеся подмножества, называемые кластерами, так, чтобы каждый кластер состоял из объектов, близких по метрике p, а объекты разных кластеров существенно (по выбранной метрике) отличались. При этом каждому объекту $x_i \in X^m$ приписывают номер кластера y_i [3].

В статье применительно к деталям машиностроения рассмотрен один из методов кластеризации — использование искусственных нейронных сетей, а именно — сетей самоорганизации.

Самоорганизующиеся нейронные сети

Свойством самоорганизации обладают нейронные сети, описанные Т. Кохоненом. Нейроны самоорганизующейся сети могут быть обучены выявлению групп (кластеров) векторов входа, обладающих некоторыми общими свойствами. Существуют самоорганизующиеся сети (сети Кохонена) с неупорядоченными нейронами, которые часто называют слоями Кохонена, и сети с упорядочением нейронов, которые называют картами Кохонена.

Кластерный анализ по исходным данным

В качестве инструментального средства для кластеризации с помощью нейронных сетей авторами была выбрана программная система MATLAB. Для создания самоорганизующихся нейронных сетей использованы следующие М-функции.

1. Функция *newc*, создающая конкурентный слой Кохонена.

Синтаксис функции *newc*:

$$net = newc(PR, S, KLR).$$

Входные параметры: PR — матрица размера $R\times2$ минимальных и максимальных значений R признаков; S — число нейронов в выходном слое; KLR — параметр скорости обучения слоя Кохонена (по умолчанию равен 0,01).

2. Функция *newsom*, отвечающая за создание карты Кохонена.

Синтаксис функции *newsom*:

$$net=newsom(PR, [d1, d2, ...], tfcn, dfcn, olr, osteps).$$

Входные параметры: PR — матрица размерности $R\times2$ минимальных и максимальных значений R признаков; $[d1,d2]$ — размер входного слоя карты (по умолчанию равен 5×8); *tfcn* — топология карты (по умолчанию используется гексагональная топология); *dfcn* — функция вычисления расстояния; *olr* — параметр скорости обучения на этапе упорядочивания (по умолчанию значение параметра равно 0,9); *osteps* — число циклов на этапе упорядочивания.

Обе функции возвращают структуру *net*, описывающую созданную сеть (слой или карту соответственно).

При создании слоя и карты Кохонена требуется указать число нейронов в выходном слое, которое совпадает с числом кластеров. В рассматриваемом случае было задано число 6, поскольку именно такое значение использовалось при анализе рассматриваемой выборки деталей другими методами кластеризации.

Процедура обучения проводится с помощью М-функции *train*.

Синтаксис функции *train*:

$$[net, pr, Y, E] = train(NET, P, T).$$

Входные параметры: NET — созданная необученная сеть; P — обучающая выборка; T — целевой вектор (может не задаваться).

Выходные параметры: *net* — новая сеть (обученная); *pr* — результат обучения (число итераций и функция выполнения); Y — выходы сети, подсчитанные по обучающей выборке; E — ошибки сети.

Отличительной чертой процесса обучения рассматриваемых сетей является то обстоятельство, что необходимо настроить веса синапсов нейронов, а не минимизировать ошибку обучения, так как нет целевого вектора. Экспериментально подбиралось число итераций обучения.

Для моделирования работы сети существует М-функция *sim*.

Синтаксис функции *sim*:

$$[Y, E] = sim(net, P).$$

Входные параметры: *net* — сеть; *P* — векторы, подаваемые на вход сети.

Выходные параметры: *Y* — выходы сети; *E* — ошибки сети.

Результатом моделирования работы обученной сети является вектор, содержащий номера классов для каждой детали. Авторами ранее были проведены эксперименты по использованию слоя и карты Кохонена для той же совокупности деталей [5].

График Эндрюса

В качестве одного из способов отображения классифицированных многомерных данных использовались кривые Эндрюса [6]. Если размерность данных равна m, каждая точка $x = (x_1, \ldots, x_m)$, где $x_i\,(i = 1, \ldots, m)$ — признаки детали, может быть представлена функцией в виде ряда Фурье

$$f_x(t) = x_1 2^{-\frac{1}{2}} + x_2 \sin t + x_3 \cos t + x_4 \sin 2t + x_5 \cos 2t + \ldots .$$

Таким образом, каждой детали соответствует линия на графике. Поскольку расстояния между кривыми Эндрюса линейно отображают расстояния между точками данных, то кривые, расположенные ближе друг к другу, соответствуют близким точкам. На рис. 1 (см. третью сторону обложки) приведен график Эндрюса. Как видно на графике, каждая кривая Эндрюса описывает отдельную деталь из исходной выборки и имеет определенный цвет, который зависит от вектора выхода сети, полученного в ходе кластеризации деталей машиностроения.

Так как число деталей невелико (34 кривые на графике), а число кластеров, как уже отмечалось ранее, равно шести, то график Эндрюса дает лишь приблизительное графическое представление о совокупности экземпляров деталей.

Факторный анализ

Для визуализации результатов кластеризации в целях понижения размерности пространства признаков можно также использовать факторный анализ. Модель факторного анализа может быть записана в следующем виде:

$$\mathbf{X} = \boldsymbol{\mu} + \Delta\mathbf{f} + \mathbf{e},$$

где $\mathbf{X}$ — вектор наблюдений многомерной случайной величины; Δ — матрица нагрузок простых факторов; $\boldsymbol{\mu}$ — вектор средних значений признаков многомерной случайной величины $\mathbf{X}$; $\mathbf{f}$ —вектор взаимно независимых стандартизованных факторов; $\mathbf{e}$ — вектор независимых специфических факторов. Вектор значений многомерной случайной величины $\mathbf{X} = \{X_1, X_2, \ldots, X_d\}$, где X_i — i-й признак многомерной случайной величины. Размер матрицы Δ равен $d \times m$, где d — число признаков объектов или размерность многомерной случайной величины X; m — число простых факторов. Элемент матрицы Δ_{ij} называется нагрузкой i-й переменной на j-й фактор, или наоборот, нагрузкой j-го фактора на i-ю переменную. Число элементов векторов $\boldsymbol{\mu}$, $\mathbf{f}$ и $\mathbf{e}$ равно d [7].

Каждый фактор зависит от сильно коррелирующих между собой признаков. Как следствие, происходит перераспределение дисперсии между признаками, в результате чего получается максимально простая и наглядная структура факторов. С этой целью была рассчитана корреляционная матрица для исходной выборки деталей машиностроения (табл. 1).

Для расчета корреляционной матрицы использована М-функция *corrcoef*.

Синтаксис функции *corrcoef*:

$$\mathbf{R} = corrcoef(\mathbf{X}).$$

Входные параметры: $\mathbf{X}$ — матрица наблюдений.

Выходные параметры: $\mathbf{R}$ — корреляционная матрица.

Для качественной оценки степени связи в теории корреляции применяют шкалу английского статистика Чеддока: слабая корреляция — 0,1...0,3; умеренная корреляция — 0,3...0,5; заметная корреляция — 0,5...0,7; высокая корреляция — 0,7...0,9; очень высокая (сильная) корреляция — 0,9...1,0. Опираясь на эту шкалу, можно отметить, что высокую корреляцию ($0{,}7 \leqslant R < 0{,}9$) имеют признаки 5 и 6. Заметно коррелируют признаки 1 и 3 ($0{,}5 \leqslant R < 0{,}7$). Умеренно коррелируют признак 1 с признаками 2, 5, 6, а признак 3 — с признаком 6. Не коррелируют друг с другом признаки 1 и 4, 2 и 3, 2 и 5. Осталь-

Таблица 1

Корреляционная матрица по исходным данным

Признаки	1. Максимальный диаметр	2. Высота	3. Число внешних диаметров	4. Число внутренних диаметров	5. Число конических поверхностей	6. Число резьб
1. Максимальный диаметр	1	0,316	0,586	0,01	0,467	0,444
2. Высота	0,316	1	−0,05	−0,294	0,094	0,208
3. Число внешних диаметров	0,586	−0,05	1	0,122	0,16	0,334
4. Число внутренних диаметров	0,01	−0,294	0,122	1	0,211	0,158
5. Число конических поверхностей	0,467	0,094	0,16	0,211	1	0,838
6. Число резьб	0,444	0,208	0,334	0,158	0,838	1

ные коэффициенты корреляции между признаками относятся к слабому уровню корреляции. Окончательный вывод о нагрузках признаков на факторы будет сделан после проведения факторного анализа.

Для проведения факторного анализа в системе MATLAB используют функцию *factoran*. Ее синтаксис:

[lambda, **psi**, **T**, **F]** = *factoran*(**X**, *m*, '*param*1', *value*1).

Входные параметры: **X** — исходная матрица наблюдений; *m* — число факторов; *param*1 — дополнительный входной параметр для управления работой алгоритма, *value*1 — его значение.

Выходные параметры: **lambda** — матрица факторных нагрузок (матрица значений коэффициентов корреляции между признаками и факторами); **psi** — вектор точечных оценок дисперсий специфических факторов; **T** — матрица вращения нагрузок; **F** — матрица рассчитанных факторных значений.

После проведения факторного анализа взаимная корреляция признаков, связанных с каждым фактором, будет выше, чем их коррелированность с другими признаками.

Важной составляющей факторного анализа является процедура вращения факторов, т. е. перераспределения дисперсии по определенному методу. Нагрузки, полученные в качестве одного из результатов факторного анализа, когда число факторов больше единицы, неоднозначны, и существует возможность получать различные эквивалентные множества нагрузок путем их преобразования — вращения. Цель ортогональных вращений — определение простой структуры факторных нагрузок [8]. Вращение методом "варимакс", предложенным Кайзером [9], позволяет упростить столбцы матрицы нагрузок, сводя значения элементов к 1 или 0. Наиболее стандартным вычислительным методом вращения является "варимакс", который максимизирует разброс квадратов нагрузок для каждого фактора, что приводит к увеличению больших и уменьшению малых значений факторных нагрузок.

При вызове этой функции необходимо определить число факторов. Максимально возможное число простых факторов определяется неравенством $(d + m) \leqslant (d - m)^2$, где d — число признаков, а m — число факторов [10]. В данном случае $d = 6$, следовательно, $m = 2$ или $m = 3$.

Важное место при выборе числа факторов занимают факторные нагрузки (корреляции между исходными признаками и факторами), полученные как один из результатов функции *factoran*. В табл. 2 приведены значения факторных нагрузок при $m = 3$.

Как видно из табл. 2, фактор 1 получил высокую нагрузку для признаков 5 и 6 и умеренную нагрузку для признака 1. Фактор 2 имеет очень высокую нагрузку для признака 3 и заметную для признака 1. На фактор 3 сильную нагрузку оказывает признак 2 и умеренную отрицательную признак 4. Каждый фактор получает нагрузку минимум от двух признаков.

Для $m = 2$ факторные нагрузки будут выглядеть так, как показано в табл. 3.

Фактор 1 вновь получил высокую нагрузку для признаков 5 и 6, но для признака 1 нагрузка значительно меньше, чем при $m = 3$. Фактор 2 получил сильную нагрузку для признака 1, умеренную для признака 2 и заметную для признака 3. Однако недостатком данного сокращения пространства признаков будет являться нагрузка для признака 4. Фактор 1 имеет нагрузку, равную 0,229, а фактор 2 имеет нагрузку, равную −0,047. Таким образом, эти два фактора описывают исходные данные в недостаточном объеме, так как признак 4 исходной выборки не коррелирует с факторами. В результате исходное пространство признаков рациональнее сократить до $m = 3$.

Таблица 2

Факторные нагрузки для трех факторов (вращение "варимакс")

Признаки	Фактор		
	1	2	3
1. Максимальный диаметр	0,457	0,584	0,182
2. Высота	0,211	0,073	0,972
3. Число внешних диаметров	0,081	0,984	−0,143
4. Число внутренних диаметров	0,167	0,06	−0,344
5. Число конических поверхностей	0,988	0,063	−0,122
6. Число резьб	0,832	0,272	0,013

Таблица 3

Факторные нагрузки для двух факторов (вращение "варимакс")

Признаки	Фактор	
	1	2
1. Максимальный диаметр	0,241	0,968
2. Высота	0,019	0,32
3. Число внешних диаметров	0,018	0,601
4. Число внутренних диаметров	0,229	−0,047
5. Число конических поверхностей	0,968	0,242
6. Число резьб	0,801	0,26

Одним из результатов факторного анализа является матрица признаков в координатах факторов размером 34 × 3. На рис. 2 приведен результат кластерного анализа по исходным данным в координатах факторов.

Видно, что детали разделены на шесть кластеров, каждая деталь отображена своим порядковым номером и соответствующей маркировкой, что дает наглядное отображение принадлежности деталей к тому или иному кластеру.

Кластерный анализ по факторным значениям

Поскольку полученные факторные значения в значительной степени описывают исходную выборку, то выполнение кластерного анализа с помощью слоя и карты Кохонена можно реализовать и для факторных значений. Уменьшение числа признаков во входной матрице снижает число нейронов во входном слое, что позволяет ускорить процесс обучения сети.

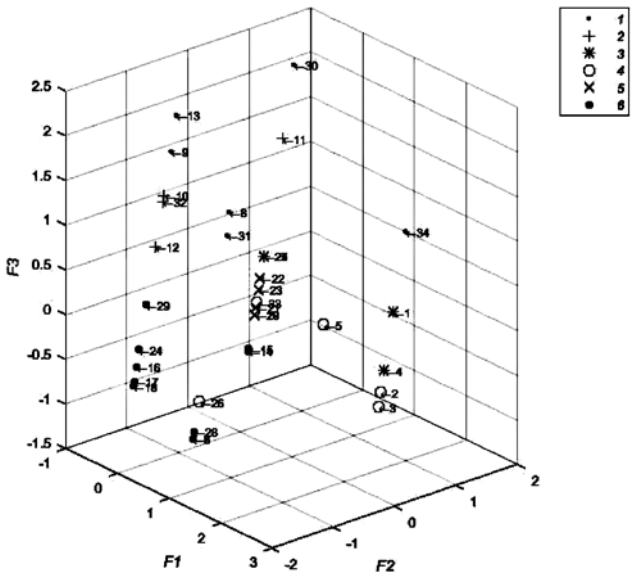


Рис. 2. Результат кластеризации по исходным данным в пространстве факторов:
F1, F2, F3 —факторы 1, 2 и 3 соответственно

На вход сетей подается матрица размера 34×3. Экспериментально подбирается число итераций обучения. На рис. 3 отображен результат кластеризации по факторным значениям.

Анализируя рис. 2 и 3, можно сказать, что результаты кластерного анализа по факторным значениям, с первого взгляда, выглядят более точными: все близкие друг к другу объекты отнесены к одному кластеру. Слой и карта Кохонена, оперируя факторными значениями (соответственно входной слой в обеих сетях имеет уже не шесть нейронов на входе, а три), быстрее проходят итерации обучения.

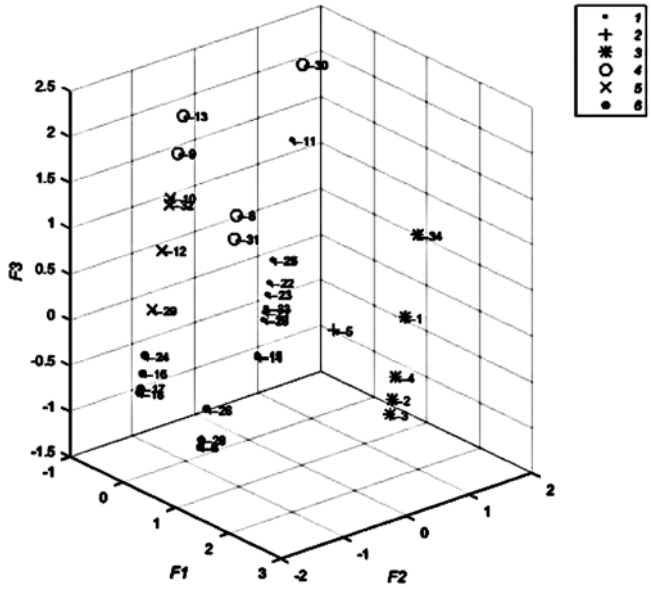


Рис. 3. Результат кластеризации по факторным значениям:
F1, F2, F3 — факторы 1, 2 и 3 соответственно

Однако факторные значения не могут полностью заменить исходную выборку, потому нельзя сказать, что результаты, полученные в ходе кластерного анализа по факторным значениям, более достоверные.

Сравнение результатов кластеризации

В работе [11] приведены результаты экспериментов с тем же набором деталей, но с применением других методов кластерного анализа (FCM, K-means, FOREL, иерархический).

В методе FOREL задано пороговое значение "кластерного радиуса" $D > 0$ и определяют гипотетический кластер S как множество объектов, которые ближе к центру тяжести множества, чем D, $S = \{i: d(i, g) < D\}$, где $d(i, g)$ — эвклидово расстояние между объектом i и центром тяжести g. Далее переопределяют S относительно нового центра на множестве, оставшемся после удаления объектов, уже отнесенных к какому-то кластеру; итерации продолжают до тех пор, пока не совпадут центр тяжести и центр нового кластера.

В методе *k*-means на E-шаге вычисляют ожидаемое значение функции правдоподобия; на M-шаге вычисляют оценку максимального правдоподобия, таким образом, увеличивается ожидаемое правдоподобие, вычисляемое на E-шаге. Затем это значение используют для E-шага на следующей итерации, которые продолжаются, пока не будет достигнута сходимость — найден конечный предел функции сходимости.

Метод нечеткой кластеризации (FCM) предполагает, что объекты принадлежат всем кластерам с определенной функцией принадлежности. Степень принадлежности определяется расстоянием от объекта до соответствующих кластерных центров. В этом методе итерационно вычисляют центры кластеров и новые степени принадлежности объектов до тех пор, пока сумма всех взвешенных расстояний не окажется минимальной.

В иерархических методах выделяют агломеративные и дивизимные методы. Дивизимные методы создают новые кластеры путем разбиения больших кластеров на меньшие, начиная с одного кластера, включающего все множество объектов. Агломеративные методы характеризуются последовательным объединением объектов и соответствующим уменьшением числа кластеров. В начале работы алгоритма все объекты являются отдельными кластерами. На первом шаге наиболее похожие объекты объединяются в кластер. На последующих шагах объединение продолжается до тех пор, пока все объекты не будут составлять один кластер.

В табл. 4 приведены сравнительные результаты кластерного анализа группы деталей совместно с результатами, полученными авторами с помощью нейронных сетей.

Как видно из данных табл. 4, существует сходство в полученных результатах. Некоторые детали практически всеми алгоритмами были отнесены к одному кластеру, а именно: 2, 3, 4, 5, 26, 33 и 19, 20, 21, 22, 23, 25, 27. В целом же результаты, полученные в ходе кластерного анализа по исходным данным с использованием сетей Кохонена, имеют наибольшее сходство с результатом алгоритма K-means.

Результаты кластерного анализа, проводимые по факторным значениям, также содержат идентичные

Таблица 4

Результы кластерного анализа

№ кластера	№ деталей							
	Метод FCM	Метод FOREL	Метод K-means	Иерархический метод	Слой Кохонена (исх.)	Карта Кохонена (исх.)	Слой Кохонена (факт.)	Карта Кохонена (факт.)
1	2, 3, 4, 5, 26, 33	2, 3, 4, 5, 26, 33	2, 3, 4, 5, 26, 33	1, 2, 3, 4, 5, 8, 9, 10, 11, 26, 30, 32, 33,34	2, 3, 4, 5, 26, 33	2, 3, 4, 5, 26, 33	1, 2, 3, 4, 5, 34	1, 2, 3, 4, 5, 34
2	1, 8, 10, 11, 30, 32, 34	1, 8, 9, 10, 11, 30, 32, 34	1, 10, 32	13	1, 10, 11, 32, 34	1, 11, 32	11, 22, 23, 25, 27, 30	11, 30
3	13, 31	13, 31	12, 13, 31	12, 29	12, 16, 18, 24, 29	12, 29	12, 16, 17, 18, 24, 29	12, 29
4	19, 20, 21, 23	19, 20, 21, 22, 23, 25, 27	19, 20, 21, 22, 23, 25, 27	19, 20, 21, 22, 23, 25, 27, 6, 7, 14, 15, 16, 17, 18, 28	19, 20, 21, 22, 23, 25, 27	19, 20, 21, 22, 23, 25, 27	14, 15, 19, 20, 21, 33	14, 15, 19, 20, 21, 22, 23, 25, 27, 33
5	12, 22, 25, 27	12, 24, 29	8, 9, 11, 30, 34	31	8, 9, 13, 30, 31	8, 9, 10, 13, 30, 31, 34	8, 9, 10, 13, 31, 32	8, 9, 10, 13, 31, 32
6	6, 7, 14, 15, 16, 17, 18, 24, 29	6, 7, 14, 15, 16, 17, 18, 28	6, 7, 14, 15, 16, 17, 18, 24, 28, 29	24	6, 7, 14, 15, 17, 28	6, 7, 14, 15, 6, 17, 18, 24, 28	6, 7, 26, 28	6, 7, 16, 17, 18, 24, 26, 28

Примечание: № деталей — порядковые номера деталей в таблице из 34 втулок; исх. — кластерный анализ по всем шести признакам; факт. — кластерный анализ по признакам в пространстве координат факторов.

наборы деталей в некоторых кластерах. Результаты, полученные от слоя Кохонена, полностью совпадают с результатами, полученными от карты Кохонена, в кластере № 1 и в кластере № 5. Оставшиеся кластеры содержат пересекающиеся множества деталей: сформированный слоем Кохонена кластер № 2 (11, 22, 23, 25, 27, 30) включает в себя элементы кластера № 2 (11, 30), сформированного картой Кохонена и т. д.

Результаты кластерного анализа с применением сетей Кохонена по исходным данным отличаются от результатов анализа по факторным значениям. Однако все разбиения на кластеры имеют некоторый общий набор деталей.

При использовании слоя Кохонена в кластере № 1 совпадают детали 2, 3, 4, 5; в кластере № 2 — 11; в кластере № 3 — 12, 16, 18, 24, 29; в кластере № 4 — 19, 20, 21; в кластере № 5 — 8, 9, 13, 31; в кластере № 6 — детали 6, 7, 28.

При использовании карты Кохонена в кластере № 1 совпадают детали 2, 3, 4, 5; в кластере № 2 — 11; в кластере № 3 перечень деталей полностью совпадает; в кластере № 4 — общий набор деталей 19, 20, 21, 22, 23, 25, 27; в кластере № 5 — 8, 9, 10, 13, 31; в кластере № 6 — детали 6, 7, 16, 17, 18, 24, 28.

Подобные расхождения результатов кластерного анализа по факторным и исходным значениям связаны с достаточно низкой корреляцией между исходными признаками и, как следствие, потерей некоторой части информации из исходной выборки в ходе факторного анализа. Тем не менее нельзя определить, какой из методов кластерного анализа дает более точные результаты на предложенной выборке, поскольку использовались не промышленные данные. По этой причине получить экспертную оценку не представляется возможным.

Тем не менее была сделана попытка сравнить результаты кластеризации с некоторым целевым вектором.

Принимая результат кластеризации методом K-means за целевой вектор, сравнивали результаты кластерного анализа по исходным данным; при этом использование слоя Кохонена дает восемь ошибок, что составило 23,53 %, карта Кохонена — пять ошибок (14,71 %). Сравнивая результаты кластерного анализа по факторным значениям с целевым вектором получили, что слой Кохонена дает 20 ошибок (58,82 %), карта Кохонена — 11 ошибок (32,35 %). Такие результаты связаны с небольшим объемом выборки.

Заключение

Для кластеризации выборки деталей машиностроения был привлечен аппарат нейронных сетей — слоя и карты Кохонена. Кроме визуализации результатов кластерного анализа с помощью графика Эндрюса, использован факторный анализ для сокращения размерности пространства признаков и вывода графика в пространстве координат факторов. Проведена кластеризация на основе полученных факторных значений. Кластерный анализ по факторным значениям проводится быстрее за счет сокращения объема данных и уменьшения числа нейронов в сети. Проведено также качественное сравнение результатов анализа с другими методами (K-means, FCM, FOREL, иерархический метод). В силу отсутствия экспертной оценки не представляется возможным выбрать наилучший метод кластеризации для рассматриваемой выборки деталей. По этой причине в дальнейшем следует применить предложенную методику к выборке деталей большего размера и с целевым вектором.

Список литературы

1. **Медведев В. С., Потемкин В. Г.** Нейронные сети. MATLAB 6. М.: ДИАЛОГ-МИФИ, 2002. 496 с.
2. **Митрофанов С. П.** Научная организация машиностроительного производства. Л.: Машиностроение, 1976. 712 с.
3. **Мандель И. Д.** Кластерный анализ. М.: Финансы и статистика, 1988. 176 с.
4. **Миркин Б. Г.** Методы кластер-анализа для поддержки принятия решений: обзор. М.: Изд. дом Национального исследовательского университета "Высшая школа экономики", 2011. 88 с.
5. **Харахинов В. А., Сосинская С. С.** Классификация деталей машиностроительного производства с использованием нейронных сетей // Винеровские чтения. 2016. С. 19—23.
6. **Грошев С. В., Пивоварова Н. В.** Использование кривых Эндрюса для визуализации многомерных данных в задачах многокритериальной оптимизации // Наука и Образование. 2015. № 12. С. 197—214.
7. **MATLAB** Documentation. URL: http:// mathworks.com/help/stats/factoran.html
8. **Харман Г.** Современный факторный анализ. М.: Статистика, 1972. 484 с.
9. **Kaiser H. F.** The varimax criterion for analytic rotation in factor analysis // Phychometrika. 1958. Vol. 23. P. 187—200.
10. **Лоули Д., Мансвелл А.** Факторный анализ как статистический метод. М.: Мир, 1967. 141 с.
11. **Янчуковский В. Н., Сосинская С. С.** Параллельный итерационный кластерный анализ на основе алгоритма FOREL // Современные технологии. Системный анализ. Моделирование. 2013. № 3 (39). С. 94—99.

The Visualization Methods for Cluster Analysis Results of Mechanical Engineering Components based on Neural Network

V. A. Kharakhinov, tes4obse@mail.ru**, S. S. Sosinskaya,** sosinskaya@mail.ru**,**
Irkutsk National Research Technical University, Irkutsk, 664074, Russian Federation

Corresponding author:

Sosinskaya Sophia S., Professor, Irkutsk National Research Technical University, Irkutsk, 664074, Russian Federation,
E-mail: sosinskaya@mail.ru

Received on November 30, 2016
Accepted on January 11, 2017

This paper considers the research results of clustering analysis based on Kohonen networks for the mechanical engineering components. Some ways to visualize these results were discussed in the paper. The first way is based on Andrews curves. It provides only an approximate graphical representation for this data. The next way uses factor analysis to reduce dimension of initial data and to visualize these data in lower dimension. The cluster analysis was applied to the factors values for time reducing of network training. A comparison of the results of cluster analysis of mechanical engineering components on the basis of self-organizing networks with outputs obtained using other clustering methods is given.

Keywords: *cluster analysis, Kohonen networks, Andrews curves, factor analysis, mechanical engineering components*

For citation:

Kharakhinov V. A., Sosinskaya S. S. The Visualization Methods for Cluster Analysis Results of Mechanical Engineering Components based on Neural Network, *Programmnaya Ingeneria*, 2017, vol. 8, no. 4, pp. 170—176.

DOI: 10.17587/prin.8.170-176

References

1. **Medvedev V. S., Potemkin V. G.** *Nejronnye seti. MATLAB 6* (Neural Networks. MATLAB 6), Moscow, DIALOG-MIFI, 2002, 496 p. (in Russian).
2. **Mitrofanov S. P.** *Nauchnaja organizacija mashinostroitel'nogo proizvodstva* (Scientific organization of engineering production), Leningrad, Mashinostroenie, 1976, 712 p. (in Russian).
3. **Mandel' I. D.** *Klasternyj analiz* (Cluster analysis), Moscow, Finansy i statistika, 1988, 176 p. (in Russian).
4. **Mirkin B. G.** *Metody klaster-analiza dlja podderzhki prinjatija reshenij: obzor* (Methods of cluster analysis for decision support: overview), Moscow, Izd. dom Nacional'nogo issledovatel'skogo universiteta "Vysshaja shkola jekonomiki", 2011, 88 p. (in Russian).
5. **Kharahinov V. A., Sosinskaya S. S.** Klassifikacija detalej mashinostroitel'nogo proizvodstva s ispol'zovaniem nejronnyh setej (Classification of mechanical engineering components based on neural network), *Vinerovskie chtenija*, 2016, pp. 19—23 (in Russian).
6. **Groshev S. V., Pivovarova N. V.** Ispol'zovanie krivyh Jendrjusa dlja vizualizacii mnogomernyh dannyh v zadachah mnogokriterial'noj optimizacii (Using the Andrews Plots to visualize multidimensional data in multi-criteria optimization), *Nauka i Obrazovanie*, 2015, no. 12, pp. 197—214 (in Russian).
7. **MATLAB** Documentation, available at: http://mathworks.com/help/stats/factoran.html
8. **Harman G.** *Sovremennyj faktornyj analiz* (Modern factor analysis), Moscow, Statistika, 1972, 484 p. (in Russian).
9. **Kaiser H. F.** The varimax criterion for analytic rotation in factor analysis, *Phychometrika*, 1958, vol. 23, pp. 187—200.
10. **Louli D., Mansvell A.** *Faktornyj analiz kak statisticheskij metod* (Factor analysis as a statistical method), Moscow, Mir, 1967. 141 p. (in Russian).
11. **Janchukovskij V. N., Sosinskaya S. S.** Parallel'nyj iteracionnyj klasternyj analiz na osnove algoritma FOREL (Parallel iterative cluster analysis based on FOREL algorithm), *Sovremennye tehnologii. Sistemnyj analiz. Modelirovanie*, 2013, no. 3 (39), pp. 94—99 (in Russian).

УДК 004.832.38 DOI: 10.17587/prin.8.177-185

А. Н. Шмарин, аспирант, e-mail: tim-shr@mail.ru,
Воронежский государственный университет

Кластеризация множества слоев в задаче нечеткого LP-вывода

LP-структуры представляют алгебраическую модель логических систем продукционного типа, широко распространенных в информатике. Представлен алгоритм кластеризации слоев в LP-структуре. Слой — это одноэлементная выборка из классов эквивалентности фактор-множества, полученного разбиением LP-структуры по уникальным правым частям пар бинарного отношения. Исследованы вопросы практической реализации данного алгоритма. Результаты могут быть использованы для разработки интеллектуальных систем продукционного типа, работающих с неполными знаниями, а также минимизирующих число трудоемких запросов о значениях признаков классифицируемых объектов предметной области.

Ключевые слова: *LP-вывод, бинарное отношение, фактор-множество, кластеризация, стохастический алгоритм, машинное обучение*

Введение

С увеличением объемов и сложности обрабатываемой информации все более актуальным становится использование систем искусственного интеллекта. При этом одной из наиболее распространенных моделей представления знаний является продукционная модель [1]. Алгоритмы поиска решений в системах искусственного интеллекта основаны на механизме логического вывода. На практике существует большое число задач принятия решений в условиях неопределенности, для которых получение информации о значениях признаков классифицируемых объектов некоторой предметной области является ресурсоемкой операцией. Такие задачи возникают, например, когда робот исследует поверхность другой планеты. Если цель робота состоит в том, чтобы добраться до определенной скалы, но будущий маршрут является наблюдаемым лишь отчасти, то робот должен попытаться выработать оптимальное решение, касающееся того, как достичь цели с учетом ограниченности ресурсов, доступных для дополнительной разведки местности. Другим примером, в котором стоимость получения новой информации может быть высока, является коммерческая медицина. Для минимизации издержек необходимо находить приемлемый способ лечения с помощью минимального количества анализов, выполненных за минимальное время. Задержка в предоставлении лечения, вызванная проведением дополнительных анализов, приводит к увеличению затрат. Более того, состояние пациента, не дождавшегося помощи, может существенно ухудшиться. Стоимость дополнительных исследований является существенной и в сфере разведки полезных ископаемых. Может оказаться, что более выгодное решение состоит в том, чтобы приступить к бурению скважины, если уверенность в успехе составляет 95 %, чем потратить значительные средства, чтобы добиться 98 %-ной уверенности.

С фактором ресурсоемкости операций получения данных для принятия решения связана задача минимизации числа запросов о признаках классифицируемого объекта (из некоторой предметной области) в ходе логического вывода. Данная задача является NP-трудной [2]. Для достижения глобальной минимизации числа внешних запросов предложен общий метод LP-вывода [3]. К сожалению, он обладает экспоненциальной вычислительной сложностью относительно числа атомарных фактов в базе знаний. Идея кластерно-релевантного LP-вывода [4] основана на вычислении специфических оценок (показателей релевантности) для ограниченного подмножества продукций и на обобщении полученных оценок на все множество продукций. Исследования и эксперименты показывают [5], что использование метода LP-вывода по сравнению с использованием обычного обратного вывода снижает число внешних запросов в среднем на 15...20 %.

Ранее [6] обсуждались вопросы реализации вычисления приближенной оценки числа слоев без циклов в задаче нечеткого LP-вывода. В настоящей работе рассмотрена математическая формулировка задачи нечеткого LP-вывода, исследованы свойства монотонности функции оценки числа слоев без циклов, а также предложен алгоритм кластеризации множества слоев — выборок по одному элементу из классов эквивалентности фактор-множества, полученного разбиением заданного на алгебраической решетке бинарного отношения по уникальным пра-

вым частям пар. Данный алгоритм предназначен для итеративного анализа таких подмножеств продукций, которые наиболее существенно влияют на показатель релевантности.

О методе нечеткого LP-вывода

Обозначим через X множество объектов (ситуаций, прецедентов) некоторой предметной области. Например, в задачах машинного обучения, встречающихся в медицине, объектами могут являться пациенты, в сфере кредитования — заемщики, в задаче фильтрации спама — отдельные сообщения.

Признак — результат измерения некоторой характеристики объекта, т. е. отображение *feature*: $X \to D_{feature}$, где $D_{feature}$ — множество допустимых значений признака. Значениями признаков в прикладных задачах могут быть числовые последовательности, изображения, тексты, функции, графы, результаты запросов к базе данных и т. д.

Пусть имеется набор признаков $feature_1, ..., feature_n$, которые соответствуют характеристикам объектов $x \in X$ некоторой предметной области. Вектор $(feature_1(x), ..., feature_n(x)) \in D_{feature_1} \times ... \times D_{feature_n}$ называется признаковым описанием объекта $x \in X$. Признаковое описание можно отождествлять с самими объектами, т. е. $X = D_{feature_1} \times ... \times D_{feature_n}$.

Предположим, что объекты X разделены некоторым образом на классы, которым соответствует конечное множество их номеров (имен, меток) Y. Пусть также задана продукционная база знаний, отражающая целевую зависимость — отображение $y^* : X \to Y$.

Задача нечеткого LP-вывода состоит в том, чтобы как можно достовернее классифицировать произвольный объект $x \in X$, т. е. найти для него класс $y^*(x) \in Y$, используя для этого как можно меньшую выборку значений из признакового описания этого объекта.

Далее рассмотрена более формализованная постановка задачи в терминах алгебраических решеток и бинарных отношений.

Пусть задано конечное множество $F = \{ f \}$. На его основе определим атомно-порожденную ограниченную алгебраическую решетку $L = \lambda(F)$, где λ — функция-булеан [7]. На решетке зададим дополнительное бинарное отношение $R = \{r = (\tau, f): \tau \in L, f \in F\}$, являющееся каноническим [5].

Применительно к продукционным логическим системам такое отношение R соответствует множеству продукций, F (множество атомов решетки) соответствует элементарным фактам продукционной системы, а сама решетка L соответствует множеству предпосылок и заключений продукций.

Атом x решетки L называется начальным при отношении R, если в R нет ни одной пары вида (τ, x).

Обозначим множество всех начальных атомов решетки как F_{init}: $F_{init} = \{f \in F : \nexists\, (\tau, f) \in R, \tau \in L\}$. Также обозначим множество всех атомов решетки, не являющихся начальными: $F_{notinit} = \{f \in F : \exists (\tau, f) \in R, \tau \in L\}$.

Упорядоченная пара $\{\alpha, \beta) \in L$, $\alpha \in L$, $\beta \in L$ называется дистрибутивно связанной отношением R, если существует такое множество $\{(\alpha_1, \beta_1), ..., (\alpha_p, \beta_p)\}$, что $\forall i = \overline{1, p}$ выполняется следующее условие: $\alpha_i \subseteq \alpha \wedge \beta \subseteq \beta_i \wedge ((\alpha_i = \beta_i \in L) \vee ((\alpha_i, \beta_i) \in R))$.

Упорядоченная пара (α, β), $\alpha \in L$, $\beta \in L$ называется логически связанной отношением R, если она дистрибутивно связана отношением R, либо существует упорядоченный набор $(\alpha, \gamma_1, ..., \gamma_l, \beta)$, $\gamma_i \in L$, $i = \overline{1, l}$ такой, что каждая пара в последовательности (α, γ_1), (γ_1, γ_2), ..., (γ_{l-1}, γ_l), (γ_l, β) дистрибутивно связана отношением R. Если пара (α, β) логически связана отношением R, то β будем называть образом α при отношении R, а α — прообразом β при отношении R. Прообраз в атомно-порожденной решетке называется начальным, если все его атомы являются начальными (при отношении R).

Введем отображение $M_R : R \to [0, 1]$, которое каждому элементу отношения R ставит в соответствие степень истинности: $M_R(r) = \mu$, $r \in R$, $\mu \in [0, 1]$. Обозначим также $M_{init} : F_{init} \to [0, 1]$ — отображение, ставящее в соответствие некоторую степень истинности каждому начальному атому решетки.

Применение отображения M_{init} соответствует выполнению запроса о неизвестном значении некоторого признака классифицируемого объекта и может оказаться ресурсоемкой операцией.

Применительно к продукционным системам отображение M_R вычисляет степени истинности продукций, а M_{init} — степени истинности значений признаков. Степень истинности задается числом $\mu \in [0, 1]$ и означает коэффициент уверенности, с которым соответствующее утверждение истинно. Пусть, например, продукции вида "если $x_1 \wedge ... \wedge x_n$, то x_{n+1}" соответствует некоторый элемент $r \in R$. Тогда значение $M_R(r) \in [0, 1]$ будет означать степень, с которой истинно утверждение данной продукции. Аналогично, если для объекта предметной области $x \in X$ значению некоторого его признака $feature(x) \in D_{feature}$ соответствует атом $f \in F_{init}$, то значение $M_{init}(f) \in [0, 1]$ будет представлять степень, с которой истинно утверждение "признак *feature* объекта x равен значению $feature(x)$".

Во введенных обозначениях задача нечеткого LP-вывода заключается в том, чтобы в некотором подмножестве $F_C \subseteq F_{notinit}$ неначальных атомов решетки найти элемент, обладающий наибольшей степенью истинности $M_R(X)$, применив при этом отображение M_{init} как можно меньшее число раз. Данный элемент соответствует решению, имеющему наибольшую степень истинности. Множество F_C соответствует некоторому множеству классов, на которые разделены объекты предметной области.

Процесс поиска обладающего наибольшей степенью истинности $f_c \in F_C$ состоит из нескольких шагов. В первую очередь, для каждого атома решетки, принадлежащего множеству $F_C \subseteq F_{notinit}$, выполняется поиск начальных прообразов при отношении R (т. е. поиск для атома логически связанных отношением R подмножеств начальных атомов решетки). Затем на

основе результатов поиска вычисляются степени истинности соответствующих атомов решетки. В заключение выбирается атом с наибольшей степенью истинности.

Предположим, что заданы некоторые бинарные операции T и T_{con} такие, что T является операцией t-нормы, а T_{con} — соответствующей ей операцией t-конормы [8]. Вычисление степеней истинности выполняется для всевозможных атомов $f_c \in F_C$ и соответствующих им начальных прообразов $\alpha \in L$. Для вычисления используют отображения M_R, M_{init}, операции t-нормы и t-конормы, а также обобщенное правило вывода *modus ponens*.

По каждому атому $f_c \in F_C$ формируется либо восстанавливается из кэш-памяти подмножество $R_1 \subseteq R$, которое логически связывает выбранные атом x и начальный прообраз α. Нахождение степени истинности некоторого неначального атома f, участвующего в логической связи α и x, начинается с выбора подмножества элементов, содержащих атом f в своей правой части: $R_f = \{(\tau, f) \in R_1,\ \tau \in L\}$. Обозначим мощность данного подмножества символом $n = |R_f|$. Тогда $R_f = \{r_1, ..., r_n\} = \{(\tau_1, f), ..., (\tau_n, f)\}$, где $\tau_i \in L$, $1 \leqslant i \leqslant n$. По определению, элементы τ_i решетки L состоят из ее атомов. То есть $\tau_i = \{g_{i,1}, ..., g_{i,m_i}\}$, где $g_{i,j} \in F$ — атомы решетки, m_i — число таких атомов в τ_i, $1 \leqslant j \leqslant m_i$, $1 \leqslant i \leqslant n$.

Используя введенные обозначения, можно выразить формулы для вычисления $\mu(f)$ — степени истинности некоторого атома f. Если $f \in F_{notinit}$, то атом является неначальным и его степень истинности задается рекурсивно:

$$\mu(f) = T_{con}\Big(T\Big(T\Big(\mu\big(g_{1,1}\big),\ ...,\ \mu\big(g_{1,m_1}\big)\Big),\ M_R(r_1)\Big),\ ...,$$
$$T\Big(T\Big(\mu\big(g_{n,1}\big),...,\mu\big(g_{n,m_n}\big)\Big),\ M_R(r_n)\Big)\Big),$$

где $r_i = (\tau_i, f) \in R_f$, $\tau_i = \{g_{i,1},\ ...,\ g_{i,m_i}\}$, $g_{i,j} \in F$, $1 \leqslant j \leqslant m_i$, $1 \leqslant i \leqslant n$.

Если же $f \in F_{init}$, т. е. атом является начальным, то $\mu(f) = M_{init}(f)$. Применение отображения M_{init} к некоторому начальному атому означает нахождение для этого атома соответствующей степени истинности. Поскольку атом является начальным, его степень истинности невозможно вывести на основе отношения R или степеней истинности других атомов решетки.

Поиск такой степени истинности эквивалентен выполнению внешнего запроса о значении неизвестного признака классифицируемого объекта. Минимизация числа подобных запросов достигается специальным порядком применения отображения M_{init} к начальным атомам решетки. В первую очередь, это отображение следует применять к тем начальным атомам, которые входят в наибольшее число найденных прообразов. Кроме того, соответствующие прообразы должны содержать как можно меньшее число элементов.

Порядок применения отображения M_{init} к начальным атомам решетки определяется на основе так называемых показателей релевантности [9]. Нахождение их точных значений является вычислительно трудной задачей [2]. Таким образом, для практического использования целесообразно вычислять показатели релевантности в ограниченных "кластерах" LP-структуры.

Обозначим символом A фактор-множество, являющееся разбиением бинарного отношения R на классы эквивалентности относительно уникальных правых частей его элементов.

То есть, если $r,\ t \in R$, $r = (\tau_r, f_r)$, $t = (\tau_t, f_t)$, то $r \sim \Leftrightarrow f_r = f_t$;

$$A = R/\sim \Leftrightarrow A = \Big\{\alpha_f = \big\{r = (\tau, f) : r \in R\big\} : f \in F\Big\}.$$

Данное фактор-множество представляет основу структурного расслоения [3] исходного отношения R на частичные отношения. Следует отметить, что в силу своего построения число классов эквивалентности фактор-множества A равно числу всех начальных атомов решетки, т. е. $|A| = |F_{notinit}|$. *Слоем* называется выборка по одному элементу из каждого класса эквивалентности $\alpha \in A$: $l = \Big\{r_1 \in \alpha_1,\ ...,\ r_{|F_{notinit}|} \in \alpha_{|F_{notinit}|}\Big\}$.

Нахождение показателей релевантности основано на поиске прообразов в таких слоях. Слой либо содержит прообраз, либо его элементы образуют цикл и не могут содержать прообраз [9].

Обозначим $S(A)$: $A \to \lambda(R)$ отображение, переводящее фактор-множество $A = \Big\{\alpha_1,\ ...,\ \alpha_{|F_{notinit}|}\Big\}$ в множество всех слоев, которые A описывает.

$$S(A) = \Big\{\Big\{r_1,...,r_{|A|}\Big\} \Big| r_i \in \alpha_i,\ \ i = \overline{1,|A|}\Big\} =$$
$$= \bigcup_{(r_1,...,r_{|A|}) \in \alpha_1 \times ... \times \alpha_{|A|}} \Big\{r_1,...,r_{|A|}\Big\}.$$

Если фактор-множество A состоит из классов α_1, ..., α_n, то число слоев на основе A равно числу всевозможных выборок из каждого класса эквивалентности по одному элементу, т. е. $|S(A)| = |\alpha_1| \cdot ... \cdot |\alpha_n|$.

Для кластеризации множества слоев могут быть использованы свойства логической связанности пар бинарного отношения, ассоциированные с этими парами значения степеней истинности, а также оценки числа слоев без циклов, включающих в себя заданный атом решетки.

Оценка числа слоев без циклов, включающих в себя заданный атом решетки

Ранее [10] были рассмотрены вопросы оценки числа слоев без циклов, включающих в себя заданный атом решетки. Далее приведем основные определения и результаты из работы [10], а затем рассмотрим свойства монотонности для функции оценки числа слоев без циклов, необходимые для нахождения ее приближенных значений.

Отображения $in(\alpha, f) = \big\{(\tau, g) \in \alpha : f \in \tau\big\}$ и $\overline{in}(\alpha, f) = \big\{(\tau, g) \in \alpha : f \notin \tau\big\}$ переводят класс эквивалентно-

сти α в множество пар, которые соответственно содержат либо не содержат в своих левых частях атом *f*. Следует заметить, что $in(\alpha, f) = \alpha \backslash \overline{in}(\alpha, f)$ и $\overline{in}(\alpha, f) = \alpha \backslash in(\alpha, f)$. Функции *in* и $\overline{in}$ можно обобщить на все фактор-множество *A*:

$$in(A, f) = \left\{ \bigcup_{a \in A} \{in(\alpha, f)\} \right\}, \quad \overline{in}(A, f) = \left\{ \bigcup_{a \in A} \{\overline{in}(\alpha, f)\} \right\},$$

а также на некоторое подмножество атомов $\upsilon \subseteq F$:

$$in(\alpha, \upsilon) = \{(\tau, g) \in \alpha : \upsilon \subseteq \tau\} \text{ и}$$

$$\overline{in}(\alpha, \upsilon) = \{(\tau, g) \in \alpha : \upsilon \not\subseteq \tau\};$$

$$in(A, \upsilon) = \left\{ \bigcup_{a \in A} \{in(\alpha, \upsilon)\} \right\},$$

$$\overline{in}(A, \upsilon) = \left\{ \bigcup_{a \in A} \{\overline{in}(\alpha, \upsilon)\} \right\}.$$

Отображение $\lambda(X, k, i)\colon X \to \lambda(X), \quad k = \overline{1, |X|}, \quad i = \overline{1, \binom{|X|}{k}}$ переводит множество *X* в *i*-й элемент множества всех его *k*-элементных подмножеств.

Отображение $A_f = \{\alpha \in A | \exists (\tau, g) \in \alpha : f \in \tau\} = \{\alpha \in A : \text{in}(\alpha, f) \neq \varnothing\}$ переводит фактор-множество *A* в такое подмножество его классов эквивалентности, что каждый элемент подмножества включает в себя хотя бы одну пару, содержащую *f* в левой части.

На основе бинарного отношения *R* построим граф $G = \{\nu\}$:

$$\nu \in G, \ \nu = (f, g), \ f, g \in F \Leftrightarrow \exists (\tau, g) \in R : f \in \tau.$$

По графу *G*, используя некоторый из известных математических методов, построим фундаментальную систему циклов $H = \{h : h \subseteq G\}$. На основе отношения *R* и системы циклов *H* определим *E — фундаментальную систему циклов канонического бинарного отношения*: на основе дуг $\nu = (f, g), f, g \in F$, образующих элементы фундаментальной системы циклов $H = \{h\}$, построим множество $E = \{e_1, ..., e_{|S|}\}$, состоящее из подмножеств отношения *R* таких, что $\forall h = \{\nu = (f, g) \in G\}$: $e_i = \{(\tau, g) \in R : f \in \tau\}$.

Пусть $\varepsilon \subseteq E$, $\varepsilon = \{e \in E\}$ — подмножество фундаментальных циклов канонического бинарного отношения *R*, $\dot{\varepsilon}(\varepsilon) = \bigcup_{e \in \varepsilon} \bigcup_{r \in e} r$ — объединение всех пар бинарного отношения, формирующих элементы данного подмножества. Сужение фактор-множества *A*, описывающее всевозможные слои, которые содержат только фундаментальные циклы из множества ε, определяется следующим отображением:

$$\theta(\alpha, \varepsilon) = \begin{cases} \alpha \cap \dot{\varepsilon}(\varepsilon), & \text{если } \alpha \cap \dot{\varepsilon}(\varepsilon) \neq \varnothing \\ \alpha, & \text{иначе} \end{cases}, \ \varepsilon \subseteq E,$$

$$\Theta(A, \varepsilon) = \left\{ \theta(\alpha_1 \varepsilon), \ ..., \ \theta\left(\alpha_{|A|}, \varepsilon\right) \right\}, \ \varepsilon \subseteq E.$$

Определим специальные отображения $S_{all}(A, \Theta, \varepsilon)$ и $S_{subsets}(A, \Theta, \varepsilon)$.

- $S_{all}(A, \Theta, \varepsilon)$ — множество всевозможных слоев, включающих в себя все циклы из множества $\varepsilon \subseteq E$.

$$S_{all}(A, \Theta, \varepsilon) = S\left(\Theta\left(A, \varepsilon\right)\right), \ \varepsilon \subseteq E;$$

$$\left|S_{all}(A, \Theta, \varepsilon)\right| = \left|\theta(\alpha_1, \varepsilon)\right| \cdot ... \cdot \left|\theta\left(\alpha_{|A|}, \varepsilon\right)\right|,$$

$$A = \left\{\alpha_1, ..., \alpha_{|A|}\right\}.$$

Отображение S_{all} можно обобщить на подмножество атомов $\upsilon \subseteq F$:

$$\vartheta(\alpha, \upsilon) = \left\{(\tau, g) \in \alpha : \upsilon \cap \tau \neq \varnothing\right\}, \ \alpha \in A, \ \upsilon \subseteq F;$$

$$\theta(\alpha, \upsilon) = \begin{cases} \vartheta(\alpha, \upsilon), & \text{если } \vartheta(\alpha, \upsilon) \neq \varnothing \\ \alpha, & \text{иначе} \end{cases}, \ \upsilon \subseteq F;$$

$$\Theta(A, \upsilon) = \left\{\theta(\alpha_1 \upsilon), \ ..., \ \theta\left(\alpha_{|A|}, \upsilon\right)\right\}, \ \upsilon \subseteq F;$$

$$S_{all}(A, \upsilon) = S\left(\Theta\left(A, \upsilon\right)\right), \ \upsilon \subseteq F;$$

$$\left|S_{all}(A, \upsilon)\right| = \left|\theta(\alpha_1, \upsilon)\right| \cdot ... \cdot \left|\theta\left(\alpha_{|A|}, \upsilon\right)\right|.$$

- $S_{subsets}(A, \Theta, \varepsilon)$ — множество всевозможных слоев, включающих в себя всевозможные подмножества циклов из множества $\varepsilon \subseteq E$.

В работе [10] показано, что для фундаментальной системы циклов $E = \{x_1, ..., x_l\}$ канонического бинарного отношения $S_{subsets}$ может быть выражено в следующем виде:

$$\left|S_{subsets}(A, \Theta, E)\right| = \sum_{x_1 \in E} \left|S_{all}\left(A, \Theta, \{x_1\}\right)\right| -$$

$$- \sum_{\substack{\{x_1, x_2\} \subset E, \\ x_1 \neq x_2}} \left|S_{all}\left(A, \Theta, \{x_1, x_2\}\right)\right| +$$

$$+ ... + (-1)^{l+1} \left|S_{all}\left(A, \Theta, \{x_1, ..., x_l\}\right)\right| =$$

$$= \sum_{k=1}^{l} (-1)^{k+1} \sum_{i=1}^{\binom{l}{k}} \left|S_{all}\left(A, \Theta, \{\lambda(E, k, i)\}\right)\right|.$$

Определим вспомогательные отображения:

$$F(A_1, \Theta, \varepsilon) = \left\{e \in \varepsilon | \forall \alpha_e \in e / \sim \exists \alpha_{A_1} \in A_1 / \sim : \alpha_e \subseteq \alpha_{A_1}\right\},$$

$$\varepsilon \subseteq E;$$

$$\tilde{A}(i, s, f) = in\left(\lambda\left(A_f, i, s\right), f\right) \cup \overline{in}\left(A \backslash \lambda\left(A_f, i, s\right), f\right),$$

$$f \in F.$$

Общее число слоев для атома f вычисляется как сумма [10]:

$$W(A, f) = \sum_{i=1}^{|A_f|} \sum_{s=1}^{\binom{|A_f|}{i}} \prod_{\beta \in \tilde{A}(i, s, f)} |\beta|.$$

Число слоев без циклов, включающих в себя заданный атом F, вычисляется следующим образом:

$$W_{noloops}(A,\ f)=\sum_{i=1}^{|A_f|}\sum_{s=1}^{\binom{|A_f|}{i}}\left(\prod_{\beta\in\tilde{A}(i,s,f)}|\beta|-\left|S_{subsets}\left(\tilde{A}(i,\ s,\ f),\ \Theta,\ F\left(\tilde{A}(i,\ s,\ f),\ \Theta,\ \varepsilon\right)\right)\right|\right).$$

Следует отметить, что число элементов в приведенных суммах растет экспоненциально относительно значения $|A_f|$.

Свойства монотонности оценок числа слоев без циклов

Рассмотрим взаимосвязь значений числа слоев для различных выборок.

Утверждение 1. Между элементами фактор-множеств $\tilde{A}(i,\ s,\ f)$ и A имеет место соотношение:

$$\forall i,\ s,\ f:\left|S\left(\tilde{A}(i,\ s,\ f)\right)\right|\leqslant|S(A)|.$$

Доказательство. Из определения отображений *in* и $\overline{in}$ следует, что $|in(\alpha,\ f)|\leqslant|\alpha|\wedge|(\alpha,\ f)|\leqslant|\alpha|$. Следовательно

$$\left|S\left(\tilde{A}(i,\ s,\ f)\right)\right|=\prod_{\alpha\in\lambda\left(A_f,i,s\right)}|in(\alpha,\ f)|\prod_{\alpha\in A\setminus\lambda\left(A_f,i,s\right)}\overline{in}|\alpha,\ f|\leqslant\prod_{\alpha\in A}|\alpha|=|S(A)|.$$

Утверждение 2. $\forall l,\ m,\ s_l,\ s_m:\ l\in\{1,\ ...,|\ A_f|\}\wedge m\in\{1,\ ...,|\ A_f|\}\wedge s_l\in\left\{1,\ ...,\binom{|A_f|}{l}\right\}\wedge s_m\in\left\{1,\ ...,\ \binom{|A_f|}{m}\right\}$, выполняется

$$\left|S\left(\tilde{A}(i,\ s_l,\ f)\right)\right|=\frac{C\left(A_f,\ l,\ m,\ s_l,\ s_m\right)}{C\left(A_f,\ m,\ l,\ s_m,\ s_l\right)}\cdot\left|S\left(\tilde{A}(m,s_m,f)\right)\right|,$$

где

$$C(X,\ l,\ m,\ i,\ j)=\begin{cases}\prod\limits_{\alpha\in\lambda(X,l,i)\setminus\lambda(X,m,j)}\dfrac{|in(\alpha,\ f)|}{|\overline{in}(\alpha,\ f)|}, & \text{если}\quad\lambda(A_f,\ l,\ i)\setminus\lambda(A_f,\ m,\ j)\neq\varnothing\\ 1, & \text{иначе}\end{cases}.$$

Данное утверждение позволяет выразить число слоев для левой выборки через число слоев правой выборки, скорректированное соответственно изменениям в выборках классов эквивалентности.

Доказательство. Зафиксируем некоторые $\{\alpha_1,\ \alpha_2\}\subseteq A_f,\ i<|\ A_f|,\ j<|\ A_f|,\ s_1\in\left\{1,\ ...,\ \binom{|A_f|}{i}\right\},\ s_2\in\left\{1,\ ...,\ \binom{|A_f|}{j}\right\}$ такие, что $\lambda(A_f,\ i,\ s_1)\setminus\lambda(A_f,\ j,\ s_2)=\{\alpha_1\}\wedge\lambda(A_f,\ j,\ s_2)\setminus\lambda(A_f,\ i,\ s_1)=\{\alpha_2\}$. Тогда:

$$\left|S\left(in\left(\lambda\left(A_f,\ j,\ s_2\right),\ f\right)\right)\right|=\frac{|in(\alpha_2,\ f)|}{|in(\alpha_1,\ f)|}\cdot\left|S\left(in\left(\lambda\left(A_f,\ i,\ s_1\right),\ f\right)\right)\right|,$$

$$\left|S\left(\overline{in}\left(A\setminus\lambda\left(A_f,\ j,\ s_2\right),\ f\right)\right)\right|=\frac{|\overline{in}(\alpha_1,\ f)|}{|\overline{in}(\alpha_2,\ f)|}\cdot\left|S\left(\overline{in}\left(A\setminus\lambda\left(A_f,\ i,\ s_1\right),\ f\right)\right)\right|.$$

Следовательно

$$\left|S\left(\tilde{A}(j,\ s_2,\ f)\right)\right|=\frac{|in(\alpha_2,\ f)|}{|\overline{in}(\alpha_2,\ f)|}\cdot\frac{|\overline{in}(\alpha_1,\ f)|}{|in(\alpha_1,\ f)|}\cdot\left|S\left(\tilde{A}(i,\ s_1,\ f)\right)\right|.$$

Рассмотрим теперь другую ситуацию. Пусть заданы $X\subseteq A_f,\ Y\subseteq A_f,\ i<|\ A_f|,\ j<|\ A_f|,\ s_1\in\left\{1,\ ...,\ \binom{|A_f|}{i}\right\}$, $s_2\in\left\{1,\ ...,\ \binom{|A_f|}{j}\right\}$ такие, что $\lambda(A_f,\ i,\ s_1)\setminus\lambda(A_f,\ j,\ s_2)=X\wedge\lambda(A_f,\ j,\ s_2)\setminus\lambda(A_f,\ i,\ s_1)=Y$. Предположим также, что выполняется равенство

$$\left|S\left(\tilde{A}(j,\, s_2,\, f)\right)\right| = \prod_{\alpha\in Y}\frac{\left|in(\alpha,\, f)\right|}{\left|\overline{in}(\alpha,\, f)\right|}\cdot\prod_{\alpha\in X}\frac{\left|\overline{in}(\alpha,\, f)\right|}{\left|in(\alpha,\, f)\right|}\cdot\left|S\left(\tilde{A}(i, s_1, f)\right)\right|.$$

Зафиксируем некоторый класс эквивалентности $\beta\in A_f$, $\beta\neq\varnothing$ такой, что: $\beta\notin X\wedge\beta\notin Y$. В силу определения отображений S и $\tilde{A}$, имеет место факт

$$\left|S\left(\tilde{A}(j,\, s_2,\, f)\right)\right| = \left|in\left(\lambda\left(A_f,\, j,\, s_2\right),\, f\right)\right|\cdot\left|\overline{in}\left(A\backslash\lambda\left(A_f,\, j,\, s_2\right),\, f\right)\right|.$$

Кроме того,

$$\left|S\left(in\left(\lambda\left(A_f,\, j,\, s_2\right)\cup\{\beta\},\, f\right)\right)\right|\cdot\left|S\left(\overline{in}\left(A\backslash\left(\lambda\left(A_f,\, j,\, s_2\right)\cup\{\beta\}\right),\, f\right)\right)\right| =$$

$$= \frac{\left|in(\beta,\, f)\right|}{\left|\overline{in}(\beta,\, f)\right|}\cdot\left|S\left(in\left(\lambda\left(A_f,\, j,\, s_2\right),\, f\right)\right)\right|\cdot\left|S\left(\overline{in}\left(A\backslash\lambda\left(A_f,\, j,\, s_2\right), f\right)\right)\right| =$$

$$= \frac{\left|in(\beta,\, f)\right|}{\left|\overline{in}(\beta,\, f)\right|}\cdot\left|S\left(\tilde{A}(j,\, s_2,\, f)\right)\right| = \frac{\left|in(\beta,\, f)\right|}{\left|\overline{in}(\beta,\, f)\right|}\cdot\prod_{\alpha\in Y}\frac{\left|in(\alpha,\, f)\right|}{\left|\overline{in}(\alpha,\, f)\right|}\cdot\prod_{\alpha\in X}\frac{\left|\overline{in}(\alpha,\, f)\right|}{\left|in(\alpha,\, f)\right|}\cdot\left|S\left(\tilde{A}(i,\, s_1,\, f)\right)\right|.$$

Очевидно, для $\lambda\left(A_f, j, s_2\right)\cup\{\beta\}$ есть соответствующая выборка классов эквивалентности, т. е. $\exists k,\ s_3$: $\lambda\left(A_f,\, j,\, s_3\right) = \lambda\left(A_f,\, j,\, s_2\right)\cup\{\beta\}$. Таким образом, индуктивное предположение доказано в силу произвольности X, Y и β.

Рассмотрим, как зависят возрастание или убывание общего числа слоев от числа элементов в выборках, т. е. как $W(A, f, i)$ зависит от изменения i.

По определению $W(A,\, f,\, i) = \sum\limits_{s=1}^{\binom{|A_f|}{i}}\left|S\left(\tilde{A}(i,\, s,\, f)\right)\right|$.

Биномиальный коэффициент $\binom{|A_f|}{i}$ возрастает при $i\in\left[1, \left[\frac{|A_f|}{2}\right]\right]$ и убывает при $i\in\left[\left[\frac{|A_f|+1}{2}\right], |A_f|\right]$. Если $|A_f|$ — четное, то $\left[\frac{|A_f|}{2}\right] = \left[\left(|A_f|+1\right)/2\right]$. Если же $|A_f|$ — нечетное, то равны соответствующие биномиальные коэффициенты. То есть $\forall i = \overline{\left[|A_f|/2\right]+1,\, |A_f|} : \binom{|A_f|}{i} \leqslant \binom{|A_f|}{i-1}$.

В силу утверждения 2, для произвольных допустимых i, k, s выполняется

$$\left|S\left(\tilde{A}(i,\, k,\, f)\right)\right| = \frac{C\left(A_f,\, i,\, i-1,\, k,\, s\right)}{C\left(A_f, i-1,\, i,\, s,\, k\right)}\left|S\left(\tilde{A}(i-1,\, s,\, f)\right)\right|.$$

Если $\lambda(A_f,\ i-1,\ s)\subset\lambda(A_f,\ i,\ k)$, то $C(A_f,\ i-1,\ i,\ s,\ k) = 1$, а $C\left(A_f, i, i-1, k, s\right) = \frac{\left|in(\alpha,\, f)\right|}{\left|\overline{in}(\alpha,\, f)\right|}$, где $\{\alpha\} = \lambda\left(A_f, i, k\right)\backslash\lambda\left(A_f, i-1, s\right)$. Если при этом число элементов класса эквивалентности α, содержащих в своей левой части атом f, меньше, чем число элементов, не содержащих заданный атом, то будет выполняться соотношение $\left|S\left(\tilde{A}(i,\, k,\, f)\right)\right| < \left|S\left(\tilde{A}(i-1,\, s,\, f)\right)\right|$. Такая ситуация может встречаться довольно часто при больших значениях $|R|$ и больших мощностях классов эквивалентности $|\alpha|\in A = R/\sim$. В интеллектуальных системах такая ситуация соответствует базам знаний с большим числом правил.

Исходя из полученных соотношений, можно сделать следующие эвристические предположения:

1) при $\left[|A_f|/2\right]+1\leqslant i\leqslant|A_f|$ часто будет выполняться соотношение $W(A,\, f,\, i)\leqslant W(A,\, f,\, i-1)$;

2) при $2\leqslant i\leqslant|A_f|$ часто будут выполняться соотношения

$$\binom{|A_f|}{i-1} \max_{s=\binom{|A_f|}{i}} \left|S\left(\tilde{A}(i, s, f)\right)\right| \leqslant W(A, f, i-1);$$

$$W(A, f, i) \leqslant \binom{|A_f|}{i} \min_{s=\binom{|A_f|}{i-1}} \left|S\left(\tilde{A}(i-1, s, f)\right)\right|.$$

Далее рассмотрим зависимость числа элементов в множестве $\theta(\alpha, \varepsilon \cup \{e\})$ от структуры класса эквивалентности α, подмножества фундаментальных циклов $\varepsilon \subseteq E$ и некоторого фундаментального цикла $e \in E$, $e \notin \varepsilon$. Имеем

$$\begin{cases} \alpha \cap \dot{\varepsilon}(\{e\}) = \varnothing \\ \alpha \cap \dot{\varepsilon}(\varepsilon) = \varnothing \end{cases} \rightarrow \begin{cases} \theta(\alpha, \{e\}) = |\alpha| \\ \theta(\alpha, \varepsilon) = |\alpha| \end{cases} \rightarrow |\theta(\alpha, \varepsilon)| = |\theta(\alpha, \varepsilon \cup \{e\})|;$$

$$\begin{cases} \alpha \cap \dot{\varepsilon}(\{e\}) = \varnothing \\ \alpha \cap \dot{\varepsilon}(\varepsilon) \neq \varnothing \end{cases} \rightarrow \begin{cases} \theta(\alpha, \{e\}) = |\alpha| \\ \theta(\alpha, \varepsilon) = |\alpha \cap \dot{\varepsilon}(\varepsilon)| \end{cases} \rightarrow |\theta(\alpha, \{e\} \cup \varepsilon)| = |\theta(\alpha, \varepsilon)|;$$

$$\begin{cases} \alpha \cap \dot{\varepsilon}(\{e\}) \neq \varnothing \\ \alpha \cap \dot{\varepsilon}(\varepsilon) = \varnothing \end{cases} \rightarrow \begin{cases} \theta(\alpha, \{e\}) = |\alpha \cap \dot{\varepsilon}(\{e\})| \\ \theta(\alpha, \varepsilon) = |\alpha| \end{cases} \rightarrow |\theta(\alpha, \{e\} \cup \varepsilon)| < |\theta(\alpha, \varepsilon)|;$$

$$\begin{cases} \alpha \cap \dot{\varepsilon}(\{e\}) \neq \varnothing \\ \alpha \cap \dot{\varepsilon}(\varepsilon) \neq \varnothing \\ \alpha \cap \dot{\varepsilon}(\{e\}) \subseteq \alpha \cap \dot{\varepsilon}(\varepsilon) \end{cases} \rightarrow \begin{cases} \theta(\alpha, \{e\}) = |\alpha \cap \dot{\varepsilon}(\{e\})| \\ \theta(\alpha, \varepsilon) = |\alpha \cap \dot{\varepsilon}(\varepsilon)| \end{cases} \rightarrow |\theta(\alpha, \{e\} \cup \varepsilon)| = |\theta(\alpha, \varepsilon)|;$$

$$\begin{cases} \alpha \cap \dot{\varepsilon}(\{e\}) \neq \varnothing \\ \alpha \cap \dot{\varepsilon}(\varepsilon) \neq \varnothing \\ \alpha \cap \dot{\varepsilon}(\{e\}) \not\subseteq \alpha \cap \dot{\varepsilon}(\varepsilon) \end{cases} \rightarrow \begin{cases} \theta(\alpha, \{e\}) = |\alpha \cap \dot{\varepsilon}(\{e\})| \\ \theta(\alpha, \varepsilon) = |\alpha \cap \dot{\varepsilon}(\varepsilon)| \end{cases} \rightarrow |\theta(\alpha, \{e\} \cup \varepsilon)| > |\theta(\alpha, \varepsilon)|.$$

Таким образом, при добавлении нового цикла $e \in E$ к подмножеству $\varepsilon \subseteq E$ значение $|S_{subsets}\left(\tilde{A}(i, s, f), \Theta, F\left(\tilde{A}(i, s, f), \Theta, \varepsilon\right)\right)$ — число слоев, которые можно построить по сужению $\Theta(A, \varepsilon)$, будет изменяться следующим образом:

- **убывать**, если $\forall \alpha \in A : \alpha \cap \dot{\varepsilon}(\varepsilon) = \varnothing$ и $\exists \alpha \in A : \alpha \cap \dot{\varepsilon}(\{e\}) \neq \varnothing$;
- **не возрастать**, если $\forall \alpha \in A : \alpha \cap \dot{\varepsilon}(\{e\}) = \varnothing \vee \alpha \cap \dot{\varepsilon}(\varepsilon) = \varnothing \vee \alpha \cap \dot{\varepsilon}(\{e\}) \subseteq \alpha \cap \dot{\varepsilon}(\varepsilon)$;
- **не убывать**, если $\forall \alpha \in A : \alpha \cap \dot{\varepsilon}(\{e\}) = \varnothing \vee \alpha \cap \dot{\varepsilon}(\varepsilon) \neq \varnothing$.

Исходя из полученных соотношений можно сделать эвристическое предположение, что наибольшие значения $|S_{all}(A, \Theta, \varepsilon)|$ достигаются при $|\varepsilon| \in \{1, |E|\}$. Таким образом, для приближенного подсчета $|S_{subsets}(A, \Theta, \varepsilon)|$ можно просуммировать некоторое число начальных и конечных компонент формулы, а приближенное значение суммы внутренних компонент найти, например, путем интегрирования методом Монте-Карло [11]. Выборку для этого метода можно строить по подпоследовательностям циклов, порождающих неубывающую последовательность числа слоев.

Кластеризация множества слоев

Ниже предложен способ эвристической кластеризации виртуального расслоения бинарного отношения, на основании которого будет построен алгоритм стохастического поиска прообразов.

Для бинарного отношения R построим граф $G = \{\nu\}$:

$$\nu \in G,\ \nu = (f, g),\ f, g \in F \Leftrightarrow \exists(\tau, g) \in R : f \in \tau.$$

Далее, используя один из известных математических методов кластеризации графов [12], разобьем вершины графа G на m кластеров: $G_i = \{f \in F\},\ i = \overline{1, m}$.

Для каждого атома решетки вычислим приближенную оценку отношения числа слоев без циклов к общему числу слоев:

$$\omega(f) \approx \frac{W_{noloops}(A, f)}{W(A, f)},\ f \in F,$$

а также медиану соответствующих степеней истинности:

$$\mathrm{med}_{\mu}(f) = \begin{cases} \mathrm{median}\left(\{M_R(r) | r = (\tau, f) \in R\}\right), & \text{если } f \in F_{notinit} \\ \mathrm{median}\left(M_{init}(f)\right), & \text{иначе} \end{cases}.$$

Таким образом, для каждого атома $f \in F$ можно построить вектор из трех числовых значений: $\nu = \{i, \omega(f), \mathrm{med}_\mu(f)\}$, где i — номер кластера графа G. Обозначим множество таких векторов как $\mathbf{V} = \{\nu_1, \ldots, \nu_{|F|}\}$.

Используя один из методов кластеризации [13, 14], выполним кластеризацию вектора $\mathbf{V}$ и обозначим полученные кластеры символами V_j, $j = \overline{1, n}$, где n — число кластеров. Обозначим $f_{i,k}$ атом, соответствующий k-му элементу кластера V_j. Для каждого кластера определим значение вероятности:

$$l_j = \sum_{k=1}^{|V_j|} \sqrt{\omega(f_{j,k}), \mathrm{med}_\mu(f_k)}, \quad p_j = \frac{l_j}{\sum_{k=1}^{n} l_k}.$$

На основе введенных обозначений можно построить алгоритм кластеризации и стохастического поиска прообразов в виртуальном расслоении бинарного отношения. Он принимает на вход четыре ограничения: C_t — максимальное время работы; C_i — максимальное число итераций; C_ρ — максимальное число найденных уникальных прообразов; C_W — максимальное число слоев в кластере для анализа. Возвращаемое значение алгоритма — множество найденных прообразов.

В ходе работы алгоритма подразумевается использование нескольких внешних функций:

- $\mathrm{rand}(X) = x \in X$ — генератор случайного элемента x множества $X = \{x_1, \ldots, x_m\}$ по равномерному распределению вероятностей;
- $\mathrm{rand}(X, P) = x \in X$ — генератор случайного элемента x множества $X = \{x_1, \ldots, x_m\}$ по распределению вероятностей $P = \{p_1, \ldots, p_m\}$;
- minPreImages(υ) —функция, выполняющая поиск прообразов на множестве слоев $S_{all}(A, \upsilon)$. Спецификация ее работы аналогична одноименной функции в методе LP-вывода [4].

Далее приведен псевдокод алгоритма стохастического поиска прообразов.

input: C_t, C_i, C_ρ, C_W
output: ρ
$i \leftarrow 0$, $t \leftarrow 0$, $\rho \leftarrow \varnothing$
while $t < C_t \wedge i < C_i \wedge |\rho| < C_\rho$ **do**
  $\upsilon \leftarrow \varnothing$
  while $\upsilon = \varnothing \vee |S_{all}(A, \upsilon)| > C_W$ **do**
    $j \leftarrow \mathrm{rand}(\{1, \ldots n\}, \{p_1, \ldots p_n\})$
    $\gamma \leftarrow \{f_{j,1}, \ldots, f_{j,|V_j|}\}$
    while $|\gamma| > 0 \vee |S_{all}(A, \upsilon)| > C_W$ **do**
      $f \leftarrow \mathrm{rand}(\gamma)$
      $\gamma \leftarrow \gamma \backslash \{f\}$
      $\upsilon \leftarrow \upsilon \cup \{f\}$
    end
  end
  $\rho \leftarrow \rho \cup \mathrm{minPreImages}(\upsilon)$
  $t \leftarrow \Delta t$
  $i \leftarrow i + 1$
end.

Заключение

Рассмотрена математическая формулировка задачи нечеткого LP-вывода. Исследованы свойства монотонности функции оценки числа слоев без циклов. Построен алгоритм кластеризации и стохастического поиска прообразов в виртуальном расслоении бинарного отношения. Полученные результаты могут быть применены для ускорения обратного вывода в интеллектуальных системах продукционного типа. Они могут быть распространены и на более сложные логические системы в информатике [15].

Список литературы

1. **Джарратано Д., Райли Г.** Экспертные системы: принципы разработки и программирование: пер. с англ. М.: Вильямс, 2007. 1152 с.
2. **Махортов С. Д., Шмарин А. Н.** Оптимизация метода LP-вывода// Нейрокомпьютеры. Разработка, применение. 2013. № 9. С. 59—63.
3. **Махортов С. Д**. Основанный на решетках подход к исследованию и оптимизации множества правил условной системы переписывания термов // Интеллектуальные системы. Теория и приложения. 2009. Т. 13. С. 51—68.
4. **Махортов С. Д.** Интегрированная среда логического программирования LPExpert // Информационные технологии. 2009. № 12. С. 65—66.
5. **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.
6. **Шмарин А. Н., Махортов С. Д.** О приближенных оценках количества слоев без циклов в задаче нечеткого LP-вывода // Нейрокомпьютеры. Разработка, применение. 2015. № 12. С. 44—51.
7. **Салий В. Н., Богомолов В. А.** Алгебраические основы теории дискретных систем. М.: Физматлит, 1997. 368 с.
8. **Пегат А.** Нечеткое моделирование и управление. М.: Бином. Лаборатория знаний, 2013. 798 с.
9. **Махортов С. Д.** LP-структуры для обоснования и автоматизации рефакторинга в объектно-ориентированном программировании // Программная инженерия. 2010. № 2. С. 15—21.
10. **Шмарин А. Н.** О реализации приближения числа слоев без циклов в задаче нечеткого LP-вывода // Программная инженерия. 2016. № 7. С. 330—336.
11. **Ермаков С. М.** Метод Монте-Карло в вычислительной математике. СПб.: Невский Диалект, 2009. 192 с.
12. **Schaeffer S. E.** Graph clustering // Computer Science Review. 2007. Vol. I. P. 27—64.
13. **Kanunfo T., Mount D. M., Netanyahu N. S.** An Efficient k-Means Clustering Algorithm: Analysis and Implementation // IEEE Transactions on pattern analysis and machine intelligence. 2002. Vol. 24, No. 7. P. 881—892.
14. **Patwary M. M., Palsetia D. A.** New Scalable Parallel DBSCAN Algorithm Using the Disjoint-Set Data Structure // IEEE. 2012. Vol. 12. P. 10—16.
15. **Чечкин А. В.** Нейрокомпьютерная парадигма информатики // Нейрокомпьютеры: разработка, применение. 2011. № 7. С. 3—9.

Clustering the Set of Layers in the Fuzzy LP-Inference Problem

A. N. Shmarin, e-mail: tim-shr@mail.ru, Voronezh State University, Voronezh, 394006, Russian Federation

Corresponding author:

Shmarin Artem N., Postgraduate Student, Voronezh State University, Voronezh, 394006, Russian Federation, E-mail: tim-shr@mail.ru

Received on December 25, 2016
Accepted on January 11, 2017

The increase in the volumes and complexity of processed information makes the use of artificial intelligence systems more and more actual. In such systems, one of the most widespread knowledge representation models is the production model, and search algorithms are based on the inference engine. In practice, the values measurement of the features of the classified objects from some data domain is a time-consuming operation for many problems. These tasks appear, for example, when the robot explores the surface of another planet. If the goal of the robot is to get to some rock, but the future route is observed only partially, then the robot should try to make the best decision on how to achieve the goal, taking into account limitation of the resources available to additional exploration of terrain. Another example, in which the cost of obtaining new information may be high, is the commercial medicine. To minimize costs, it is necessary to find the acceptable treatment method using the minimum number of analyses performed in minimal time. The delay in providing treatment, caused by the additional analyses, leads to increased costs. Moreover, the condition of a patient who was not provided timely assistance, can significantly deteriorate. Also, the cost of additional research is essential in the field of mineral exploration. It can turn out that a more cost-effective solution is to begin drilling, if confidence of the success is 95 %, than to spend the considerable resources to achieve the 98 % confidence.

The complexity of the operations of data acquisition necessary for decision-making, leads to the problem of minimizing the number of requests for information about values of the features of the classified object during inference. This problem is NP-hard. To achieve global minimization there is the general method of LP-inference with exponential computational complexity relative to the number of atomic facts in the knowledge base. However, some of its heuristic modifications have polynomial complexity. The goal of the provided research is development of the approximating method to better minimize the number of feature values requests performed in the inference.

Earlier, the questions of implementation of computing approximate estimates of the number of layers without cycles in the fuzzy LP-inference task were considered. This paper presents the mathematical statement of the fuzzy LP-inference task, researches the monotonicity properties of the function evaluating the number of layers without cycles, and presents the clustering algorithm of the set of layers — singleton samples from equivalence classes of the quotient set, that obtained by partitioning of the binary relation by the unique right parts of pairs. This algorithm is designed for the iterative analysis of such subsets of productions, which most significantly influence to the relevancy. The presented approach can be used to accelerate the reverse inference in production type systems of artificial intelligence.

Keywords: *LP-inference, binary relation, quotient set, clustering, stochastic algorithm, machine learning, algorithms*

For citation:

Shmarin A. N. Clustering the Set of Layers in the Fuzzy LP-Inference Problem, *Programmnaya Ingeneria,* 2017, vol. 8, no. 4, pp. 177—185.

DOI: 10.17587/prin.8.177-185

References

1. **Dzharratano D., Rajli G.** *Jekspertnye sistemy: principy razrabotki i programmirovanie* (Expert systems: design principles and programming), Moscow, Wil'ams, 2007, 1152 p. (in Russian).

2. **Makhortov S. D., Shmarin A. N.** Optimizacija metoda LP-vyvoda (Optimizing LP-inference method), *Nejrokomp'jutery. Razrabotka, primenenie*, 2013, no 9, pp. 59—63 (in Russian).

3. **Makhortov S. D.** Osnovannyj na reshetkah podhod k issledovaniju i optimizacii mnozhestva pravil uslovnoj sistemy perepisyvanija termov (Research and optimization of a set of rules in a conditional term rewriting system using approach based on the lattices), *Intellektual'nye sistemy. Teorija i prilozhenija*, 2009, Vol. 13, pp. 51—68 (in Russian).

4. **Makhortov S. D.** Integrirovannaja sreda logicheskogo programmirovanija LPExpert (LPExpert: Integrated development environment for logical programming), *Informacionnye tehnologii*, 2009, no. 12, pp. 65—66 (in Russian).

5. **Bolotova S. Ju., Makhortov S. D.** Algoritmy relevantnogo obratnogo vyvoda, osnovannye na reshenii produkcionno-logicheskih uravnenij (Algorithms of the relevant backward inference on production-logic equations solving), *Iskusstvennyj intellekt i prinjatie reshenij*, 2011, no. 2, pp. 40—50 (in Russian).

6. **Makhortov S. D., Shmarin A. N.** O priblizhennyh ocenkah kolichestva sloev bez ciklov v zadache nechetkogo LP-vyvoda (About approximate estimates of the number of layers without cycles in the fuzzy LP-inference), *Nejrokomp'jutery. Razrabotka, primenenie*, 2015, no. 12, pp. 44—51 (in Russian).

7. **Salij V. N., Bogomolov V. A.** *Algebraicheskie osnovy teorii diskretnyh sistem* (Algebraic theory of discrete systems), Moscow, Fizmatlit, 1997, 368 p. (in Russian).

8. **Pegat A.** *Nechetkoe modelirovanie i upravlenie* (Fuzzy Modeling and Controling), Moscow, Binom, Laboratorija znanij, 2013, 798 p. (in Russian).

9. **Makhortov S. D.** LP-struktury dlja obosnovanija i avtomatizacii refaktoringa v ob#ektno-orientirovannom programmirovanii (LP-structure for validation and automation of the refactoring in object-oriented programming), *Programmnaya Ingeneria*, 2010, no 2, pp. 15—21 (in Russian).

10. **Shmarin A. N.** O realizacii priblizhenija chisla sloev bez ciklov v zadache nechetkogo LP-vyvoda (About Implementation of Approximation of Layers without Cycles in the Fuzzy LP-Inference Problem), *Programmnaya Ingeneria*, 2016, vol. 7, no. 7, pp. 330—336 (in Russian).

11. **Ermakov S. M.** *Metod Monte-Karlo v vychislitel'noj matematike* (Monte Carlo method in computational mathematics), St. Petersburg, Nevskij Dialekt, 2009, 192 p. (in Russian).

12. **Schaeffer S. E.** Graph clustering, *Computer Science Review,* 2007, vol. I, pp. 27—64.

13. **Kanunfo T., Mount D. M., Netanyahu N. S.** An Efficient k-Means Clustering Algorithm: Analysis and Implementation, *IEEE Transactions on pattern analysis and machine intelligence,* 2002, vol. 24, no. 7, pp. 881—892.

14. **Patwary M. M., Palsetia D. A.** New Scalable Parallel DBSCAN Algorithm Using the Disjoint-Set Data Structure, *IEEE,* 2012, vol. 12, pp. 10—16.

15. **Chechkin A. V.** Nejrokomp'juternaja paradigma informatiki (Neurocomputing paradigm of informatics), *Nejrokomp'jutery: Razrabotka, Primenenie*, 2011, no. 7, pp. 3—9 (in Russian).

УДК 004.75 + 004.41 DOI: 10.17587/prin.8.186-192

А. С. Шундеев, канд. физ.-мат. наук, вед. науч. сотр., e-mail: alex.shundeev@gmail.com,
НИИ механики МГУ имени М. В. Ломоносова

Об одной задаче распознавания автоматных языков

Конечные автоматы и автоматные языки являются эффективным средством описания и моделирования программных систем. Для исследования свойств автоматных языков могут быть применены подходы, которые традиционно относятся к области анализа данных.

В работе рассмотрено решение задачи классификации слов над заданным алфавитом с использованием методов машинного обучения с учителем. В этой задаче рассмотрены два класса слов, один из которых представляется автоматным языком, а второй класс является дополнением этого языка. Однако существенным ограничением является то, что конечный автомат, задающий этот язык, считается неизвестным. Известны только эталонные представители каждого из классов.

Ключевые слова: *детерминированный конечный автомат, автоматный язык, задача классификации, задача распознавания, машинное обучение*

Введение

Задача *классификации*, возникающая в области анализа данных, может быть сформулирована следующим образом. Рассматривается некоторое непустое множество объектов X, а также предполагается, что существует некоторая функция $f: X \to Y$, областью значений которой является конечное множество Y. Таким образом, множество объектов X может быть представлено в виде объединения попарно непересекающихся множеств

$$X = \bigcup_{y \in Y} C_y, \quad C_y = \{x \in X \mid f(x) = y\}.$$

Множества C_y называют классами, и каждый объект $x \in X$ принадлежит ровно одному классу. При этом функцию f естественно называть классификатором. Трудности решения этой задачи обусловлены тем, что в явном виде определение классификатора f может быть недоступно или его использование связано с большими вычислительными затратами. Поэтому появляется необходимость строить и использовать некоторое его приближение $\tilde{f}$. Соответственно, приходится оперировать не целевыми классами объектов C_y, а их некоторыми аппроксимациями $\tilde{C}_y$.

Существуют общие подходы к решению поставленной задачи. Предполагается, что в какой-то степени известна структура объектов множества X. Например, в статистических методах анализа данных [1] объекты отождествляют с наборами качественных и количественных параметров, на основе совокупного анализа которых объект с некоторой долей достоверности может быть отнесен к тому или иному классу. Учитывают также возможные ограничения, которым должен удовлетворять классификатор f.

Часто используют так называемые обучающие выборки. В этом случае говорят, что происходит обучение с учителем. В противном случае говорят, что происходит обучение без учителя, а исходную задачу часто называют задачей *кластеризации*. В дальнейшем будем рассматривать только методы обучения с учителем.

До начала построения приближения $\tilde{f}$ к классификатору f из каждого класса C_y выбирают некоторое конечное множество "эталонных" представителей этого класса $E_y \subseteq C_y$. Совокупность множеств $E_y (y \in Y)$ образует *обучающую выборку*. При построении приближения классификатора $\tilde{f}$ учитывается структура объектов обучающей выборки, а также — возможные ограничения, которые накладываются на классификатор f.

Рассмотрим частный вариант задачи классификации. Предположим, что $Y = \{+, -\}$. В этом случае задача классификации может быть проинтерпретирована как задача распознавания принадлежности объекта некоторому целевому множеству $L \subseteq X$. Каждый объект $x \in X$ должен быть отнесен либо к множеству $L = C_+$, либо к дополнению этого множества $\bar{L} = C_-$.

В настоящей работе будет рассмотрен этот частный вариант задачи классификации. В качестве множества объектов X будет выступать множество всех слов над некоторым алфавитом Σ. В качестве классов объектов будут выступать некоторый автоматный (регулярный) язык [2] L и его дополнение $\bar{L}$.

Содержание статьи построено следующим образом. В разд. 1 приведены основные понятия, относящиеся к детерминированным конечным автоматам (ДКА). В разд. 2 введено понятие аппроксимирующего автомата. Детерминированный конечный автомат полностью задает автоматный язык, а также допол-

нение этого языка. В контексте решаемой задачи ДКА представляет собой классификатор, который неизвестен, а аппроксимирующий автомат представляет собой приближение к этому классификатору. В разд. 3 введены и исследованы понятия слабой и сильной корректности приближения классификатора. В разд. 4 введено и исследовано понятие минимальности аппроксимирующего автомата. В разд. 5 обсуждено решение задачи классификации. Заключение содержит итоговые выводы.

1. Детерминированный конечный автомат

Непустое конечное множество будем называть *алфавитом*. Элементы алфавита будем называть *буквами* или *символами*. *Словом* над алфавитом Σ называется конечная последовательность букв Σ. *Длиной* слова называется число содержащихся в нем букв. Множество всех таких слов над алфавитом Σ будем обозначать Σ^*, а множество всех слов ненулевой длины будем обозначать Σ^+. Языком над алфавитом Σ будем называть подмножество в Σ^*.

При работе с языками выделяют две основные задачи. Первая задача состоит в описании или определении языка. Вторая задача состоит в распознавании языка. Распознавание заключается в проверке принадлежности произвольного слова $\omega \in \Sigma^*$ заданному языку $L \subseteq \Sigma^*$. Одной из самых эффективных моделей вычислительного устройства, позволяющего решать обозначенные задачи, является ДКА. Языки, описываемые и распознаваемые ДКА, образуют класс *автоматных* (*регулярных*) языков.

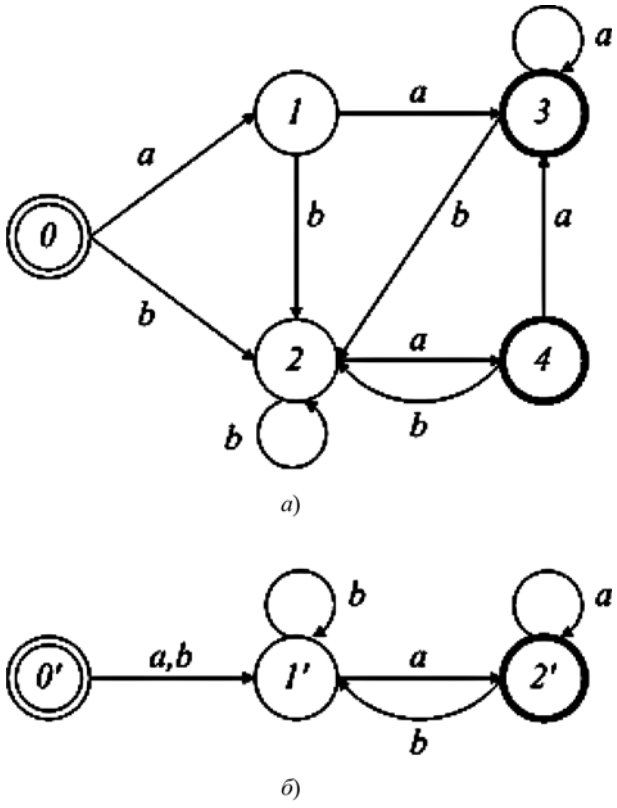
Определение. *Детерминированным конечным автоматом* называется пятерка $\mathcal{A} = (\Sigma, Q, \delta, s, F)$, где

- Σ — входной алфавит;
- Q — непустое конечное множество *состояний*;
- $\delta : Q \times \Sigma \to Q$ — функция *переходов* (возможно частично-определенная);
- $s \in Q$ — выделенное *начальное* состояние;
- $F \subseteq Q$ — множество *заключительных* состояний.

Распространенным способом задания ДКА является использование диаграммы переходов. Диаграммой переходов называется ориентированный граф. Вершинами такого графа являются состояния автомата. Из вершины $q_1 \in Q$ в вершину $q_2 \in Q$ следует дуга $q_1 \xrightarrow{a} q_2$, помеченная буквой $a \in \Sigma$ тогда и только тогда, когда $\delta(q_1, a) = q_2$. Подобные дуги будем называть переходами. В дальнейшем будем отождествлять функцию переходов ДКА и множество помеченных дуг его диаграммы переходов.

Графически вершины диаграммы переходов изображают кружочками. При этом начальное состояние изображается двойным кружочком, а заключительные состояния — кружочками с жирной границей.

На рис. 1 изображены диаграммы переходов двух ДКА $\mathcal{A}_1$ (рис. 1, *а*) и $\mathcal{A}_2$ (рис. 1, *б*). В обоих случаях входным является двухбуквенный алфавит $\{a, b\}$. Первый ДКА $\mathcal{A}_1$ имеет множество состояний {0, 1, 2, 3, 4}. Начальным является состояние 0, а заключительными состояниями являются состояния 3 и 4.



а)

б)

Рис. 1. Примеры диаграмм переходов ДКА

Этот ДКА имеет десять переходов. Второй ДКА $\mathcal{A}_2$ имеет множество состояний $\{0', 1', 2'\}$. Начальным является состояние $0'$, а единственным заключительным состоянием является состояние $2'$. Этот ДКА имеет шесть переходов.

Последовательность переходов ДКА $\mathcal{A} = (\Sigma, Q, \delta, s, F)$ вида

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \ldots \xrightarrow{a_n} q_n$$

будем называть *вычислением* из состояния $q_0 \in Q$ в состояние $q_n \in Q$. При этом будем считать, что это вычисление *прочитывает* слово $\omega = a_1 a_2 \ldots a_n \in \Sigma^+$. Важным является следующее свойство ДКА. Если существует вычисление, начинающееся в заданном состоянии q_0 и прочитывающее заданное слово ω, то оно единственно. Поэтому для вычисления корректно использовать обозначение $q_0 \xrightarrow{\omega}_* q_n$.

Слово $\omega \in \Sigma^+$ *распознается* ДКА $\mathcal{A}$, если существует вычисление вида $s \xrightarrow{\omega}_* f$, где $f \in F$. Подобное вычисление начинается в начальном состоянии и завершается в некотором заключительном состоянии. Будем считать, что язык

$$L(\mathcal{A}) = \left\{ \omega \in \Sigma^+ \,\middle|\, s \xrightarrow{\omega}_* f,\ f \in F \right\}$$

распознается ДКА $\mathcal{A}$.

Рассмотрим примеры вычислений. Детерминированный конечный автомат $\mathcal{A}_1$ распознает слово *bbbaa*

и не распознает слово *bbbab*. Для первого слова соответствующее вычисление, начинающееся в начальном состоянии, имеет вид

$$0 \xrightarrow{b} 2 \xrightarrow{b} 2 \xrightarrow{b} 2 \xrightarrow{a} 4 \xrightarrow{a} 3.$$

Последнее состояние 3 является заключительным, поэтому слово распознается. Для второго слова соответствующее вычисление, начинающееся в начальном состоянии, имеет вид

$$0 \xrightarrow{b} 2 \xrightarrow{b} 2 \xrightarrow{b} 2 \xrightarrow{a} 4 \xrightarrow{b} 2.$$

Последнее состояние 2 не является заключительным, поэтому слово не распознается.

Два ДКА называют *эквивалентными*, если они распознают один язык. Например, $\mathcal{A}_1$ и $\mathcal{A}_2$ являются эквивалентными. Среди эквивалентных ДКА существует единственный ДКА (с точностью до переименования состояний) с наименьшим числом состояний, который называется *минимальным*. Детерминированный конечный автомат $\mathcal{A}_2$ является минимальным.

2. Аппроксимирующий автомат

В подлежащей решению задаче классификации ДКА выступает в качестве неизвестного классификатора. В качестве приближения к классификатору будем использовать аппроксимирующий автомат.

Определение. *Аппроксимирующим автоматом* называется шестерка $\mathcal{B} = (\Sigma, Q, \delta, s, F^+, F^-)$, где

- Σ — входной алфавит;
- Q — непустое конечное множество *состояний*;
- $\delta : Q \times \Sigma \to Q$ — функция *переходов* (возможно частично-определенная);
- $s \in Q$ — выделенное *начальное* состояние;
- $F^+ \subseteq Q$ — множество *положительных заключительных* состояний;
- $F^- \subseteq Q$ — множество *отрицательных заключительных* состояний.

Для задания аппроксимирующих автоматов также будем использовать диаграммы переходов (например, рис. 2, *в*). На подобной диаграмме положительное заключительное состояние будем дополнительно помечать символом "+", а отрицательное заключительное состояние — символом "–".

Как и в случае ДКА вводится понятие вычисления. Будем говорить, что слово $\omega \in \Sigma^+$ *положительно* (*отрицательно*) *распознается* аппроксимирующим автоматом $\mathcal{B}$, если существует вычисление вида $s \xrightarrow{\omega}_* f$, где $f \in F^+ (f \in F^-)$. Соответственно, можно ввести два языка. Первый язык состоит из всех положительно распознаваемых слов:

$$L^+(\mathcal{B}) = \left\{ \omega \in \Sigma^+ \,\middle|\, s \xrightarrow{\omega}_* f,\ f \in F^+ \right\},$$

второй язык состоит из всех отрицательно распознаваемых слов:

$$L^-(\mathcal{B}) = \left\{ \omega \in \Sigma^+ \,\middle|\, s \xrightarrow{\omega}_* f,\ f \in F^- \right\}.$$

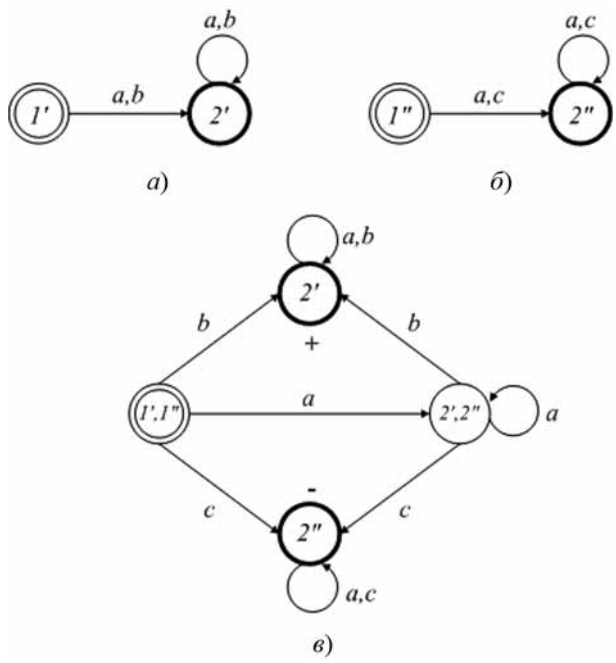


Рис. 2. Иллюстрация алгоритма разделения

Определение. Два аппроксимирующих автомата $\mathcal{B}$ и $\mathcal{B}'$ будем называть *эквивалентными* и использовать обозначение $\mathcal{B} \sim \mathcal{B}'$, если $L^+(\mathcal{B}) = L^+(\mathcal{B}')$ и $L^-(\mathcal{B}) = L^-(\mathcal{B}')$.

Определение. Аппроксимирующий автомат $\mathcal{B}$ будем называть *минимальным*, если он имеет наименьшее число состояний из всех эквивалентных ему автоматов.

3. Слабая и сильная корректность

В рассматриваемой задаче классификации используется обучающая выборка E_+, $E_- \subset \Sigma^*$, содержащая, по мнению учителя, эталонных представителей каждого из двух классов. На первом этапе построения классификатора анализируется структура элементов обучающей выборки и строится приближение классификатора $E_+ \subset L_+$, $E_- \subset L_-$. Построение этого начального приближения будет обсуждено позже. Возникает вопрос об оценке качества этого приближения классификатора.

Определение. Будем считать, что приближение классификатора L_+, L_- с обучающей выборкой E_+, E_- обладает *слабой корректностью*, если $E_+ \subseteq L_+$, $E_- \subseteq L_-$, $L_+ \cap E_- = \varnothing$ и $L_- \cap E_+ = \varnothing$.

Слабая корректность является минимальным требованием, которое необходимо предъявлять приближению классификатора.

Определение. Будем считать, что приближение классификатора L_+, L_- с обучающей выборкой E_+, E_- обладает *сильной корректностью*, если $E_+ \subseteq L_+$, $E_- \subseteq L_-$ и $L_+ \cap L_- = \varnothing$.

Очевидно, что из сильной корректности следует слабая корректность.

Далее, будет введено понятие разделяющего аппроксимирующего автомата, который позволяет трансформировать слабо корректное приближение классификатора к сильно корректному виду.

Определение. Пусть заданы два ДКА — $\mathcal{A}' = (\Sigma, Q', \delta', s', F')$ и $\mathcal{A}'' = (\Sigma, Q'', \delta'', s'', F'')$, у которых множества состояний не пересекаются $Q' \cap Q'' = \varnothing$. Аппроксимирующий автомат $\mathcal{B} = (\Sigma, Q, \delta, s, F^+, F^-)$ будем называть *разделяющим автоматом* для ДКА $\mathcal{A}'$ и $\mathcal{A}''$, если

- $Q = Q' \times Q'' \cup Q' \cup Q''$;
- $\delta = \overline{\delta} \cup \delta' \cup \delta''$;
- $s = (s', s'')$;
- $F^+ = F' \cup F' \times (Q'' \backslash F'')$;
- $F^- = F'' \cup (Q' \backslash F') \times F''$,

где функция $\overline{\delta}$ определяется следующим образом. Для произвольных $q_1' \in Q'$, $q_1'' \in Q''$, и $a \in \Sigma$ по определению положим:

- $\overline{\delta}((q_1', q_1''), a) = (q_2', q_2'')$, если $\delta'(q_1', a) = q_2'$ и $\delta''(q_1'', a) = q_2''$;
- $\overline{\delta}((q_1', q_1''), a) = q_2'$, если $\delta'(q_1', a) = q_2'$, а значение $\delta''(q_1'', a)$ не определено;
- $\overline{\delta}((q_1', q_1''), a) = q_2''$, если $\delta''(q_1'', a) = q_2''$, а значение $\delta'(q_1', a)$ не определено.

Состояния из множества $Q' \times Q''$ будем называть *парными*, а из множеств Q' и Q'' — *одинарными*.

На рис. 2 приведены диаграммы переходов двух ДКА и аппроксимирующего автомата, который является для них разделяющим. Первый ДКА $\mathcal{A}'$ (рис. 2, *а*) задает язык $\{a, b\}^+$. Второй ДКА $\mathcal{A}''$ (рис. 2, *б*) задает язык $\{a, c\}^+$. Эти два языка имеют непустое пересечение $\{a\}^+$. На рис. 2, *в* представлена диаграмма переходов аппроксимирующего автомата $\mathcal{B}$, который является разделяющим для ДКА $\mathcal{A}'$ и $\mathcal{A}''$. Как видно, этот аппроксимирующий автомат "вырезает" это пересечение. Положительно распознаваемый язык имеет вид $L^+(\mathcal{B}) = \{a, b\}^+ \backslash \{a\}^+$, отрицательно распознаваемый язык имеет вид $L^-(\mathcal{B}) = \{a, c\}^+ \backslash \{a\}^+$. Это свойство справедливо и для любого разделяющего автомата.

Утверждение. Пусть $\mathcal{B} = (\Sigma, Q, \delta, s, F^+, F^-)$ — разделяющий автомат для произвольных ДКА $\mathcal{A}' = (\Sigma, Q', \delta', s', F')$ и $\mathcal{A}'' = (\Sigma, Q'', \delta'', s'', F'')$ с непересекающимися множествами состояний. Тогда

$$L^+(\mathcal{B}) = L(\mathcal{A}') \backslash L(\mathcal{A}'') \text{ и } L^-(\mathcal{B}) = L(\mathcal{A}'') \backslash L(\mathcal{A}').$$

Проведем доказательство первого равенства. Второе равенство доказывается аналогично. Предварительно отметим, что любое вычисление разделяющего автомата, начинающееся в начальном состоянии, устроено следующим образом. В нем либо участвуют только парные состояния (тип 1), либо вначале находятся только парные состояния, а затем следуют только одинарные состояния (тип 2). Во втором случае все одинарные состояния принадлежат либо множеству Q', либо множеству Q''.

Проверим включение $L^+(\mathcal{B}) \subseteq L(\mathcal{A}') \backslash L(\mathcal{A}'')$. Рассмотрим произвольное слово $\omega = a_1 a_2 \ldots a_n \in L^+(\mathcal{B})$. Это слово положительно распознается $\mathcal{B}$, и этому распознаванию соответствует вычисление либо типа 1, либо типа 2. Последовательно рассмотрим эти два случая.

Предположим, что вычисление имеет тип 1:

$$(s', s'') \xrightarrow{a_1} (q_1', q_1'') \xrightarrow{a_2} \ldots \xrightarrow{a_n} (q_n', q_n''). \quad (1)$$

По определению функции δ у ДКА $\mathcal{A}'$ имеется вычисление $s' \xrightarrow{\omega}{}^* q_n'$, а у ДКА $\mathcal{A}''$ имеется вычисление $s'' \xrightarrow{\omega}{}^* q_n''$. Так как по предположению слово ω положительно распознается аппроксимирующим автоматом $\mathcal{B}$, то $(q_n', q_n'') \in F^+$. Следовательно, с одной стороны, $q_n' \in F'$ и $\omega \in L(\mathcal{A}')$, а с другой стороны, $q_n'' \notin F''$ и $\omega \notin L(\mathcal{A}'')$.

Предположим, что вычисление имеет тип 2:

$$(s', s'') \xrightarrow{a_1} \ldots (q_k', q_k'') \xrightarrow{a_{k+1}} q_{k+1}' \ldots \xrightarrow{a_n} q_n' \quad (2)$$

для некоторого числа $k < n$. При этом значение $\delta''(q_k'', a_{k+1})$ не определено. Это значит, что у ДКА $\mathcal{A}''$ не существует вычисления, начинающегося в его начальном состоянии и прочитывающего слово ω. Следовательно, $\omega \notin L(\mathcal{A}'')$. В то же время по определению функции δ у ДКА $\mathcal{A}'$ имеется вычисление $s' \xrightarrow{\omega}{}^* q_n'$. Так как по предположению слово ω положительно распознается аппроксимирующим автоматом $\mathcal{B}$, то $q_n' \in F^+$. Следовательно, $q_n' \in F'$ и $\omega \in L(\mathcal{A}')$.

Проверим включение $L(\mathcal{A}') \backslash L(\mathcal{A}'') \subseteq L^+(\mathcal{B})$. Рассмотрим произвольное слово $\omega = a_1 a_2 \ldots a_n \in L(\mathcal{A}') \backslash L(\mathcal{A}'')$. Так как $\omega \in L(\mathcal{A}')$, то у ДКА $\mathcal{A}'$ имеется вычисление вида

$$s' \xrightarrow{a_1} q_1' \xrightarrow{a_2} \ldots \xrightarrow{a_n} q_n',$$

где $q_n' \in F'$.

Так как $\omega \notin L(\mathcal{A}'')$, то возможны два случая.

В первом случае у ДКА $\mathcal{A}''$ имеется вычисление вида

$$s'' \xrightarrow{a_1} q_1'' \xrightarrow{a_2} \ldots \xrightarrow{a_n} q_n'',$$

при этом $q_n'' \notin F''$. Но тогда разделяющий автомат $\mathcal{B}$ положительно распознает слово ω с помощью вычисления вида (1). Значит $\omega \in L^+(\mathcal{B})$.

Во втором случае у ДКА $\mathcal{A}''$ существует вычисление только для собственного префикса слова ω. Это вычисление будет иметь вид

$$s'' \xrightarrow{a_1} q_1'' \ldots \xrightarrow{a_k} q_k'',$$

для некоторого числа $k < n$. При этом значение $\delta''(q''_k, a_{k+1})$ должно быть не определено. Это значит, что разделяющий автомат $\mathcal{B}$ положительно распознает слово ω с помощью вычисления вида (2). Следовательно $\omega \in L^+(\mathcal{B})$.

4. Минимизация аппроксимирующего автомата

Построение и доказательство единственности минимального аппроксимирующего автомата почти совпадает с аналогичными построениями для ДКА. Приведем основные этапы.

Прежде всего, отметим, что важную роль в приводимых ниже построениях играет свойство детерминированности, которое является отличительной чертой ДКА, и которое унаследовано от него аппроксимирующим автоматом. Если существует вычисление, начинающееся в заданном состоянии и прочитывающее заданное слово, то однозначно определяется как конечное состояние, так и последовательность всех промежуточных состояний.

Определение. Аппроксимирующий автомат называется *приведенным*, если каждое его состояние используется в вычислении, с помощью которого положительно или отрицательно распознается некоторое слово.

Определение. Пусть $\mathcal{B} = (\Sigma, Q, \delta, s, F^+, F^-)$ — аппроксимирующий автомат и $q \in Q$ — его произвольное состояние. Введем следующее обозначение $\mathcal{B}[q] = (\Sigma, Q, \delta, s, F^+, F^-)$.

Будем называть состояния $q_1 \in Q$ и $q_2 \in Q$ *эквивалентными* и писать $q \sim q'$, если $\mathcal{B}[q] \sim \mathcal{B}[q']$.

Утверждение. Пусть заданы два произвольных аппроксимирующих автомата $\mathcal{B} = (\Sigma, Q, \delta, s, F^+, F^-)$ и $\mathcal{B}' = (\Sigma, Q', \delta', s', F^{+'}, F^{-'})$. Если эти автоматы эквивалентны $\mathcal{B} \sim \mathcal{B}'$ и $\mathcal{B}$ является приведенным, тогда для любого состояния $q \in Q$ первого автомата найдется состояние $q' \in Q'$ второго автомата такое, что $\mathcal{B}[q] \sim \mathcal{B}'[q']$.

Так как аппроксимирующий автомат $\mathcal{B}$ является приведенным, то рассматриваемое состояние $q \in Q$ участвует в положительном или отрицательном распознавании некоторого слова $\omega \in L^+(\mathcal{B}) \cup L^-(\mathcal{B})$. Это вычисление может быть представлено в виде

$$s \xrightarrow{u}{}_* \, q \xrightarrow{v}{}_* \, f,$$

где $\omega = uv$, $f \in F^+ \cup F^-$.

Так как $\mathcal{B} \sim \mathcal{B}'$, то $\omega \in L^+(\mathcal{B}') \cup L^-(\mathcal{B}')$ и у аппроксимирующего автомата $\mathcal{B}'$ имеется вычисление вида

$$s' \xrightarrow{u}{}_* \, q' \xrightarrow{v}{}_* \, f',$$

где $q' \in Q'$, $f' \in F^{+'} \cup F^{-'}$. Покажем, что $\mathcal{B}[q] \sim \mathcal{B}'[q']$.

Проверим включение $L^+(\mathcal{B}[q]) \subseteq L^+(\mathcal{B}'[q'])$. Рассмотрим произвольное слово $x \in L^+(\mathcal{B}[q])$. Для этого слова у автомата $\mathcal{B}[q]$ имеется вычисление вида $q \xrightarrow{x}{}_* \, f$, для некоторого положительного заключительного состояния $f \in F^+$. Тогда у автомата имеется вычисление $s \xrightarrow{u}{}_* \, q \xrightarrow{x}{}_* \, f$. Следовательно, по определению $ux \in L^+(\mathcal{B})$.

В силу эквивалентности автоматов $\mathcal{B} \sim \mathcal{B}'$, имеет место включение $ux \in L^+(\mathcal{B}')$. Это значит, что у автомата $\mathcal{B}'$ имеется вычисление вида $s' \xrightarrow{ux}{}_* \, f$, для некоторого положительного заключительного состояния $f' \in F^{+'}$. Такое вычисление будет иметь следующую структуру: $s' \xrightarrow{u}{}_* \, q' \xrightarrow{x}{}_* \, f'$. Следовательно $q' \xrightarrow{x}{}_* \, f'$ и $x \in L^+(\mathcal{B}'[q'])$.

Аналогично проверяется включение $L^+(\mathcal{B}'[q']) \subseteq L^+(\mathcal{B}[q])$, и, следовательно, $L^+(\mathcal{B}[q]) = L^+(\mathcal{B}'[q'])$. Точно также устанавливается равенство $L^-(\mathcal{B}[q]) = L^-(\mathcal{B}'[q'])$.

Утверждение. Пусть $\mathcal{B} = (\Sigma, Q, \delta, s, F^+, F^-)$ — приведенный аппроксимирующий автомат, все состояния которого попарно неэквивалентны. Тогда автомат $\mathcal{B}$ является минимальным.

Допустим, что существует эквивалентный $\mathcal{B}$ аппроксимирующий автомат $\mathcal{B}' = (\Sigma, Q', \delta', s', F^{+'}, F^{-'})$ с меньшим числом состояний $|Q'| < |Q|$. В силу предыдущего утверждения для каждого состояния автомата $\mathcal{B}$ у автомата $\mathcal{B}'$ найдется эквивалентное состояние. Тогда, в силу принципа Дирихле, у автомата $\mathcal{B}$ существуют два различных эквивалентных состояния, что противоречит исходному предположению.

Теперь понятно, что для получения минимального аппроксимирующего автомата нужно взять приведенный автомат и "склеить" все его эквивалентные состояния. Процедура "склеивания" интуитивно понятна, поэтому описываться не будет. Очевидно, что получившийся минимальный автомат будет единственным с точностью до переименования состояний.

5. Процедура классификации

Опишем решение рассматриваемой задачи классификации. В качестве иллюстрации будем использовать примеры автоматов, приведенных на рис. 2. Как уже было отмечено ранее, в процессе обучения используется обучающая выборка.

Шаг 1. На этом шаге учитель формирует обучающую выборку, содержащую эталонных представителей каждого из двух классов. Напомним, что первый класс — это множество всех слов распознаваемого автоматного языка L, а второй класс — это дополнение этого языка $\overline{L}$.

Например, возьмем в качестве входного алфавита $\Sigma = \{a, b, c\}$, а в качестве обучающей выборки возьмем $E_+ = \{b, ab, aab, aba\}$ и $E_- = \{c, ac, cccaca\}$.

Шаг 2. На этом шаге должен быть проведен анализ структуры элементов обучающей выборки. Результатом этого анализа должно стать построение первого приближения классификатора. Возникает вопрос, как обобщить набор эталонных представи-

телей класса? Предлагаемый подход, на взгляд автора, интуитивно понятен, хотя возможно и требует дальнейшего теоретического обоснования.

Предположим, что требуется обобщить эталонных представителей E_+. Возьмем минимальный ДКА, который распознает только элементы из множества E_+. Пусть у этого ДКА число состояний равно N. Далее нужно рассмотреть все возможные минимальные автоматы с числом состояний меньше N, распознающие элементы множества E_+ и не распознающие элементы множества E_-.

В иллюстрирующем примере первым приближением классификатора стали языки, задаваемые соответственно ДКА $\mathcal{A}'$ (рис. 2, *а*) и $\mathcal{A}''$ (рис. 2, *б*). Таким образом, $L \approx L(\mathcal{A}') = \{a, b\}^+$ и $\bar{L} \approx L(\mathcal{A}'') = \{a, c\}^+$.

Шаг 3. От первого приближения классификатора можно требовать только выполнение свойства слабой корректности. Как уже было показано ранее, в рассматриваемом примере приближение классификатора в виде пары языков $L(\mathcal{A}')$ и $L(\mathcal{A}'')$ обладает свойством слабой корректности, но не обладает свойством сильной корректности. Их пересечение не пусто: $L(\mathcal{A}') \cap L(\mathcal{A}'') = \{a\}^+$.

На этом шаге с помощью построения разделяющего аппроксимирующего автомата $\mathcal{B}$ строится второе приближение классификатора. Приближением первого класса $L \approx L^+(\mathcal{B})$ выступает положительно распознаваемый автоматом $\mathcal{B}$ язык. Приближением второго класса $\bar{L} \approx L^-(\mathcal{B})$. Второе приближение классификатора уже будет обладать свойством сильной корректности.

В иллюстрирующем примере $L \approx \{a, b\}^+ \backslash \{a\}^+$ и $\bar{L} \approx \{a, c\}^+ \backslash \{a\}^+$.

Шаг 4. Второе приближение классификатора признается итоговым. Однако с целью повышения эффективности его использования аппроксимирующий автомат может быть минимизирован.

Заключение

В настоящей статье представлено решение задачи классификации на основе обучения с учителем. В этой задаче рассматривается два класса, один из которых представляется автоматным (регулярным) языком, а второй класс является дополнением этого языка. Однако существенным ограничением является то, что ДКА, задающий этот язык, считается неизвестным. Известны только эталонные представители каждого из классов.

Для решения поставленной задачи было введено новое понятие аппроксимирующего автомата, являющегося обобщением ДКА. Было введено понятие слабой и сильной корректности приближения классификатора. С помощью аппроксимирующего автомата предложена и обоснована процедура построения приближения классификатора, обладающего свойством сильной корректности. Решена задача построения минимального аппроксимирующего автомата.

Список литературы

1. **James G., Witten D., Hastie T., Tibshirani R.** An Introduction to Statistical Learning with Applications in R. Springer, 2013. 426 p.

2. **Замятин А. П., Шур А. М.** Языки, грамматики, распознаватели: Учеб. пособие. Екатеринбург: Изд-во Урал. ун-та, 2007. 248 с.

On a Recognition Problem of the Automata Languages

A. S. Shundeev, alex.shundeev@gmail.com, Institute of Mechanics, Lomonosov Moscow State University, Moscow, 119991, Russian Federation

Corresponding author:

Shundeev Aleksandr S., Leading Researcher, Institute of Mechanics, Lomonosov Moscow State University, Moscow, 119991, Russian Federation
E-mail: alex.shundeev@gmail.com

Received on January 18, 2017
Accepted on February 02, 2017

Finite state machines and automata languages are effective instruments of describing and modeling software systems. For research of the automata language properties one can use approaches, that traditionally belong to the field of data analysis.

For example, the problem of automata language recognition can be viewed as the problem of classifying all words over a given alphabet. One class of words is defined by an automata language. Another class of words is

the complement of this automata language. In the classical formulation of this task it is assumed that the classifier is completely defined. It is assumed that the finite state machine that recognizes the automata language or its complement language, is defined. In this case the solution is to minimize the classifier to reduce its computational complexity.

In this article, it is assumed that the classifier is unknown. It specifies only the training sample, which contains a finite number of reference representatives of the automata language and complement of this automata language. Accordingly, the task is to build an approximate classifier by analyzing the structure of the training sample elements. To do this, a new concept of approximate automata that acts as an approximation of the classifier is introduced. As the quality characteristics of the solution of the problem we introduce the concepts of weak and strong correctness of the approximate classifier. We investigate the relationship of these characteristics. An algorithm for constructing approximate machine automata that has the property of strong correctness and solve task of words classification is described.

Keywords: *DFA, automata language, classification problem, recognition problem, machine learning*

For citation:

Shundeev A. S. On a Recognition Problem of the Automata Languages, *Programmnaya Ingeneria,* 2017, vol. 8, no. 4, pp. 186—192.

DOI: 10.17587/prin.8.186-192

References

1. **James G., Witten D., Hastie T., Tibshirani R.** *An Introduction to Statistical Learning with Applications in R*, Springer, 2013, 426 p.
2. **Zamjatin A. P., Shur A. M.** *Jazyki, grammatiki, raspoznavateli*: Uchebnoe posobie (Languages, grammars, recognizers: Tutorial), Ekaterinburg, Izd-vo Ural. un-ta, 2007, 248 p. (In Russian).

ИНФОРМАЦИЯ

26—27 мая 2017 г. в Москве пройдет
XXI международная конференция в области обеспечения качества ПО

"Software Quality Assurance Days"

Конференция охватит широкий спектр профессиональных вопросов в области обеспечения качества, ключевыми из которых являются:

- методики и инструменты тестирования ПО;
- автоматизация тестирования ПО;
- подготовка, обучение и управление командами тестировщиков;
- процессы обеспечения качества в компании;
- управление тестированием и аутсорсинг;
- совершенствование процессов тестирования и инновации.

Подробности: http://sqadays.com

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

Сдано в набор 07.02.2016 г. Подписано в печать 21.03.2017 г. Формат 60×88 1/8. Заказ PI417
Цена свободная.

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