DOI: 10.25728/mlsd.2023.1264

ПОДХОД К ОРГАНИЗАЦИИ РАСПРЕДЕЛЕННОЙ МУЛЬТИСЕРВЕРНОЙ ОБРАБОТКИ НА ОСНОВЕ СЕТЕВОГО ОБОБЩЕНИЯ UNIX-КОНВЕЙЕРА

Асратян Р.Э.
Институт проблем управления им. В.А. Трапезникова РАН, Москва, Россия
rubezas@yandex.ru

Аннотация. Рассмотрены принципы организации сетевой службы RECS (Remote Executable Call Service), предназначенной для организации конвейерной обработки данных в мульти-серверной среде. Основная идея подхода заключается в переносе конвейерной обработки с клиентских рабочих мест на серверы и использовании ее для организации мульти-серверной обработки информационных запросов в распределенных информационных системах.

Ключевые слова: распределенные системы, Интернет-технологии, конвейерная обработка, мульти-серверная обработка, информационное взаимодействие.

Введение

Программный «конвейер» (pipeline), появившийся еще в 70-х годах в первых же версиях системы UNIX, является одним из наиболее изящных способов объединения возможностей нескольких программ для решения общей задачи [1,2]. То, что впоследствии он был воспроизведен на во всех основных платформах, включая Windows, Linux и Solaris, говорит само за себя. При работе из «командной строки» конвейер представляется в форме последовательности вида

$Cmd_1 \mid Cmd_2 \mid \ldots \mid Cmd_n$,

в которой каждый компонент Cmd_i является собой командой запуска какого-то модуля-обработчика. Взаимодействие модулей основано на том, что стандартный вывод Cmdi соединяется со стандартным вводом Cmd_{i+1} (i=1,2, … n-1), а результат работы последнего модуля выводится на экран. Очень важно, что поведение каждого обработчика в конвейере остается точно таким же, как при одиночном вызове: он должен читать входные данные со стандартного ввода, писать выходные в стандартный вывод и заканчивать свою работу после закрытия его стандартного ввода модулем-предшественником. Unix-конвейер на многие годы определил стиль работы с системой, основанный на знании уже готовых программных модулей и умении использовать их в разных сочетаниях [3].

Изначально Unix-конвейер был спроектирован как локальный программный механизм, в котором все модули-обработчики запускались на клиентской машине. Именно в этом качестве он был в значительной степени вытеснен из употребления с появлением графического оконного интерфейса как альтернативы работе из командной строки. Однако, появление Интернета и распределенных информационных систем [4] создают условия для создания новых способов воплощения идеи программного конвейера. Это в первую очередь относится к средствам организации распределенной мульти-серверной обработки, когда к обработке информационного запроса привлекаются сразу несколько серверов системы и множество программных модулей-обработчиков, работающих на этих серверах. К самым общим требованиям, к этим средствам можно отнести следующие:

- способность обеспечивать сетевое информационное взаимодействие между компонентами распределенных систем, работающих на разных сетевых узлах,
- наличие удобного программного интерфейса, дающего возможность обращения к сервисным модулям системы из произвольного клиентского модуля (а не из «командной строки»).

Ниже рассматриваются подход к построению таких средств, основанный на использовании идеи программного конвейера для организации распределенной обработки информационных запросов на основе перехода от архитектуры «клиент – сервер» к архитектуре «клиент – сервер, сервер, …, сервер».

1. Принципы организации сетевой служба RECS

Интернет-служба RECS (Remote Executables Call Service – служба удаленного вызова исполняемых модулей) основана на той же идее последовательной обработки клиентских данных цепочкой модулей-обработчиков, что и общепринятый программный конвейер. Однако имеется несколько важных отличий.

- RECS представляет собой сетевую службу, работающую не на клиентской машине, а на сервере. Все модули, участвующие в обработке, запускаются специальной постоянно активной программой – RECS-сервером. Для организации конвейерной обработки программа-клиент выполняет сетевое

соединение с обслуживающему ее RECS-серверу и передает ему составную командную строку вида Command1 , Command2 , … , Commandn для исполнения.

- Каждый элемент составной командной строки Commandi может являться просто командой запуска исполняемого модуля на обслуживающем сервере. Но может представлять собой более сложную конструкцию, называемую «вложенной командной строкой». Эта конструкция имеет вид «имя_сервера@команда» или даже «имя_сервера@(команда1, команда2, … , командаn)» и предназначена для запуска одного или нескольких исполняемых модулей-обработчиков на дополнительном удаленном RECS-сервере с заданным именем. При этом командная строка может иметь несколько уровней «вложенности», т.е. каждая «команда» может, в свою очередь, представлять собой аналогичную конструкцию с именем другого дополнительного RECS-сервера и произвольным списком команд и. т.д.
- RECS предоставляет клиентской программе удобный прикладной программный интерфейс (API), ориентированный на язык C++. Этот интерфейс поддержан специальной клиентской библиотекой функций и может использоваться, как в «консольных» программах, так и в программах с графической оконной архитектурой.

Чтобы сразу же получить представление о клиентском API, рассмотрим следующий пример кода на языке C++ (рис.1). Предположим, что клиентская программа обращается к обслуживающему серверу alpha.ru чтобы вычислить контрольную сумму всех исполняемых файлов (файлов типа «exe») в каталогах C:\WINDOWS\system32 на этом сервере (например, с целью проверки, изменился ли набор исполняемых файлов, или нет). При этом программа «хочет» использовать модуль-обработчик dirlist.exe, который считывает со стандартного ввода имя произвольного каталога, обрабатывает содержащиеся в нем файлы и заносит в стандартный вывод построчный список имен всех содержащихся в нем файлов и их контрольных сумм. Для решения поставленной задачи программа организует обращение к еще одному серверу (с именем beta.ru) на платформе UNIX или Linux, чтобы использовать уже имеющиеся в нем широко известные программы-обработчики grep и sort, выполняющие фильтрацию и сортировку полученного списка соответственно. Формирование конечного результата - окончательной 32-байтовой контрольной суммы от этого списка - выполняется ется снова на сервере alpha.ru с помощью известной программы md5sum.exe. Разумеется, мы предполагаем, что и на сервере alpha.ru и на сервере beta.ru установлено серверное программное обеспечением RECS.

```
char  Checksum [32];
RECSclnt  Rcln = new RECScln("~/cert/client_cert.pem.","~/cert/client_key.pem");
Rcln.Connect("alpha.ru");
Rcln.Process("dirlist, beta.ru$(grep '.exe', sort), md5sum");
Rcln<<"c:\\windows\\system32\n";
Rcln >> Checksum;
cout<<Checksum<<"\n"<<Rcln.GetSysMsg()<<"\n";
Rcln.Quit();
Rcln.Disconnect();
```

Рис. 1. Пример кода обращения к RECS из клиентской программы

Две первые строки примера содержат определения определяют двух переменных: строковой переменной Checksum и объектf класса RECSclnt, в котором описаны все свойства и методы, необходимые для работы с RECS. Как видно из рис. 1, конструктор класса получает в качестве параметров имена двух файлов, содержащих закрытый ключ и сертификат открытого ключа клиента (RECS использует защищенный протокол SSL/TLS [5] для безопасной передачи данных через Интернет, но в данной работе мы не будем останавливаться на технологиях криптозащиты).Третья строка кода выполняет сетевое соединение клиента с определенным RECS-сервером, а четвертая выполняет запуск трех удаленных модулей-обработчиков с помощью метода Process. Как видно из заданной составной командной строки, первый из обработчиков (dirlist.exe) запускается непосредственно на обслуживающем сервере (alpha.ru), два следующих (grep и sort) – на сервере beta.ru, а последний (md5sum) – снова на сервере alpha.ru. После запуска все модули-обработчики переходят к ожиданию появления данных на их стандартных вводах.

Пятая строка выполняет передачу на сервер символьную строку, содержащую имя проверяемого каталога (c:\\windows\\system32), которая сразу же попадает на стандартный ввод первого модуля-обработчика. Шестая строка - считывание конечного результата обработки и вывод на экран вычисленной контрольной суммы, а также итогового диагностического сообщения сервер.

Метод Quit вызывает завершение обработки запроса: последовательное закрытие стандартного ввода во всех обработчиках и прекращение их работы, а последняя строка – закрытие сетевого соединения с сервером alpha.ru (диалог клиента с сервером мог бы при необходимости продолжаться и дальше).

Разумеется, из данного фрагмента кода специально удалены операторы проверки успешности выполнения методов класса RECSclnt (например, метода Connect) и обработки исключительных ситуаций.

На рис. 2 проиллюстрирована структура информационных связей в мульти-серверной среде при обработке вышеприведенного примера.

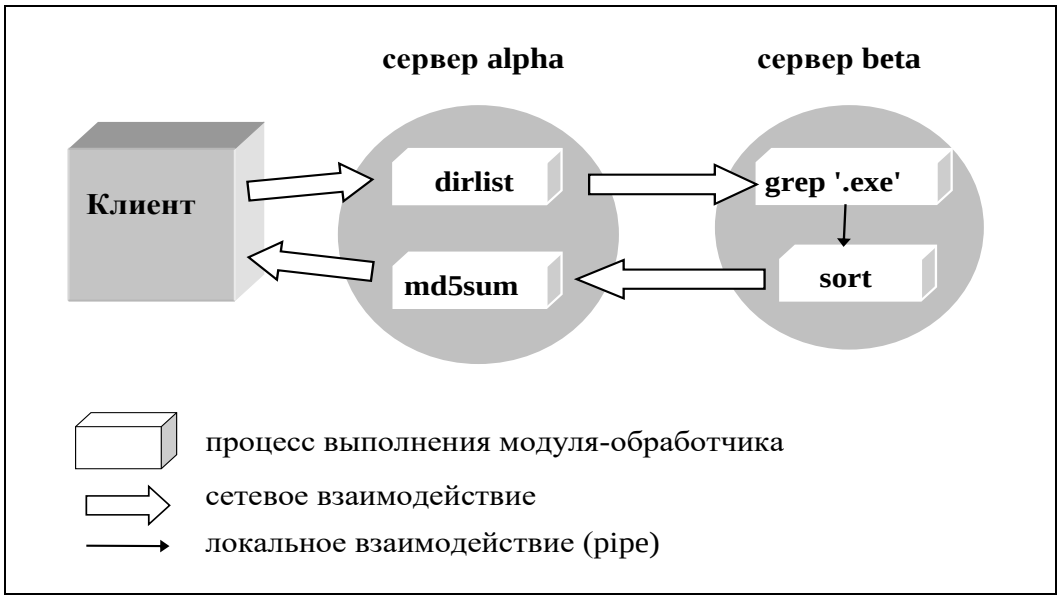


Рис. 2. Пример структуры конвейерных соединений

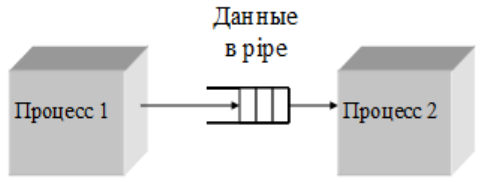
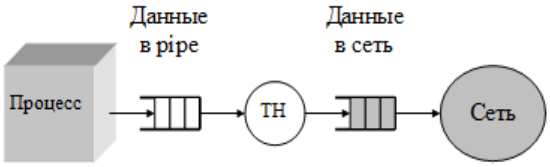
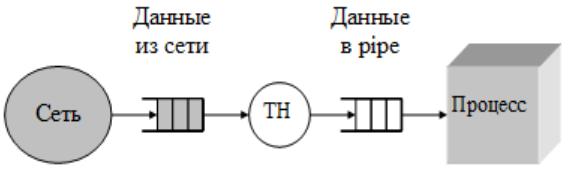
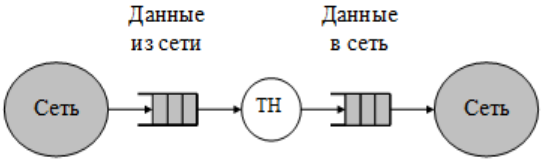
2. Типы конвейерных соединений

Обработка командной строки выполняется «рекурсивно»: каждый вовлеченный в обработку сервер получает относящийся к нему фрагмент командной строки от другого сервера с помощью точно такого же метода Process. В зависимости от того, на одном ли сервере выполняются взаимодействующие модули-обработчики или на разных, для организации связи между ними применяются различные типы конвейерных соединений (см. таблицу 1).

Рассмотрим в качестве примера структуру конвейерных соединений, созданную для выполнения командной строки A,B,beta.ru@C в сервере alpha.ru (см. рис. 3). Процессы выполнения модулей-обработчиков обозначены «кубиками». Как видно из рисунка, даже в таком простом примере задействованы все четыре типа соединений.

Так как невозможно подать данные из сетевого сокета [6,7] непосредственно на стандартный ввод обработчика «A», конвейерное соединение (1) обеспечивает передачу всех данных, полученных по сети от клиента, сначала в канал pipe, который соединен со стандартным вводом модуля-обработчика «A». Это соединение использует сокет, открытый при установлении сетевого соединения с клиентом и относится к типу «Socket-Pipe». Конвейерное соединение (2) представляет собой обычный pipeline между стандартным выводом обработчика «A» и стандартным вводом обработчика «B», организованный через неименованный канал pipe (в данном случае никаких Транспортных программных нитей не требуется).

Таблица 1. Типы конвейерных соединений

Тип конвейерного соединения	**Принцип организации**	**Пояснение**
Pipe		Обычный трубопровод для локального взаимодействия процессов на одном сервере. Один из процессов использует дескриптор записи в pipe в качестве стандартного вывода, а другой - дескриптор чтения из pipe в качестве стандартного ввода.
Pipe-Socket		Передача данных со стандартного вывода процесса на удаленный сервер через сеть Транспортная программная нить (ТН) обеспечивает непрерывный перенос данных из выходного дескриптора, pipe-а в сетевой сокет, соединенный с удаленным сервером.
Socket-Pipe		Прием данных из сети на стандартный ввод процесса. Транспортная программная нить (ТН) обеспечивает непрерывный перенос данных из сокета, соединенного с удаленным сервером, во входной дескриптор, pipe-а и далее на стандартный ввод процесса-получателя.
Socket-Socket		Транзитная передача данных через сервер без обработки. Транспортная программная нить (ТН) обеспечивает непрерывный перенос данных из одного сетевого сокета в другой.

Конвейерные соединения (3) и (4) относятся к типам «Pipe-Socket» и «Socket-Pipe» соответственно. Их задача заключается в обеспечении передачи данных от обработчика «B» на сервере alpha.ru к обработчику «C» на сервере beta.ru. Наконец, соединения (5) и (6), относящиеся к типам «Pipe-Socket» и «Socket-Socket» соответственно, обеспечивают возврат клиенту всех данных со стандартного вывода обработчика «C». Так как возврат данных непосредственно клиенту с сервера beta.ru невозможен, в качестве посредника используется сервер alpha.ru и конвейерное соединение типа «Socket-Socket». Отметим, что в конвейерных соединениях (1) и (6) используется одним и тем же сокетом для чтения данных из сети и для записи данных в сеть соответственно – сокет сетевого соединения между клиентом и сервером alpha.ru. То же самое можно сказать и о соединениях (4) и (5), а также о соединениях (6) и (3) – они используют одни и те же сокеты, обеспечивающие сетевое соединение между серверами alpha.ru и beta.ru.

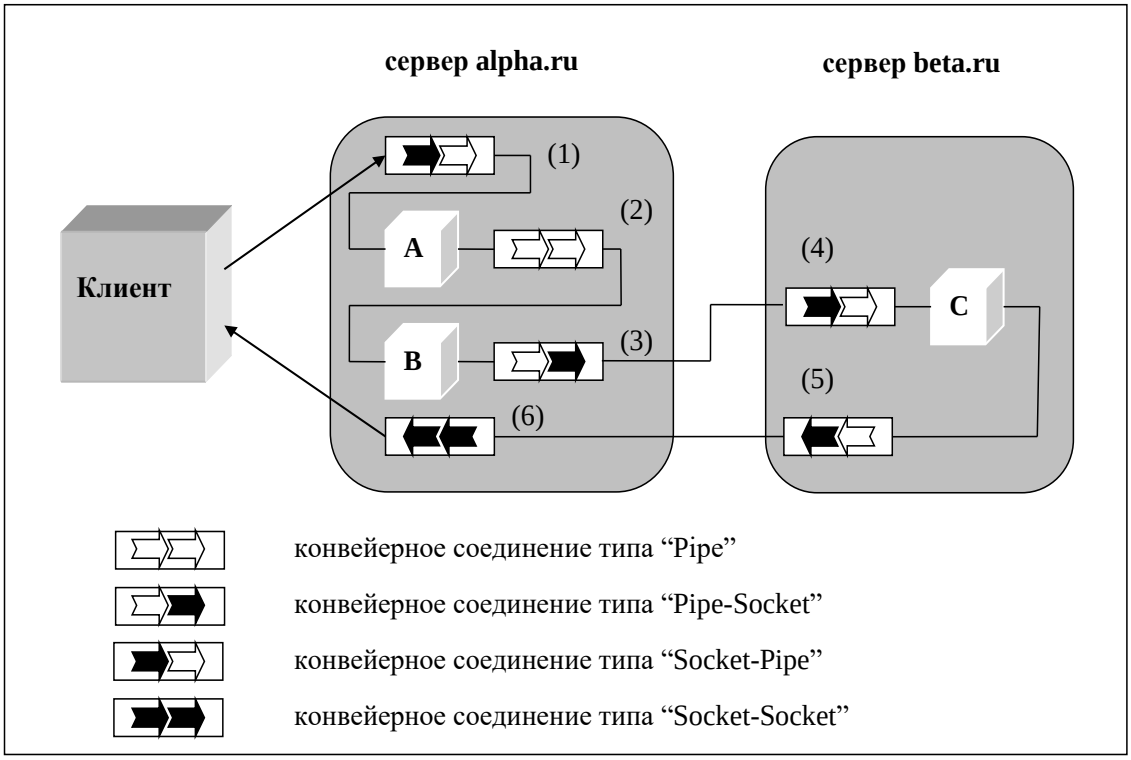


Рис. 3. Пример структуры конвейерных соединений

3. Временные оценки

Современные Интернет-технологии в принципе позволяют организовать последовательную конвейерную обработку данных в цепочке серверов. Например, готовое решение этой задачи может быть получено путем использования службы сетевого командного интерпретатора SSH [8], который позволяет оператору ввести с клавиатуры сложную командную строку, инициирующую мульти-серверную обработку. Главным недостатком SSH является отсутствие прикладного программного интерфейса, что весьма затрудняет его использование в разработках распределенных систем для организации взаимодействия между серверными и клиентскими компонентами. В самом деле, для решения этой задачи клиентской программе пришлось бы создать отдельный процесс для выполнения клиентской утилиты SSH с помощью системных вызовов fork() и exec (), а также организовать передачу обрабатываемых данных на его стандартный ввод с помощью системных вызовов pipe() и dup() [6]. Разумеется, подобное решение довольно «громоздко», требует определенных трудозатрат и вносит дополнительную задержку в обработку.

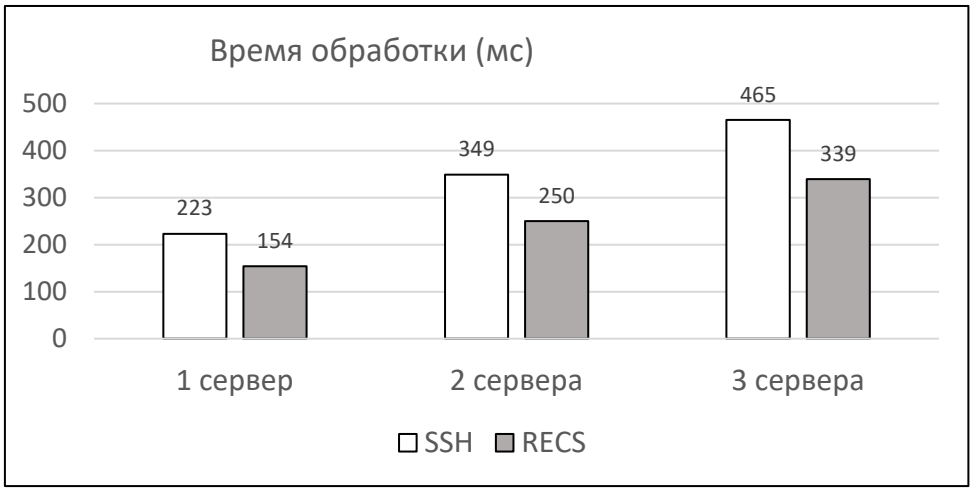


Рис. 4. Время выполнения конвейерной обработки в SSH и RECS

Для измерения временных задержек, связанных с использованием SSH в программных конвейерах, были проведены несколько простых экспериментов в лабораторной локальной сети с использованием

достаточно скромных четырех-ядерных серверов на платформе Linux Debian 9 и Debian 10. В SSH использовалась аутентификация по ключу [8]. Эксперименты заключались в задании составных командных строк в клиентской утилите SSH, инициирующих последовательную обработку данных в нескольких серверах (от 1 до 3). Например,
ssh rea@192.168.0.7 'ls -l /usr/lib | grep lib | sort' (обработка в одном сервере),
ssh rea@192.168.0.8 'ls -l /usr/lib | ssh rea@192.168.0.7 'grep lib' | sort' (обработка в 2-х серверах),
ssh rea@192.168.0.8 'ls –l /usr/lib | ssh rea@192.168.0.7 'grep lib | ssh rea@192.168.0.6 'sort'" (обработка в 3-х серверах).

Результаты экспериментов приведены на рис. 4. Там же приведены результаты измерения времени аналогичной обработки, проведенной с помощью RECS с теми же серверами и модулями-обработчиками, в одинаковых условиях и с одинаковым результатом. Разумеется, показанная на рис. 4 разница в скорости обработки практически неощутима при вводе команд «с клавиатуры» (для чего и предназначен SSH) но в разработках распределенных систем она может иметь существенное значение.

4. Заключение

Как уже отмечалось, появление графического оконного интерфейса у клиентских программ практически вытеснило применение Unix-конвейера в разработках распределенных систем. Главная мысль данной работы заключается в том, что этот конфликт может иметь вполне конструктивное разрешение: перенос конвейерной обработки с клиентских рабочих станций на серверы. Другими словами, новые формы реализации Unix-конвейера в виде серверных и мульти-серверных технологий могут пробудить новый интерес к этой замечательной идее.

Описанный в данной работе подход базируется на определенном сходстве между локальным каналом pipe и сетевым TCP-каналом между удаленными программами: оба типа каналов реализуют неструктурированные потоки байтов, оба поддержаны удобными блокирующими функциями ввода-вывода (с автоматической приостановкой выполнения при отсутствии данных в операции чтения или при переполнении буферов в операции записи) и т.п. Служба RECS построена таким образом, чтобы максимально скрыть от пользователя фундаментальные различия в физической природе каналов.

К важным преимуществам описанного подхода относится удобство отладки сервисных программ-обработчиков: каждая такая программа допускает автономную отладку вне сетевой среды, основанную на подаче тестового потока байтов на стандартный ввод и анализе стандартного вывода (фактически, это преимущество целиком «унаследовано» у общепринятого одно-машинного конвейера).

Описанный подход не является единственным возможным решением задачи организации конвейерной обработки в мульти-серверной среде. Одно из альтернативных решений описано в [9]. Оно основано на использовании структуры «защищенного сообщения» в стандарте CMS (Cryptographic message Syntax) в качестве единицы обмена и обработки в мульти-серверном конвейере (вместо неструктурированного потока байтов). Возможность конвейерной обработки в этом случае основана на том, что все сервисный функции-обработчики получают в качестве аргумента объект класса «защищенное сообщение и возвращают другой объект того же класса в качестве результата, что обеспечивает принципиальную совместимость этих функций по входу и выходу.

Литература

1. *Келли-Бутл С.* Введение в Unix. – М.: ЛОРИ, 1995. – 596 с.
2. *Negus C.* Linux Bible. – N.J.: Wiley, 2015. – 912 p.
3. *Таненбаум Э.* Современные операционные системы. – СПб.: Питер, 2002. – 1040 с.
4. *Гофф М.* Сетевые распределенные вычисления: достижения и проблемы. – М.: КУДИЦ-ОБРАЗ, 2005. – 320 с.
5. *Baka P, Schatten J.* SSL/TLS under lock and key. – Keyko books, 2020. – 132 p.
6. *Снейдер Й.* Эффективное программирование TCP/IP. – СПб.: Символ-Плюс, 2002. – 320 с
7. *Хант К.* TCP/IP. Сетевое администрирование. – СПб.: Питер, 2007. – 816 с
8. *Barret D.J.,Byrnes R.G.,Silverman R.E.* SSH, the Secure Shell. – O'Railly Media, 2005. – 672 p.
9. *Асратян Р.Э., Лебедев В.Н.* Интернет-служба конвейерной обработки защищенных информационных запросов в мультисетевой среде // Информационные технологии. 2017. N 8. – С. 589-597.