12 СЕКЦИЯ. ИНФОРМАЦИОННОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СИСТЕМ УПРАВЛЕНИЯ КРУПНОМАСШТАБНЫМИ ПРОИЗВОДСТВАМИ

DOI:

БЕЗОПАСНАЯ КОНВЕЙЕРНАЯ ОБРАБОТКА ДАННЫХ В РАСПРЕДЕЛЕННЫХ ИНФОРМАЦИОННЫХ СИСТЕМАХ СО МНОГИМИ ОБРАБАТЫВАЮЩИМИ СЕРВЕРАМИ[1]

Асратян Р.Э.

Институт проблем управления им. В.А. Трапезникова РАН, Россия, г. Москва

ул. Профсоюзная д.65

rea@ipu.ru

Аннотация: Рассмотрены принципы организации конвейерной обработки защищенных сетевых сообщений в мульти-серверной среде. Основная идея подхода к обеспечению информационной безопасности заключается в использовании реквизитов подписантов защищенного сообщения, содержащихся в сопровождающих электронных подписях, для разграничения доступа к информационным ресурсам (серверным обрабатывающим функциям) без необходимости централизованной регистрации пользователей и прав доступа.

Ключевые слова: распределенные системы, Web-сервисы, Интернет-технологии, информационный обмен, защита данных, информационная безопасность.

Введение

Сетевые технологии являются важнейшей составной частью современных распределенных информационных систем и в значительной степени, определяют их архитектуру и характеристики. Сегодня разработчики таких систем имеют в своем распоряжении целый ряд сетевых технологий, обладающих высокой универсальностью и гибкостью [1-3]. В их числе следует в первую следует упомянуть Web-технологии [2], имеющие замечательно широкий спектр применений от электронной прессы до распределенных вычислений. Однако универсальность имеет не только положительную, но и отрицательную сторону. Она зачастую становится препятствием для подготовки готовых решений целого ряда важных для разработчика задач, в том числе в области информационной безопасности [4,5]. Этим объясняется возрастающий интерес к созданию сетевых технологий, более точно сфокусированных на поддержку распределенных информационных систем.

В работе [6] описана новая сетевая служба PMS (Protected Message Service), цель создания которой заключалась в совместной реализации тесно связанных между собой средств сетевого информационного обмена и средств информационной защиты в составе одного программного класса. С точки зрения программиста эти средства поддержаны функциями-членами (методами) главного класса службы – класса «Защищенное сообщение» (PmsMessage), отображающего своего рода контейнер для хранения электронных документов (информационных запросов или ответов), снабженный одной или несколькими удостоверяющими электронными цифровыми подписями (ЭЦП). Важное отличие описываемой службы от, например, широко используемой технологии Web-сервисов [2] заключается в том, что она опирается не на модель вызова методов удаленных объектов, а на модель обмена сообщениями. Применительно к PMS это означает, что все сервисные обрабатывающие функции имеют одинаковую жесткую структуру: они получают объект класса «Защищенное сообщение» в качестве параметра и возвращают другой объект класса «Защищенное сообщение» в качестве результата. В соответствии с этим подходом класс PmsMessage включает программные методы для формирования и проверки ЭЦП, для шифрования и дешифрования информации и для распределенной обработки данных в мульти-серверной среде.

Упомянутая жесткая спецификация сервисных функций в PMS и принадлежность их входного параметра и их возвращаемого значения к одному и тому же программному классу дают принципиальную возможность организации т.н. «программного конвейера». Другими словами, речь идет об обработке клиентского информационного запроса не одной сервисной функцией, но

[1] *Работа выполнена в рамках фундаментальной темы «Модели, методы анализа и синтеза структуры и сценариев развития социально-экономических и технических систем управления, повышения их управляемости и безопасности функционирования в условиях неопределенности, структурных возмущений и чрезвычайных ситуаций» (направление №31 ПФНИ ГАН на 2013–2020 годы)*

последовательностью функций. При этом защищенное сообщение, сформированное каждой сервисной функцией, передается на вход следующей функции в последовательности, а последняя из функций возвращает защищенное сообщение клиенту. Разумеется, здесь легко увидеть аналогию 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 предметом данной статьи являются сетевая служба, сетевые сообщения и распределенные системы, наибольший интерес представляет мульти-серверная организация конвейера, в которой сервисные функции, задействованные в обработке информационного запроса, могут выполняться не на одном и том же, но на разных серверах сети.

1 Основы организации PMS и мульти-серверной конвейерной обработки

Основу сетевой службы составляет PMS-сервер – постоянно активная программа, обеспечивающая сетевой доступ к сервисным функциям для удаленных клиентов. Взаимодействие с клиентами осуществляется по собственному сетевому протоколу службы PMP (Protected Message Protocol), опирающемуся непосредственно на базовый сетевой протокол TCP [7]. Для того, чтобы сразу получить представление об организации PMS и конвейерной обработке в мульти-серверной среде, рассмотрим следующий пример. Предположим, что на сервере pharm.com имеется база данных (БД), хранящая данные о лекарствах и их свойствах, а также PMS-сервер, предназначенный для обслуживания информационных запросов к этой БД. Предположим также, что имеется сервисная функция GetAnalogs, которая получает наименование лекарства и возвращает список названий его аналогов. Рассмотрим действия клиентской программы на языке C#, необходимые для выполнения вызова этой функции.

Прежде всего программа клиента должна подготовить защищенное сообщение, содержащее информационный запрос. Следующие три оператора обеспечивают создание объекта класса PMSMessage (переменная Query), заполнение его данными (в данном случае, символьной строкой, содержащей название лекарства и формировании двух электронных подписей (например, исполнителя и руководителя), удостоверяющих подлинность запроса.

```
PmsMessage Query = new PmsMessage("Пенталгин");
PmsCertList  SenderCerts =new PmsCertList(new string [] {"Иванов",Петров"});
Query.AddSignatures(SenderCerts);
```

Следующие три оператора устанавливают сетевое соединение с сервером pharm.com. для этой цели используется переменная MyConn, относящаяся к другому важному классу службы – PMSConntction. Программный метод Connect этого класса позволяет открыть сетевое соединение с сервером pharm.com, а метод GetServerCertificate – сразу же получить сертификат PMS-сервера, содержащий публичный ключ для шифрования данных в сети.

```
PmsConnection MyConn = new PmsConnection();
MyConn.Connect("pharm.com");
PmsCertList  ServerCert = MyConn.GetServerCertificate();
```

Теперь у клиента есть все необходимое для отправки запроса на сервер. Метод Process, примененный к переменной Query, использует открытое сетевое соединение (переменная MyConn) для передачи запроса к сервисной функции GetAnalogs в зашифрованной форме (для шифрования используется сертификат сервера, занесенный в переменную ServerCert).

```
PmsMessage Reply=Query.Process (MyConn, "GetAnalogs", ServerCert);
if(Reply != null)
    Console.WriteLine (Reply.GetString());
else
    Console.WriteLine ("Ошибка: " +MyConn.ErrMsg);
MyConn.Disconnect();
```

Результат обработки заносится в переменную Reply класса PmsMessage. Программа клиента извлекает из нее символьную строку и выводит на консоль. В результате на экране должен появиться

список названий лекарств. Если предположить, что функция GetAnalogs формирует результат в форме XML-документа, он мог бы выглядеть так:

```
<medlist>
  <med>Пенталгин</med>
  <med>Бенальгин</med>
  <med>Пиралгин</med>
     . . .
  <med>Тетралгин</med>
</medlist
```

Автор намеренно не стал обременять пример операторами обработки ошибок и исключительных ситуаций, возникающих, например, вследствие отсутствия закрытых ключей для формирования ЭЦП или вследствие невозможности установления сетевого соединения с сервером.

Попробуем немножко усложнить наш пример. Предположим, что в сети имеется несколько дополнительных серверов с именами apo1.com, apo2.com, . . . , apoN.com, каждый из которых содержит информацию о свойствах и о наличии лекарств в определенной аптечной сети (или отдельной аптеке). Предположим также, что каждый из них оснащен PMS-сервером, в котором имеется сервисная функция FindDrugs, которая получает XML-документ со списком названий лекарств в уже знакомой нам форме и добавляет к нему всю имеющуюся информацию о каждом лекарстве по соответствующей аптечной сети (количество единиц на складе, цена, изготовитель, расфасовка и т.п.). Изменим оператор вызова метода Process в нашем примете, заменив имя единственной сервисной функции списком имен функций (т.н. «конвейерной строкой»):

```
PmsMessage Reply=Request.Process (MyConn,
"GetAnalogs,apo1.com/FindDrugs,apo2.com/FindDrugs, ... , apoN.con/FindDrugs",
ReceiverCert);
```

Отметим, что для вызова функций FindDrugs должно быть указано квалифицированное имя в формате «имя_сервера/имя_функции».

На рис. 1 проиллюстрирована конвейерная обработка метода Process в мульти-серверной среде.

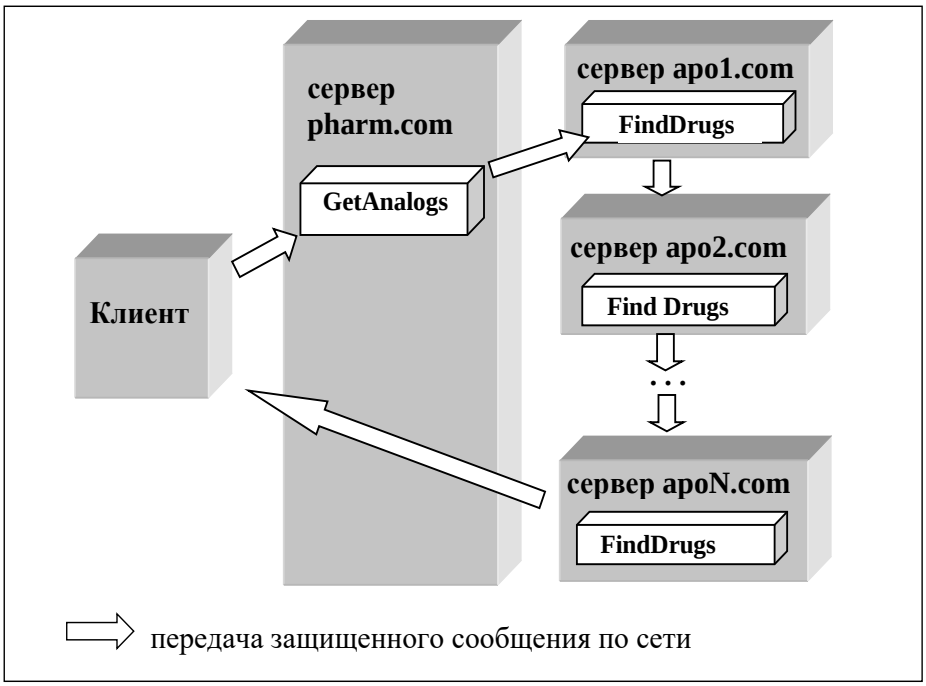

Рис 1. Пример конвейерной мульти-серверной обработки запроса

Как видно из рисунка, защищенное сообщение, содержащее результат выполнения каждой сервисной функции, передается на обработку следующей по списку функции в качестве входного параметра, а результат выполнения последней (apoN.com/FindDrugs) возвращается клиенту. (Важно подчеркнуть, что рисунок отображает общую логику процесса обработки, технически же передача

защищенного сообщения с одного дополнительного сервера на другой выполняется сервером pharm.com, который обеспечивает последовательное обращение ко всем серверам, упомянутым в конвейерной строке.) В результате вызова будет получен единый отчет в XML-формате, содержащий детальную информацию о заданном лекарстве и его аналогах по всем торговым сетям (см. рис.2).

```
<medlist>
  <med>Пенталгин</med>
  <med>Бенальгин</med>
  <med>Пиралгин</med>
  . . .
  <med>Тетралгин</med>
</medlist

<aponet name="Аптечная сеть №1">
  <res name="Пенталгин" number="80" price="200.00" made="Венгрия" ... </res>
  <res name="Беналгин" number="150" price="250.00" made="Чехия" ... </res>
  <res name="Пиралгин" number="200" price="150.00" made="Польша" ... </res>
   . . .
  <res name="Тетралгин" number="100" price="180.00" made="Россия" ... </res>
</aponet>

     . . .

<aponet name="Аптечная сеть №N">
  <res name="Пенталгин" number="70" price="280.00" made="Венгрия" ... </res>
  <res name="Беналгин" number="100" price="210.00" made="Чехия" ... </res>
  <res name="Пиралгин" number="150" price="240.00" made="Польша" ... </res>
  . . .
  <res name="Тетралгин" number="200" price="200.00" made="Россия" ... </res>
</aponet>
```

Рис. 2. Пример результата конвейерной обработки запроса в PMS

2 Разграничение прав доступа к ресурсам в сервере PMS

Одной из наиболее важных задач средств информационной безопасности в распределенных системах является аутентификация и разграничение доступа пользователей к информационным ресурсам [8,9]. Традиционный подход к решению этой задачи основан на использовании той или иной формы централизованной регистрации пользователей с использованием специальных серверов или служб аутентификации и авторизации, ответственных за управление учетными записями пользователей и ассоциированными с ними правами. Однако, такой подход предполагает централизованное администрирование, реализация которого в больших системах часто оказывается затруднительным.

В PMS используется другой подход к решению этой задачи, основанный на использовании реквизитах подписантов информационного запроса, содержащихся в удостоверяющих ЭЦП защищенного сообщения (точнее, в сертификатах открытого ключа, встроенных в ЭЦП). Так как доступ к информационным ресурсам осуществляется через сервисные функции, рассматриваемая задача решается с помощью встроенных в PMS средств управления доступом к сервисным функциям.

Работа этих средств проиллюстрирована на рис. 3. Как видно из рисунка, все сервисные функции сгруппированы в одну или несколько динамических библиотек функций, подключаемых к серверу в момент его запуска. С каждой библиотекой связан собственный файл конфигурации, в котором заданы общие свойства библиотеки и/или отдельных сервисных функций. К числу этих свойств относятся и требования к реквизитам подписантов: запрос к сервисной функции будет отклонен, если среди его подписантов нет ни одного, чьи реквизиты соответствуют этим требованиям.

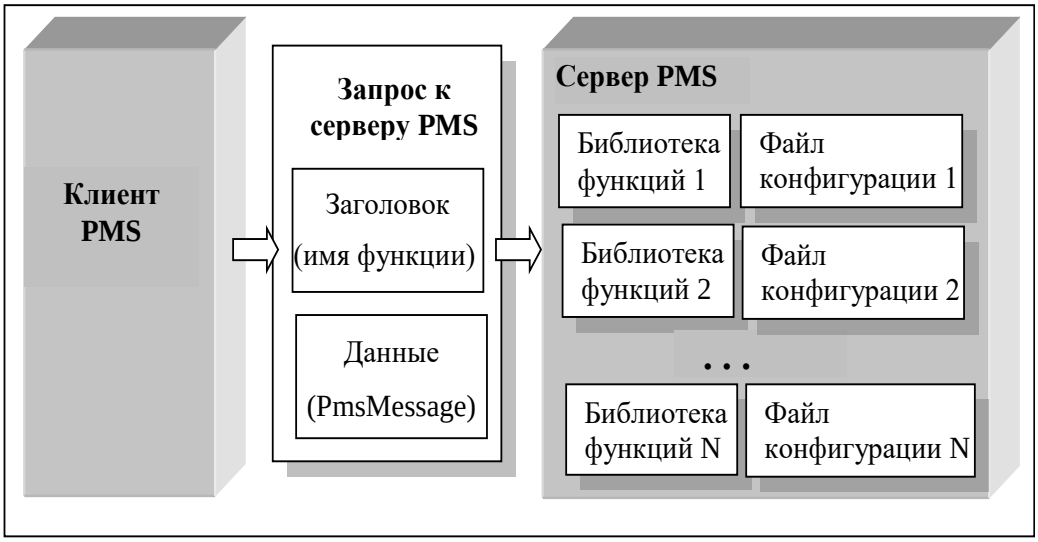

Рис. 3. Вызов сервисной функции в сервере PMS

Рассмотрим следующий пример. Предположим, что большая организация Titan имеет филиалы в областных центрах страны, а на PMS-серверах каждого филиала имеется библиотека сервисных функйий, предназначенная для регистрации служащих этого филиала (см. рис. 4). Эта библиотека содержит ряд относительно простых функций, обеспечивающих добавление, удаление и коррекцию записей о служащих в БД филиала (AddPerson, DeletePerson и CorrectPerson). Кроме того, она содержит функцию Report, которая позволяет сформировать отчет о кадровой статистике филиала за любой период времени (например, о динамике средней зарплаты служащих в разрезе должностей и подразделений).

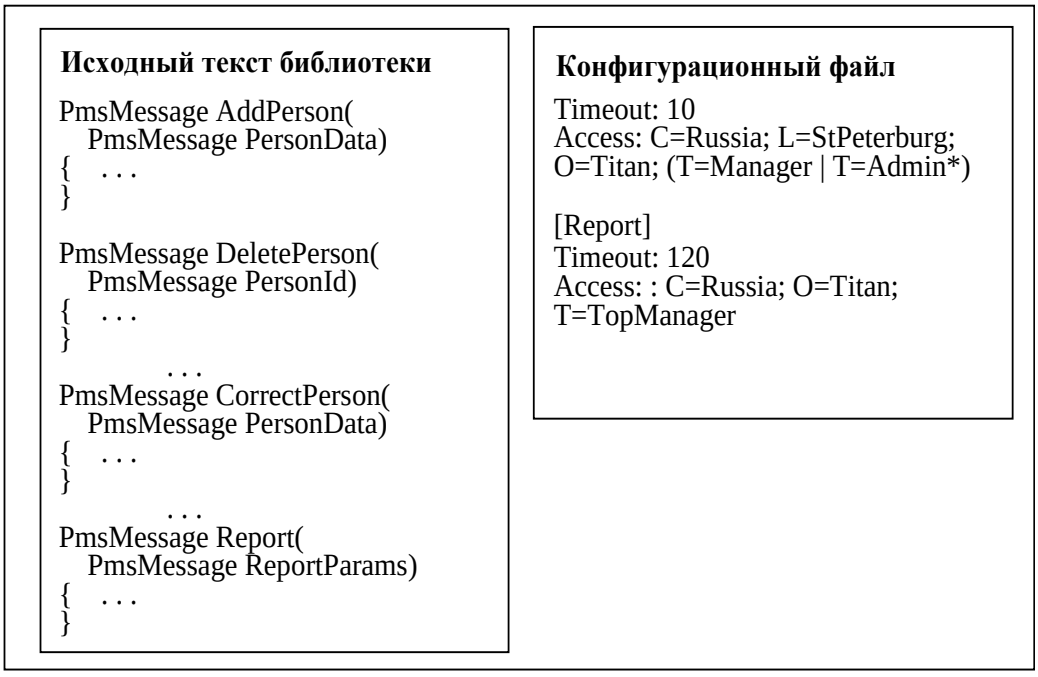

Рис. 4. Пример библиотеки сервисных функций и конфигурационного файла

На рис. 4 приведен пример конфигурационного файла этой библиотеки, содержащего значения основных характеристик сервисных функций. Первые три строки этого файла относятся ко всем простым функциям библиотеки и задают предельно допустимое время выполнения в секундах (Timeout) и ограничения доступа к ним (Access). Эти ограничения означают, что доступ к этим функциям разрешен только служащим данного филиала компании (L=StPeterburg; O=Titan) в должности менеджера или администратора (T=Manager | T=Admin*). Послкдующие строки конфигурационного содержат значения основных характеристик сервисной функции Report. Как видно из рисунка, предельно допустимое время выполнения у данной функции существенно выше, чем у

простых функций, а доступ к ней разрешен всем топ-менеджерам компании Titan (O=Titan). Поскольку эти характеристики заданы строго для функции Report, они имеют приоритет над ообщими характеристиками функций библиотеки, заданных в первых трех строках файла.

Как видно из приведенного примера, в формате конфигурационного файла используется нотация стандарта X509 для сертификатов открытого ключа [10].

3 Возможность параллельной обработки

Наличие множества обрабатывающих серверов создает предпосылки для применения параллельной обработки в системе. Поэтому, важным направлением развития PMS является введение средств распараллеливания обработки сложных информационных запросов в мульти-серверной среде, вписывающихся в модель обмена защищенными сообщениями. В новой версии PMS эти средства позволяют:

- перейти от строго последовательного вызова сервисных функций, поименованных в «конвейерной строке» к параллельному вызову отдельных функций, если логика обработки допускает их параллельное выполнение,
- определять сервисные обрабатывающие функции, которые имеют дело не с одиночными объектами класса «Защищенное сообщение», но с массивами таких объектов.

Организацию параллельной обработки в PMS лучше всего пояснить на уже знакомом примере, рассмотренном в разделе 1. Очевидно, что работа функций FindDrugs на разных серверах с формированием множества «частных» отчетов по различным аптечным сетям может выполняться параллельно. В результате будут сформированы N защищенных сообщений (объектов класса PMSMessage), содержащих маленькие XML-документы с тегом medlist и тегом aponet (со всеми вложенными тегами). От сервера pharm.com, обрабатывающего «конвейерную строку» понадобится умение запустить функции FindDrugs на серверах apo1.com, apo2.com, … , apoN.com в параллельном режие, дождаться окончания их работы и сформировать массив объектов класса PmsMessage, содержащий результаты их работы. Кроме того, чтобы избавить клиента от необходимости иметь дело с множеством частных отчетов понадобиться еще одна сервисная функция Join, способная получив в качестве входного параметра массив объектов класса PmsMessage, сформировать единый отчет по всем аптечным сетям (см. рис. 2). Изменим оператор вызова метода Process в нашем примете, заменив последовательность вызовов функции FindDrugs в квадратные скобки и добавив в конец «конвейерной строки» вызов функции Join:

```
PmsMessage Reply=Request.Process (MyConn,
"GetAnalogs,[apo1.com/FindDrugs,apo2.com/FindDrugs,...,apoN.con/FindDrugs], Join",
ReceiverCert);
```

Квадратные скобки в «конвейерной строке» означают переход в режим параллельной обработки. Логика обработки метода Process для данного случая проиллюстрирована на рис. 5. Отметим, что во всех дополнительных серверах функция FindDrug получает в качестве входного параметра одну и ту же информацию: список имен лекарств, сформированный функцией GetAnalogs и размещенный в объекте класса PmsMessage.

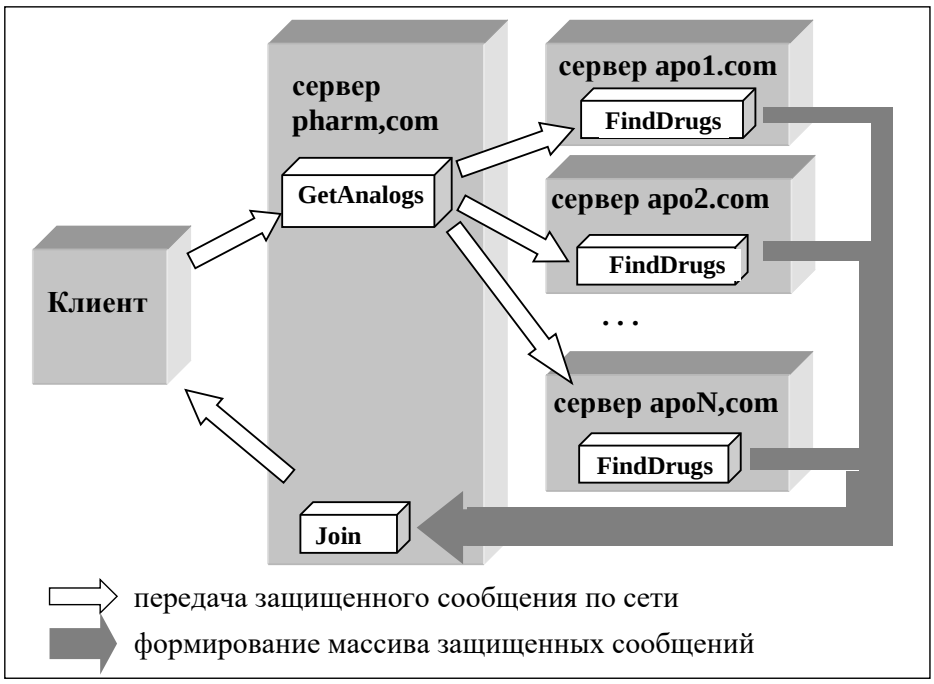

Рис 5. Пример параллельной мульти-серверной обработки запроса

Выигрыш от применения параллельной обработки в данном случае зависит от числа дополнительных серверов N, а также от времен выполнения сервисных функций GetList, FindDrugs и Join. Однако, даже при небольших N он может быть весьма значительным. На рис. 6 приведен один из результатов экспериментального сравнения времени последовательной и параллельной обработки в рассмотренном примере для N=3. Время выполнения функции GetAnalogs равнялось 0.1 сек., время выполнения функции GetAnalogs равнялось 0.3 сек, а время выполнения функции Join составляло менее 0.01 сек. В эксперименте была использована криптосистема «КриптоПро» версии 3.6., соответствующая требованиям действующих в России ГОСТов в области криптографической защиты информации. Эксперименты проводились в одинаковых условиях (скоростная лабораторная сеть 100 мбит/сек., четырехъядерные серверы с тактовой частотой 2.4.ГГц) и с одинаковым результатом.

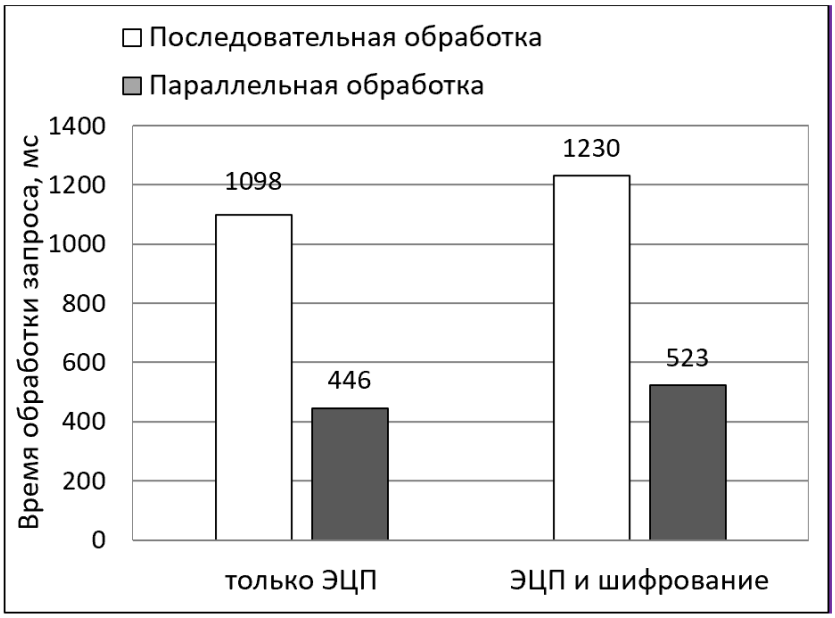

Рис 6. Пример соотношения скоростей обработки

Заключение

Основная идея PMS заключается в отказе от сетевой обработки бесконечного многообразия программных типов (как встроенных, так и определенных пользователем) и в замене всего этого многообразия понятием защищенного сообщения, как «единицы» обмена и защиты. Здесь можно провести аналогию с государственной почтовой службой, которая имеет дело только с понятием почтового контейнера – ящика, защищенного печатями и штемпелями, а не с огромной номенклатурой пересылаемых вещей.

В данной работе автор попытался показать, что отказ от модели вызова методов удаленных объектов с любым количеством и любыми типами параметров в пользу более «жесткой» модели обмена защищенными сообщениями позволяет не только тесно соединить функции информационного обмена и информационной защиты, но также

- обеспечить разграничение доступа к информационным ресурсам в мульти-серверных системах на основе реквизитов подписантов информационных запросов, содержащихся в электронных подписях, без централизованной регистрации пользователей,
- организовать последовательную конвейерную обработку информационного запроса в цепочке серверов по схеме «клиент – сервер$_1$ – сервер$_2$ - . . . – сервер$_N$»,
- создать основу для организации параллельной обработки информационного запроса в много-серверной среде на основе организации обменов массивами защищенных сообщений.

Литература

1. *Хант К. TCP/IP*. Сетевое администрирование. – СПб.: Питер, 2007. – 816 с
2. *Jackson J.C.* Web Technologies: A Computer Science Perspective. –. London: Pearson, 2011. – 574p.
3. *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.
4. *Згоба А.И., Маркелов Д.В., Смирнов П.И.* Кибербезопасность: угрозы, вызовы, решения / Вопросы кибербезопасности, 2014, № 5. С.30 – 38.
5. *Козлов А.Д., Орлов В.Л.* Методы и средства обеспечения информационной безопасности распределенных корпоративных систем. – М.: ИПУ РАН, 2017. – 156 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. *Смит Р.Э.* Аутентификация: от паролей до открытых ключей. – М.: Вильямс, 2016. – 432 с.
9. *Boyd C., Mathuria A.* Protocols for authentication and key establishment. – Berlin:Springler,2012. – 321 с.
10. *Полянская О.Ю., Горбатов В.С.* Инфраструктуры открытых ключей. – М.: Бином, 2013. – 368 с.