СЕКЦИЯ 12: ИНФОРМАЦИОННОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ СИСТЕМ УПРАВЛЕНИЯ КРУПНОМАСШТАБНЫМИ ПРОИЗВОДСТВАМИ

ИНТЕРНЕТ - СЛУЖБА СО СРЕДСТВАМИ ПАРАЛЛЕЛЬНОЙ ОБРАБОТКИ ЗАЩИЩЕННЫХ ИНФОРМАЦИОННЫХ ЗАПРОСОВ В МУЛЬТИ - СЕРВЕРНЫХ РАСПРЕДЕЛЕННЫХ СИСТЕМАХ

Асратян Р.Э.

Институт проблем управления им. В.А. Трапезникова,
Россия, г. Москва, ул. Профсоюзная, д.65

rubezas@yandex.ru

Аннотация: Рассмотрены принципы организации сетевой службы, предназначенной для реализации параллельной обработки защищенных запросов в распределенных информационных системах, ориентированных на работу в сложных сетевых средах со многими обрабатывающими серверами. Описано применение этой службы для комбинирования последовательной («конвейерной») и параллельной обработки запросов в мульти-серверной среде.

Ключевые слова: распределенные системы, Web-технологии, Интернет-технологии, параллельные вычисления, информационное взаимодействие, информационная безопасность.

Введение

Практически все современные информационные системы (ИС) проектируются из расчета на сетевую среду и сетевое взаимодействие. Даже самые небольшие из них сегодня как правило включают выделенный сервер приложений и/или выделенный сервер баз данных. Те же, которые принято называть крупномасштабными, зачастую используют десятки и даже сотни территориально распределенных серверов. Сегодня уже трудно представить себе ИС, работающую вне Интернета или, по крайней мере, частной сети.

Использование мульти-серверной среды в ИС имеет ряд особенностей, отличающих их от универсальных систем распределенных вычислений в сетевой среде (Grid-системы, Hadoop, Spark или Disco [1-3]). Если последние имеют целью достижение максимальной производительности за счет создания иллюзии единого мощного вычислительного ресурса, построенного на основе множества сетевых узлов, то распределенные ИС как правило решают задачу создания иллюзии единого информационного ресурса, построенного на основе множества серверов баз данных и серверов приложений. Причем задача повышения производительности в ИС обычно уступает «первый план» задачам обеспечения удобства конечного пользователя и информационной безопасности [4-6].

Разумеется, разработчики распределенных ИС всегда уделяли внимание увеличению скорости обработки информационных запросов. В том числе, за счет параллельной обработки запроса на нескольких серверах, в нескольких БД (если логика обработки допускает такую возможность). Например, обслуживающий запрос Web-сервис [7,8] может ретранслировать его сразу нескольким удаленным Web-сервисам, сопряженным с несколькими БД для выполнения параллельного поиска. Но подобные приемы как правило реализуются на уровне множества частных решений, приспособленных к потребностям конкретных систем. Что повсеместно приводит к потерям во времени разработки и, зачастую, в информационной безопасности (т.к. «бреши» в защите данных чаще всего локализуются в подобных частных решениях).

В работе [9] описана новая сетевая служба PMS (Protected Message Service), разработанная для поддержки безопасной обработки информационных запросов в распределенных ИС. 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 программного 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 (каждая библиотека может рассматриваться как отдаленный аналог Web-сервиса в .NET), и становятся доступными для клиентских компонент.

Базисное свойство PMS – фиксированная спецификация сервисных функций с объектами одного и того же класса на входе и на выходе функции – дает принципиальную возможность организации «программного конвейера», основанного на направлении объекта класса «Защищенное сообщение» с выхода одной функции на вход другой. Другими словами, речь идет об обработке 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 (PMS-конвейер). Разумеется, здесь имеется прямая аналогия с известным еще с первых версий системы UNIX механизмом «трубопровода» (pipeline), основанном на последовательном соединении стандартных выводов и вводов у нескольких процессов в компьютере.

Логика «конвейера» всегда предполагала строго последовательную обработку данных. Однако, возможности сложных сетевых сред с десятками и сотнями сетевых узлов, в которых работают многие современные распределенные системы, пробуждают интерес к средствам параллельной обработки информационных запросов, как к источнику повышения производительности [10,11]. В данной работе рассматриваются основы организации новой версии PMS, включающей механизмы распределенной обработки защищенных сообщений в мульти-серверной среде.

1 Последовательная конвейерная обработка запросов в PMS

Инструментарий клиента PMS опирается на два основных программных класса: PmsMessage (Защищенное сообщение) и PmsConnection (Защищенное соединение) (см. Таб. 1).

Таблица 1. Основные классы и программные методы PMS

Программный класс	**Основные функции в составе класса**	**Описание функции**
PmsMessage	PmsMessage	Семейство конструкторов, позволяющих создавать объекты класса и инициировать их пользовательскими данными разных типов (массив байтов, символьная строка, символьная строка с несколькими двоичными приложениями т.п).
	AddSignatures	Добавление одной или нескольких ЭЦП к защищенному сообщению
	CheckSignature	Проверка подписи с заданным порядковым номером в защищенном сообщении с извлечением реквизитов подписанта
	GetBytes GetString GetDocument	Семейство функций для извлечения данных разных типов из объектов класса: массив байтов, символьная строка, символьная строка с несколькими двоичными приложениями и т.п.
	Process	Отправка защищенного сообщения на обработку в сервер PMS и получение результата обработки в форме другого объекта класса PmsMessage.
PmsConnection	Connect	Открытие сетевого соединения с сервером PMS по заданному Интернет-имени или адресу.
	GetCertificate	Получение сертификата сервера по открытому соединению с проверкой валидности сертификата и реквизитов владельца
	Disconnect	Закрытие сетевого соединения с сервером PMS.

Класс PmsMessage представляет собой своего рода «контейнер» для пользовательских документов, оснащенный специальными функциями-членами (методами) для формирования и для проверки ЭЦП, а также для выполнения шифрации и дешифрации данных, размещенных в этом «контейнере». Кроме того, класс PmsMessage оснащен сетевыми свойствами, позволяющими клиентской программе отправить объект этого класса (вместе со всеми, содержащимися в нем документами) на обработку в любой сервер PMS, с которым установлено сетевое соединение, а также получить от сервера другой объект этого класса, содержащий результаты обработки.

Класс PmsConnection представляет защищенное сетевое соединение между клиентом и сервером PMS. Содержащиеся в нем программные методы позволяют клиенту открывать и закрывать сетевые соединения с серверами PMS, запрашивать у сервера сертификат открытого ключа, проверять валидность этого сертификата и реквизиты его владельца (защита от сфальсифицированного сервера) и выполнять шифрацию защищенных сообщений при отправке их на сервер PMS в автоматическом режиме.

Как уже отмечалось, все сервисные обрабатывающие функции группируются в одну или несколько динамических библиотек, которые подключаются к серверу PMS в момент его запуска. 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),

в которой параметр Inpt - входное сообщение (запрос); параметр Conf – конфигурационные данные в форме массива строк вида «имя=значение» (считываются из конфигурационного файла сервераPMS); строковый параметр Msg на входе принимает значение опционального параметра функции, 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.

Собственно, передача защищенного сообщения на обработку в сервер PMS осуществляется с помощью программного метода Process. Самым важным параметром этого метода является «конвейерная строка», задающая последовательность запуска сервисных функций для обработки запроса. Первая функция в последовательности получает в качестве фактического параметра объект класса PmsMessage, полученный от клиента, а каждая последующая функция – результат выполнения предыдущей. Результат выполнения последней (или единственной) сервисной функции возвращается клиенту. В качестве примера рассмотрим оператор вызова программного метода Process, демонстрирующий различные возможности конвейерной обработки.

```
PmsMessage Reply=Query.Process (MyConnection,
"Lib1.Func1, Srv2/Lib2.Func2, Srv3/(Lib3.Func3,Lib4.Func4), Lib5.Func5  param");
```

В данном примере заключенная в кавычки конвейерная строка включает 4 основных элемента, разделенных запятыми. Первый из этих элементов обозначает вызов функции Func1 из серверной библиотеки Lib1 непосредственно на обрабатывающем сервере, второй – временную передачу запроса в удаленный сервер Srv2 для его обработки там с помощью функции Func2 из библиотеки Lib2, третий элемент также обозначает временную передачу запроса в другой удаленный сервер (Srv3) для последовательной обработки его с помощью двух функций, выделенных круглыми скобками. Последовательность завершается вызовом функции Func5 из серверной библиотеки Lib5 непосредственно на обрабатывающем сервере с передачей ей опционального строкового параметра (param).

Для организации взаимодействие между клиентом и сервером служба PMS использует собственный протокол PMP ((Protected Message Protocol), занимающий «уровень приложения» в иерархии семейства протоколов TCP/IP [12]. PMP, в свою очередь, опирается на стандарт CMS (Cryptographic Message Syntax) для представления защищенных данных в сети (см. RFC 5652 в https://tools.ietf.org/html/rfc5652).

2 Параллельная обработка запросов в PMS

Переход от строго последовательной обработки к параллельно-последовательной в новой версии PMS основан на использовании массива защищенных сообщений в качестве «единицы обмена» в сетевом взаимодействии. Этот переход был выполнен согласно следующим принципам.

- Сервисные функции получают и возвращают не один объект класса PmsMessage, а произвольный массив таких объектов. Другими словами, спецификация сервисной функции в новой версии выглядит так:
- PmsMessage [] *имя_функции* (PmsMessage [] Inpt, string [] Conf, ref string Msg)
- (разумеется, и входной и выходной массивы объектов PmsMessage могут по-прежнему включать только один элемент, но могут и несколько).
- В структуру конвейерной строки вводится разметка групп параллельно и последовательно выполняющихся элементов с помощью квадратных и фигурных скобок соответственно.
- В качестве примера рассмотрим следующую конвейерную строку:

"Lib1.Func1, [Srv2/Lib2.Func2, {Lib3.Func3, Lib4.Func4, Lib5.Func5}, Srv3/(Lib6.Func6, Lib7.Func7)], Lib8.Func8"

Принципы обработки этой конвейерной строки в сервере PMS проиллюстрированы на рис. 1. Последовательно выполняемые элементы разнесены по горизонтали, а параллельно выполняемые – по вертикали.

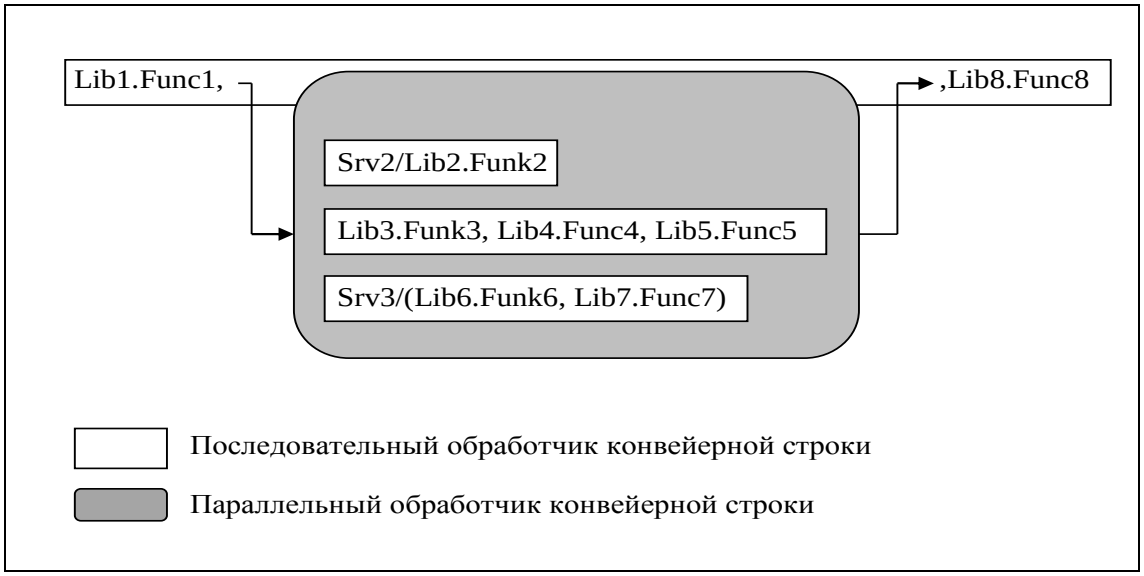

Рис. 1 Параллельно-последовательная обработка конвейерной строки

Как видно из рисунка, три группы элементов конвейерной строки, которые были выделены квадратными скобками, выполняются параллельно. Первая группа включает единственный элемент, который обозначает вызов обрабатывающей функции на удаленном сервере Srv2. Вторая группа включает последовательный вызов трех функций непосредственно на обрабатывающем сервере (в данном случае группировка выполнена с помощью фигурных скобок). Наконец, третья группа подразумевает последовательное выполнение двух обрабатывающих функций на удаленном сервере Srv3 (конвейерная строка "Lib6.Func6, Lib7.Func7" целиком передается на выполнение в удаленный сервер). Отметим, что на вход первой сервисной функции в каждой группе будет передан один и тот же объект класса PmsMessage, сформированный сервисной функцией Lib1.Funk1, а на вход функции Lib8.Funk8 будет передан массив из трех таких объектов, сформированных сервисными функциями всех трех параллельно выполненных групп элементов.

Логика обработки конвейерной строки сосредоточена в двух программных классах: Последовательный обработчик конвейерной строки (ПСО) и Параллельный обработчик конвейерной строки (ПРО). При обработке каждого обращения от клиента сервер PMS сразу же создает экземпляр класса ПСО и передает ему на обработку полученную конвейерную строку вместе с полученным массивом защищенных сообщений (объектов класса PmsMeccage). ПСО обеспечивает последовательное выделение каждого элемента из конвейерной строки и инициирует его обработку: или вызов соответствующей сервисной функции из собственных библиотек или же обращение к удаленному серверу.

Если ПСО обнаруживает группу параллельно выполняемых элементов в конвейерной строке, то он полностью выделяет ее, создает экземпляр класса ПРО и передает ее этому объекту на выполнение вместе с массивом объектов PmsMessage, сформированным последней выполненной сервисной функцией. Логика работы ПРО включает:

- выделение всех параллельно выполняемых элементов из полученной конвейерной подстроки,
- создание отдельного экземпляра класса ПСО для каждого выделенного элемента и создание параллельных программных нитей (потоков) для организации одновременной работы каждого из экземпляров,
- запуск всех созданных программных нитей с передачей каждой соответствующего элемента конвейерной строки и массива объектов класса PmsMessage, полученного от первичного ПСО,
- ожидание конца работы всех созданных программных нитей, объединение результатов их работы в единый массив объектов класса PmsMessage и возврат его в ожидающий ПСО для продолжения обработки (очень важно, что результирующий массив формируется не в порядке завершения нитей, но в порядке, заданном в конвейерной строке).

Подчеркнем, что «вторичные» экземпляры класса ПСО, созданные для обработки параллельно выполняемых элементов и последовательностей элементов, ничем не отличаются от «первичного», созданного при обнаружении обращения от клиента. Это, в частности, означает, что при обнаружении группы параллельно выполняемых элементов в обрабатываемой конвейерной подстроке «вторичный» экземпляр класса ПСО создаст, в свою очередь, новый экземпляр класса ПРО и организует взаимодействие с ним по вышеуказанным правилам.

3 Временные оценки

Эксперименты с новой версией PMS проводились в лабораторной локальной сети (100 мбит/сек) с использованием нескольких четырехъядерных серверов с тактовой частотой 2.4.ГГц и операционной средой Window 2003 Server, а также одноядерной клиентской рабочей станции с тактовой частотой 2.8 ГГц. Цель экспериментов заключалась в подтверждении корректности вышеописанных принципов организации сетевой службы, а также в получении количественных оценок того ускорения обработки, которое могут дать новые средства распараллеливания в условиях применения средств криптозащиты.

Рассмотрим один из таких экспериментов. Предположим, что у нас имеется четыре сервера PMS с именами Gun, Prsr, Prsf и Prsi, причем сервер Gun связан с базой данных, хранящей сведения об охотничьем оружии, а серверы Prsr, Prsf и Prsi связаны с базами данных, хранящими сведения о физических лицах, объявленных в региональный розыск, федеральный розыск и международный розыск соответственно. Допустим также, что

- в библиотеке Lb на сервере Gun имеется сервисная функция Owner, которая получает во входном сообщении серийный номер оружия и возвращает в выходном сообщении полное имя и дату рождения владельца одной строкой,
- в библиотеке Lb на серверах Prsr, Prsf и Prsi имеется сервисная функция FindPerson, которая получает в первой строке входного сообщении полное имя и дату рождения физического лица в качестве условий поиска, отыскивает в базе данных детальные сведения о физических лицах (гражданство, реквизиты документов и т.п.), отвечающих этим условиям, добавляет в конец сообщения эти сведения построчно и возвращает результат в качестве выходного сообщения.

Если в данных условиях клиентская программа поместит серийный номер оружия в защищенное сообщение (объект класса PmsMessage), сформирует для него одну или несколько электронных подписей, установит соединение с сервером Gun и передаст сообщение на обработку с помощью метода «Process», задав следующую конвейерную строку

"Lb.Owner, Prsr/Lb.FindPerson, Prsf/Lb.FindPerson, Prsr/Lb.FindPerson",

то в результате последовательной обработки в четырех серверах он получит защищенное сообщение, содержащее результат: детальные сведения о всех разыскиваемых лицах, которые могут иметь отношение к оружию с заданным серийным номером.

Очевидно, что поиск физических лиц в трех базах данных можно выполнять параллельно, а не последовательно. Для использования средств распараллеливания новой версии PMS нам понадобиться еще одна сервисная функция на сервере Gun – функция Join, которая получает на входе массив защищенных сообщений и формирует одно выходное сообщение, содержащее объединение (конкатенацию) данных, содержащихся во входном массиве. Если повторить эксперимент с использованием следующей конвейерной строки

"Lb.Owner, [Prsr/Lb.FindPerson, Prsf/Lb.FindPerson, Prsi/Lb.FindPerson], Lb.Join",

то клиент получит ту же информацию, что и в предыдущем случае (и так же в единственном защищенном сообщении).

На рис. 2 отображены оценки времени обработки в миллисекундах для двух рассмотренных экспериментов в форме диаграммы (белый и серый столбики) в двух режимах: при применении только ЭЦП и при применении ЭЦП вместе с шифрованием данных на основе криптосистемы «КриптоПро CSP» версии 3.6. В обоих экспериментах время операции поиска владельца паспорта в базе данных оружия по его серийному номеру составляло 0.1 секунды, а время поиска детальной информации о физических лицах в базах данных физических лиц – 0.3 секунды.

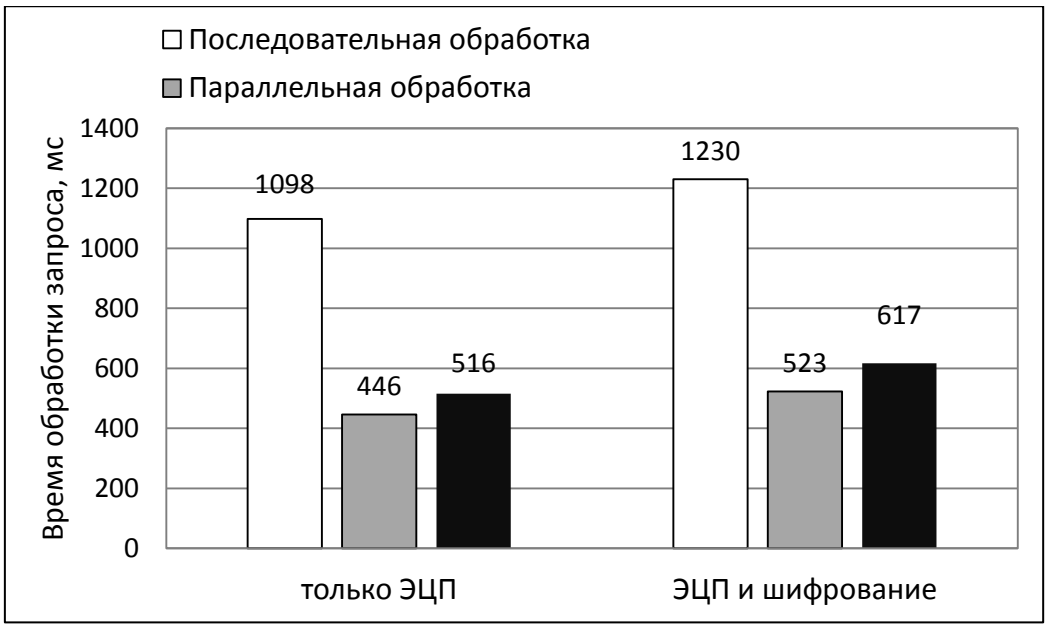

Рис 2. Пример соотношения скоростей обработки

Разумеется, распараллелить процесс поиска данных можно и на уровне клиента. Представим себе клиентскую программу, которая

- сначала обращается к серверу Gun и с помощью сервисной функции Owner получает полное имя и дату рождения владельца оружия,
- порождает 3 параллельных программных нити (потока), в каждой из которых он организует операцию поиска в базах данных физических лиц с помощью серверов Prsr, Prsf и Prsi и сервисной функции FindPerson,
- Дожидается завершения обработки во всех трех нитях и объединяет результаты поиска из трех полученных защищенных сообщений в один документ результата.

Очевидно, что время работы «параллельной» клиентской программы будет практически таким же, как и время параллельной обработки на сервере Gun (серый столбик на рис. 2). Однако, данный подход может встретить серьезные трудности.

Дело в том, что прямой вызов серверов Prsr, Prsf и Prsi из клиентской программы может оказаться невозможным, так как эти серверы могут не «знать» подписи клиента. При проверке ЭЦП в запросе сервер не только контролирует целостность входного сообщения и валидность сертификата подписанта, но, также, выборочно сверяет реквизиты подписанта (имя, должность, организация и т.п.) со. «списком доступа», содержащимся в его конфигурационных данных. Этот список содержит реквизиты «известных» серверу абонентов в формате, соответствующем стандарту X509, но с возможностью использования символов-заместителей. Например, строка списка доступа «CN=Сервер PMS*» открывает доступ к серверу всем абонентам, имя которых начинается со строки «Сервер PMS», за которой может следовать все, что угодно. Если среди подписантов запроса нет ни одного, соответствующего списку доступа, сервер отвергнет запрос.

При большом число серверов и клиентов в системе организация, в которой «все знают всех» становится затруднительной. Более конструктивный подход заключается в ограничении «круга знакомств» каждого сервера. Например, в вышеприведенном примере сервер Gun должен «знать» подписи всех обслуживаемых им клиентов, а каждому из серверов Prsr, Prsf и Prsi достаточно знать подпись сервера Gun (когда сервер PMS посылает запрос на исполнение другому серверу, он всегда добавляет свою ЭЦП к этому запросу). Это означает, что в вышеописанной схеме распараллеливания на уровне клиента программным нитям придется обращаться к серверам Prsr, Prsf и Prsi через сервер Gun, используя три отдельных вызова с короткими конвейерными строками "Prsr/Lb.FindPerson", "Prsf/Lb.FindPerson" и "Prsi/Lb.FindPerson" соответственно, что существенно увеличит число передач защищенных сообщений между сетевыми узлами и число вызовов функций криптосистемы. Время обработки запроса при данном подходе отображено на рис.2 черным столбиком. Как видно из рисунка, использование средств параллельной обработки в сервере PMS не только значительно проще для разработчика (так как является готовым и отлаженным решением), но и предпочтительнее с точки зрения быстродействия.

Из рассмотренного примера видно, что распараллеливание может дать значительный выигрыш в скорости при обработке одиночных запросов. Однако, на том же примере можно убедиться, что при высокой нагрузке на сервер соотношение скоростей обработки может быть более сложным.

Сервер PMS имеет многоканальную организацию, т.е. он создает по крайней мере один отдельный обрабатывающий программный поток («нить») для обслуживания каждого запроса. На рис 3 приведены результаты экспериментальной оценки быстродействия PMS в условиях высокой нагрузки: при обработке пакетов одновременно поступивших информационных запросов с применением криптозащиты (ЭЦП и шифрования). Кривые, представленные на рисунке, отражают зависимость скорости обработки запросов от их количества в пакете для рассмотренного выше примера. Первая из кривых соответствует строго последовательной обработке каждого отдельного запроса (т.е., без использования квадратных скобок в конвейерной строке), а вторая – параллельной обработке. Другими словами, вторая кривая отражает эффект от сочетания двух уровней распараллеливания: многопоточное выполнение запросов в пакете дополняется параллельным выполнением каждого отдельного запроса. Скорость обработки для обеих кривых вычислялась, как частное от деления числа запросов в пакете на полное время его выполнения.

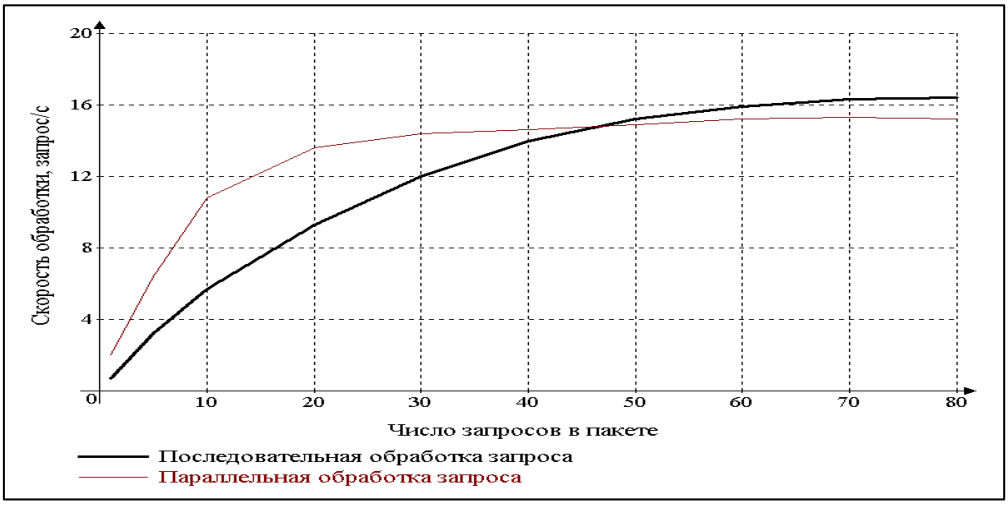

Рис. 3. Скорость обработки пакетов одновременных запросов

Как видно из рисунка, обе кривые ведут себя в целом одинаково. При увеличении числа запросов в пакете растет и скорость обработки, что объясняется положительным эффектом от многопоточной обработки запросов в сервере PMS. Например, при 10-ти запросах в пакете скорость обработки достигает 5.7 запросов в секунду в случае последовательного выполнения отдельного запроса и 10.8 – в случае параллельного. При 20-ти запросах в пакете значения производительности составляют 9.3 и 13.6 соответственно. При дальнейшем увеличении размеров пакета рост скорости обработки замедляется, а потом и вовсе останавливается вследствие достижения предельной производительности.

Однако, на этом сходство в поведении кривых заканчивается. Как видно из рисунка, при небольших размерах пакета параллельная обработка запроса дает значительный эффект в сравнении с последовательной, но по мере увеличения размера этот эффект уменьшается, а предельное значение у второй кривой оказывается даже меньшим, чем у первой. По-видимому, это объясняется тем, что при большом числе одновременных запросов накладные расходы на управление параллельными нитями и издержки конкуренции между ними оказываются слишком высокими. В этих условиях механизмы распараллеливания обработки запроса не приводят к росту производительности, но становятся скорее лишним обременением для сервера.

На рис. 4 приведены результаты измерения еще одного важного показателя – среднего времени выполнения запроса в пакете. Кривые, представленные на рисунке, отражают зависимость среднего времени обработки запроса от их количества в пакете для рассмотренного выше примера. Так же, как и на рис. 3, первая из кривых соответствует строго последовательной обработке каждого отдельного запроса (т.е., без использования квадратных скобок в конвейерной строке), а вторая – параллельной обработке. В отличие от рис. 3 обе кривые демонстрируют устойчивый и даже ускоренный рост. Тем не менее оба рисунка отражают одну и ту же характерную тенденцию: значительный выигрыш в

скорости от распараллеливания, наблюдаемый при небольших размерах пакета, постепенно исчезает по мере повышения нагрузки на сервер.

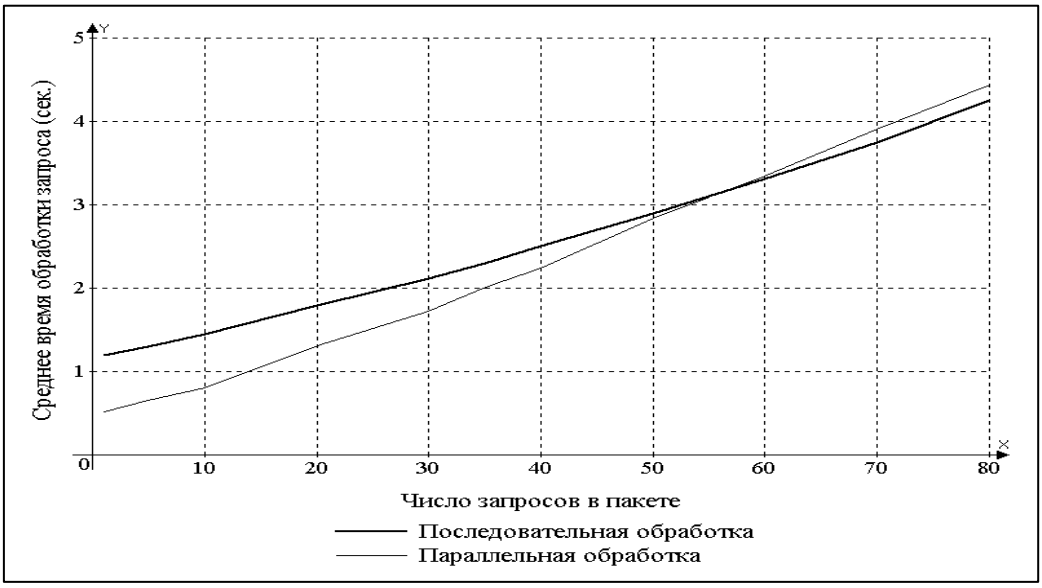

Рис. 4. Среднее время обработки запроса в пакете

Заключение

Хотя приведенные оценки относятся к конкретному примеру, они позволяют сформулировать следующие выводы.

- Выигрыш в быстродействии от применения параллельной обработки в новой версии PMS может быть весьма значительным (т.е. измеряться в «разах», а не в процентах).
- Применение средств параллельной обработки в целом не разрушает положительного эффекта от многопоточной обработки пакетов запросов, но способно усилить его.
- Как в режиме одиночных запросов, так и в режиме пакетов запросов средства параллельной обработки в новой версии PMS могут позволить значительно повысить скорость обработки в мульти-серверной среде. Однако, при повышении нагрузки на сервер PMS эффект от применения этих средств уменьшается и может вовсе исчезнуть (см. рис. 3 и 4).

Литература

1. *Уайт Т.* Hadoop: Подробное руководство. – СПб.: Питер, 2013. – 672 с.
2. *Любанович Б.* Простой Python. Современный стиль программирования. – СПб.: Питер, 2016. – 480 с.
3. *Klymash M., Hordiichuk-Bublivska O., Tchaikovskyi I., Deschynskiy Y.* Modeling and Research of Processing Big Data Sets // Proceedings of 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering, TCSET 2020. – Lviv-Slavske: IEEE, 2020. – P. 858-863.
4. *Козлов А.Д., Орлов В.Л.* Методы и средства обеспечения информационной безопасности распределенных корпоративных систем. – М.: ИПУ РАН, 2017. – 156 с.
5. *Жаранова А.О., Птицына Л.К.* Анализ влияния распределенности на качество функционирования комплексных систем защиты информации // Актуальные проблемы инфотелекоммуникаций в науке и образовании (АПИНО 2020): Сборник научных статей IX Международной научно-технической и научно-методической конференции. – СПб: СПБГУТ, 2020. – С. 324-327.
6. *Згоба А.И., Маркелов Д.В., Смирнов П.И.* Кибербезопасность: угрозы, вызовы, решения / Вопросы кибербезопасности, № 5, 2014, С. 30 – 38.
7. *Шапошников И.В.* Web-сервисы Microsoft .NET. – СПб: БХВ-Петербург, 2002. – 336 с.
8. *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.
9. *Асратян Р.Э.* Интернет-служба защищенной обработки информационных запросов в распределенных системах // Программная инженерия, № 11, 2016, С. 490 – 497.
10. Артемов И.Ю. Использование параллельной обработки данных для оптимизации работы программного обеспечения // Программные продукты и системы. – 2020. – № 3. – С. 471-475.
11. Golubtsov P. Scalability and Parallelization of Sequential Processing: Big Data Demands and Information Algebras // Advances in Intelligent Systems and Computing. – 2020. – Vol. 1127 AISC. – P. 274-298.
12. *Хант К.* TCP/IP. Сетевое администрирование. – СПб.: Питер, 2007. – 816 с.