УДК 004.738

ОРГАНИЗАЦИЯ КОНВЕЙЕРНОЙ ПРОГРАММНОЙ ОБРАБОТКИ В МУЛЬТИСЕРВЕРНОЙ СЕТЕВОЙ СРЕДЕ

Р.Э. Асратян

Рассмотрены принципы организации Интернет-службы, ориентированной на поддержку распределенных приложений и систем и представляющую собой своего рода «сетевое обобщение» известного механизма программного конвейера, позволяющее распространить его применение на мультисерверную среду. Отмечено, что характерная особенность новой службы заключается в возможности одновременного запуска нескольких удаленных исполняемых модулей на различных сетевых узлах с обеспечением каналов обмена данными между ними.

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

ВВЕДЕНИЕ

Механизм программного «трубопровода» или «конвейера» (pipeline), впервые реализованный в системе UNIX, остается одним из основных средств соединения возможностей нескольких программ для решения общей задачи [1, 2] во всех основных платформах. В большинстве командных процессоров этот механизм представляется в форме составной командной строки вида

$$\mathbf{Cmd_1 \mid Cmd_2 \mid \ldots \mid Cmd_n},$$

в которой каждый компонент $\mathrm{Cmd_i}$ представляет собой команду вызова какого-то локального модуля-обработчика. Предполагается, что стандартный вывод $\mathbf{Cmd}_i$ соединяется со стандартным вводом $\mathbf{Cmd}_{i+1}$ ($i = 1, 2, \ldots, n - 1$). На поведение каждого обработчика накладывается единственное ограничение: он должен читать входные данные со стандартного ввода, писать выходные данные в стандартный вывод и заканчивать свою работу после закрытия стандартного ввода.

Главный недостаток механизма трубопровода заключается в его локальности: все модули-обработчики должны выполняться на одном компьютере. Это ограничение не позволяет использовать его для организации распределенной обработки [3] в мультисерверной сетевой среде (в частности, в распределенных управляющих системах).

В настоящей работе рассматривается «сетевое обобщение» механизма трубопровода, предназначенное именно для этой цели и позволяющее задавать последовательность модулей-обработчиков в форме

$$\mathbf{Имя\_сервера_1@\ Cmd_1,\ Имя\_сервера_2@Cmd_2,\ \ldots,\ Имя\_сервера_n@Cmd_n},$$

в которой каждый компонент последовательности содержит не только команду запуска модуля-обработчика, но и сетевое имя сервера, на котором он должен быть запущен. Более того, каждая команда тоже может представлять собой заключенную в скобки «вложенную» последовательность команд, также содержащую имена серверов и модулей для описания последовательного межсерверного взаимодействия. Другими словами, предлагаемый подход позволяет организовать выполнение произвольного множества программ-обработчиков в произвольной цепочке серверов с организацией взаимодействия через стандартные вводы-выводы одним простым вызовом из программы-клиента. В частности, он допускает использование разработанных в течение десятилетий хорошо известных программ (таких, например, как grep, sed, md5sum, sort и многих других) в сетевой мультисерверной мультиплатформенной среде без какой-либо адаптации. Наиболее известные сетевые технологии, в том числе завоевавшие высокую популярность у разработчиков в последние годы, WCF [4], WebSocket [5] или системы организации распределенных кластерных вычислений типа Hadoop [6] не предоставляют готового решения этой задачи.

Известно, что такое решение, в принципе, может быть получено путем использования службы сетевого командного интерпретатора SSH [7] в

обычном программном конвейере для создания сетевых каналов взаимодействия. Однако, как показывает практика, на организацию взаимодействия в цепочке из 2—3 серверов при этом подходе может потребоваться несколько секунд. Такая задержка чаще всего является вполне комфортной при работе «с клавиатуры», но может значительно затруднить применение этого решения для организации взаимодействия между программными компонентами в распределенных системах.

В отличие от традиционного механизма трубопровода, опирающегося на локальный механизм «неименованного канала» pipe [1, 2], описываемый подход опирается на специальную сетевую службу, позволяющую организовать сетевой обмен между стандартными выходами и стандартными входами программ, выполняющихся на разных серверах с достаточно высокой скоростью (сетевые задержки обычно измеряются долями секунд, но не секундами, как в упомянутом решении). Важное свойство службы заключается в наличии удобного программного интерфейса, позволяющего легко организовать межсерверную конвейерную обработку из любой программы (в том числе, из программных компонентов распределенных систем) без использования командного процессора.

1. ПРИНЦИПЫ ОРГАНИЗАЦИИ МУЛЬТИСЕРВЕРНОГО КОНВЕЙЕРА

Интернет-служба RECS (Remote Executables Call Service — служба удаленного вызова исполняемых модулей) [8] основана на той же идее межпроцессного взаимодействия через стандартные вводы и выводы, что и общепринятый одномашинный программный конвейер, но со следующими важными отличиями.

- RECS является сетевой службой. Для организации трубопровода программа-клиент обращается к обслуживающему ее RECS-серверу и передает ему составную командную строку вида **Command$_1$, Command$_2$, ..., Command$_n$** для исполнения.
- Каждый компонент **Command$_i$** представляет собой или просто команду запуска исполняемого модуля на обслуживающем сервере или конструкцию, называемую «вложенной командной строкой». Эта конструкция имеет вид «имя_сервера@команда» или даже «имя_сервера@(команда$_1$, команда$_2$, ..., команда$_n$)» и предполагает запуск одного или нескольких исполняемых модулей на удаленном сервере с заданным Интернет-именем. При этом каждая «команда» может, в свою очередь, представлять собой такую же конструкцию с именем другого сервера и списком команд и т. д.

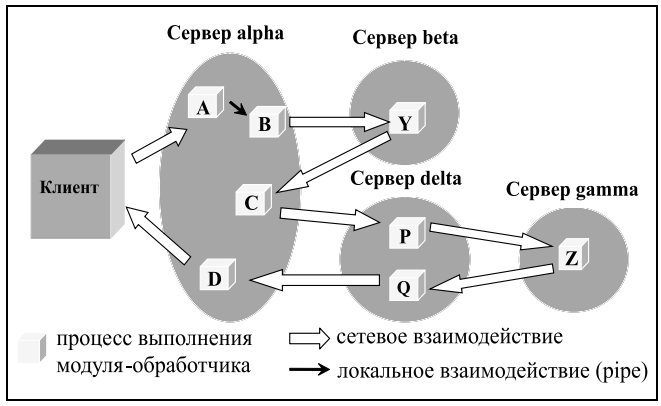


Рис. 1. Пример структуры конвейерных соединений

Другими словами, RECS позволяет программе-клиенту осуществить одновременный вызов сразу нескольких исполняемых модулей (обработчиков), расположенных на разных сетевых узлах. В момент вызова непосредственно обслуживающий клиента RECS-сервер организует запуск указанных обработчиков на собственном сетевом узле и/или на удаленных узлах (разумеется, также оснащенных RECS-серверами). В тот же момент усилиями всех задействованных серверов создается множество «каналов», попарно соединяющих запущенные обработчики по принципу «стандартный вывод предыдущего со стандартным вводом следующего». Конечно же, стандартный ввод первого обработчика и стандартный вывод последнего соединяются с клиентом. С этого момента все отправленные клиентом данные будут проходить последовательную обработку во всех запущенных исполняемых модулях, а результирующие данные будут немедленно направляться клиенту по мере их появления. На содержание данных (текстовые или двоичные) и последовательность их отправки и приема не накладывается никаких ограничений (хотя, разумеется, поведение клиентов должно быть логически согласовано с поведением обработчиков). На рис. 1 проиллюстрирована структура информационных связей в мультисерверной среде при обработке командной строки

A, B, beta@Y, C, delta@(P, gamma@Z, Q), D.

Предполагается, что сервер alpha является обслуживающим клиента сервером, а прописные латинские буквы являются именами исполняемых модулей. Важно отметить, что в данном примере серверы beta и/или gamma могли бы быть недоступны непосредственно для клиента. Например, клиент мог бы быть расположен в частной сети организации, а указанные серверы — вне нее. В этом случае серверу alpha фактически выпала бы роль сервера-посредника, расположенного на границе двух сетей.

Понятие обобщенного «конвейерного соединения», связывающего обработчиков друг с другом (а также крайних обработчиков с клиентом) является основополагающим для RECS. Хотя для его организации в разных ситуациях используются разные технологии — «трубопровод» (pipe), если соединяемые обработчики находятся на одном узле, и межсерверное сетевое взаимодействие, если на разных — у пользователя создается иллюзия единого связующего механизма, достаточно ясного и простого в обращении.

2. КОРОТКИЙ ПРИМЕР ПРИМЕНЕНИЯ RECS

Как и всякая сетевая служба, основанная на протоколах TCP/IP [9], RECS поддержана клиентским и серверным программным обеспечением (ПО). Сервер RECS представляет собой постоянно активную программу, обслуживающую запросы на обработку от клиентов или других серверов. Клиентское ПО представляет собой библиотеку функций, реализующих прикладной программный интерфейс (API) к RECS. Этот интерфейс служит «лицом» RECS с точки зрения пользователя.

Рассмотрим короткий пример кода на языке C++, в котором клиентская программа обращается к обслуживающему серверу на платформе win32 для вычисления контрольной суммы всех исполняемых файлов (файлов типа «exe») в каталогах C:\WINDOWS\system32 и C:\WINDOWS\system на этом сервере (например, с целью проверки, изменились ли эти файлы в каталогах или нет). Предположим, что на этом сервере имеется обработчик dirlist.exe, который получает на стандартный ввод имя каталога и записывает в стандартный вывод построчный список имен всех содержащихся в нем файлов и их контрольных сумм, причем контрольная сумма каждого файла и его имя выводятся в отдельной строке. Для вычисления общей контрольной суммы только исполняемых файлов программа привлекает еще один сервер (с именем unode) на платформе UNIX или Linux, чтобы использовать уже имеющиеся в нем широко известные программы-обработчики grep и sort, обеспечивающие фильтрацию и сортировку полученного списка соответственно. Формирование окончательной 32-байтовой контрольной суммы от этого списка (с отправкой результата в стандартный вывод) реализуется с помощью известной программы md5sum.exe снова на обслуживающем сервере (разумеется, мы предполагаем, что оба сервера оснащены серверным ПО RECS):

```
char Checksum [32];
RECSclnt Rcln;
Rcln.Config("alpha", 8130,"alibaba","sezam");
Rcln.Connect();
Rcln.Process("dirlist.exe, unode$(grep.exe$,sort),
   md5sum.exe");
Rcln<<"c:\\windows\\system";
Rcln.GetLine(Checksum, 32);
cout<<Checksum<<"\n"<<Rcln.GetSysMsg()<<"\n";
Rcln<<"c:\\windows\\system32";
Rcln.GetLine(Checksum, 32);
cout<<Checksum<<"\n"<<Rcln.GetSysMsg()<<"\n";
Rcln.Quit(); Rcln.Disconnect();
```

Первые две строки определяют две переменные: строковую переменную и объект класса RECSclnt. Этот класс содержит в себе все методы, необходимые клиенту для работы с RECS. Третья строка (метод Config) настраивает объект на работу с определенным RECS-сервером. В качестве параметров объекту передаются IP-адрес (или Интернет-имя) сервера, номер порта, имя и пароль пользователя (RECS-сервер обслуживает только зарегистрированных пользователей).

Четвертая строка устанавливает TCP-соединение с сервером, а пятая запускает три удаленных обработчика (с помощью метода Process). Первый из них (dirlist.exe) запускается непосредственно на обслуживающем сервере (alpha), два следующих (grep и sort) — на сервере unode, а последний (md5sum) — снова на сервере alpha. Сразу после запуска все серверы переходят к ожиданию появления данных на их стандартных вводах.

Шестая, седьмая и восьмая строки обеспечивают передачу на сервер имени первого каталога, считывание результата обработки (метод GetLine) и вывод на экран полученной контрольной суммы вместе с итоговым диагностическим сообщением сервера.

Три последующие строки повторяют операцию вычисления контрольной суммы для второго каталога.

Предпоследняя строка извещает сервер о завершении диалога (он мог бы продолжаться неограниченно), а последняя прекращает сетевое соединение. Метод Quit вызывает последовательное закрытие стандартного ввода во всех обработчиках и завершение их работы.

Разумеется, из данного фрагмента кода намеренно удалены операторы проверки успеха выполнения методов и обработки исключений.

3. СОЗДАНИЕ КОНВЕЙЕРНЫХ СОЕДИНЕНИЙ

Как видно из приведенного примера, сеанс связи в RECS включает в себя три основные фазы:

— соединение с обслуживающим сервером и передача ему командной строки, включающей в себя имена серверов и модулей-обработчиков (метод Process);

— отправку данных для обработки на удаленных серверах и получение результатов обработки;

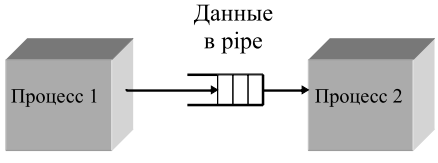
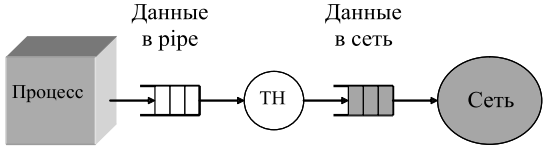
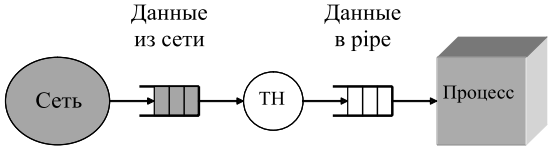
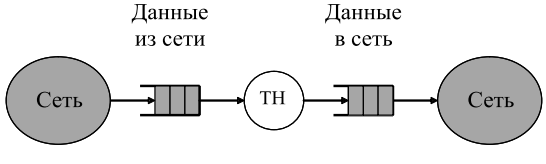
— завершение сеанса связи.

Первая фаза наиболее сложная. Она включает последовательный запуск модулей-обработчиков не только в обслуживающем сервере, но и на всех серверах, поименованных в командной строке, вместе с созданием специальных каналов обмена данными между ними — конвейерных соединений.

Обработка метода Process 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). Если первый из типов представляет собой обычный локальный канал pipe, соединяющий два вычислительных процесса, то все последующие включают в себя по два канала (входной и выходной), в число которых обязательно входят TCP-каналы, ассоциированные с открытыми сетевыми соединениями. Важным компонентом этих типов конвейерных соединений является транспортная программная нить (ТН), обеспечивающая постоянный перенос данных из входного канала в выходной (по мере их появления) и закрытие выходного канала после закрытия входного. При запуске ТН получает в качестве аргументов два системных дескриптора. Каждый из них может представлять собой дескриптор чтения или записи канала pipe или же дескриптор сетевого сокета [9] (но с учетом того, что перенос данных из одного канала pipe в другой — бессмысленная операция). Описываемый подход опирается на определенное сходство между двумя типами каналов: несмотря на совершенно разную физическую природу, оба представляют собой буферизованные байтовые каналы обмена, функционирующие по принципу FIFO (first in first out). Вместе с тем, он учитывает и принципиальное различие между ними: в отличие от дескрипторов канала pipe, дескриптор сетевого сокета не может быть непосредственно связан со стандартным вводом или выводом процесса.

Обработки метода Process в сервере RECS включает в себя прием командной строки по TCP-соединению с клиентом и последовательной обработки всех ее компонентов по приведенному ниже алгоритму. Предполагается, что AccessSocket — индекс сокета для соединения с клиентом (т. е. сокета, полученного с помощью системного вызова accept), переменная RemoteSocket используется для хранения индекса последнего сокета, открытого для связи с удаленным сервером, а переменная ReadPipe — для хранения дескриптора чтения последнего созданного канала pipe.

Типы конвейерных соединений

Тип конвейерного соединения	Принцип организации	Пояснение
Pipe		Общепринятый трубопровод для локального взаимодействия на одном сервере. Один из процессов использует pipe в качестве стандартного вывода, а другой — в качестве источника для стандартного ввода
Pipe-Socket		Отправка данных со стандартного вывода процесса в сеть (для передачи на стандартный ввод удаленного процесса). Транспортная программная нить *ТН* обеспечивает непрерывное чтение данных из выходного дескриптора pipe-а и запись их в сетевой сокет, соединенный с удаленным сервером
Socket-Pipe		Прием данных из сети для передачи на стандартный ввод процесса. Транспортная программная нить *ТН* обеспечивает непрерывное чтение данных из сокета, соединенного с удаленным сервером и запись их во входной дескриптор pipe-а, соединенного со стандартным вводом процесса-получателя
Socket-Socket		Транзитная передача данных через сервер. Транспортная программная нить *ТН* обеспечивает непрерывное чтение данных из сокета, соединенного с одним удаленным сервером и запись их в сокет, соединенный с другим

Шаг 1. Если очередной компонент представляет собой вложенную командную строку (т. е. обращением к удаленному серверу), то перейти к шагу 2, в противном случае — к шагу 3.

Шаг 2. Выделение имени удаленного сервера RECS из вложенной командной строки. Создание нового сокета для связи с удаленным сервером и установление сетевого соединения с ним. В случае невозможности соединения перейти к шагу 5. Передача удаленному серверу RECS вложенной командной строки по данному соединению (для этого фактически применяется тот же самый клиентский метод Process). Получение сообщения о завершении обработки вложенной командной строки от удаленного сервера. Если от удаленного сервера получено сообщение об ошибке, перейти к шагу 5. Создание конвейерного соединения по правилам:

— если компонент первый в командной строке, то создание соединения типа «Socket-Socket» с использованием дескриптора AccessSocket и дескриптора нового сокета;

— если предыдущий компонент в командной строке является вложенной командной строкой, то создание соединения типа «Socket-Socket» с использованием дескриптора RemoteSocket и дескриптора нового сокета;

— если предыдущий компонент в командной строке не является вложенной командной строкой, то создание соединения типа «Pipe-Socket» с использованием дескриптора ReadPipe и дескриптора нового сокета.

Занесение дескриптора нового сокета в переменную RemoteSocket и переход к шагу 4.

Шаг 3. Запуск процесса выполнения модуля-обработчика, соответствующего очередному компоненту. В случае невозможности запуска модуля перейти к шагу 5. Создание конвейерного соединения по правилам:

— если компонент первый в командной строке, то создание неименованного канала pipe (с соединением его дескриптора чтения со стандартным вводом запущенного процесса) и создание соединения типа «Socket-Pipe» с использованием дескриптора AccessSocket и дескриптора записи этого канала pipe;

— если предыдущий компонент в командной строке является вложенной командной строкой, то создание неименованного канала pipe (с соединением его дескриптора чтения со стандартным вводом запущенного процесса) и создание соединения типа «Socket-Pipe» с использованием дескриптора RemoteSocket и дескриптора записи этого канала pipe;

— если предыдущий компонент в командной строке не является вложенной командной строкой, то создание соединения типа «Pipe» путем соединения дескриптора ReadPipe со стандартным вводом запущенного процесса (общепринятое соединение двух процессов через pipe).

Создание неименованного канала pipe с соединением его дескриптора записи со стандартным выводом запущенного процесса и занесением его дескриптора чтения в переменную ReadPipe.

Шаг 4. Если очередной компонент не последний в командной строке, то выбрать следующий компонент в качестве очередного компонента и вернуться к шагу 1. Если очередной компонент является вложенной командной строкой, то создание завершающего конвейерного соединения типа «Socket-Socket» с использованием дескриптора RemoteSocket и дескриптора AccessSocket. В противном случае — создание завершающего конвейерного соединения типа «Pipe-Socket» с использованием дескриптора ReadPipe и дескриптора AccessSocket. Запись сообщения об успешном завершении обработки метод Process («OK») в сокет с дескриптором AccessSocket.

Шаг 5. Аварийное завершение обработки. Прекращение работы всех запущенных модулей-обработчиков, закрытие всех каналов pipe и сетевых соединений. Запись сообщения об ошибке в сокет с дескриптором AccessSocket.

Важно отметить, что обработка вложенной командной строки в удаленном сервере RECS (шаг 2 рассмотренного алгоритма) осуществляется в точности по этому же алгоритму с построением конвейерных соединений, в которых задействуются сокеты, обеспечивающие сетевую связь между серверами.

Рассмотрим в качестве примера структуру конвейерных соединений, созданную для выполнения командной строки **A, B, beta@C** в сервере alpha на основе описанного алгоритма (рис. 2). Процессы выполнения модулей-обработчиков обозначены «кубиками». Как видно из рисунка, даже в таком простом примере задействованы все четыре типа соединений.

Конвейерное соединение (1) обеспечивает передачу всех данных, полученных по сети от клиента, в канал pipe, соединенный со стандартным вводом модуля-обработчика «A». Это соединение использует сокет, открытый при установлении сетевого соединения с клиентом (дескриптор AccessSocket). Конвейерное соединение (2) представляет собой обычную связь между стандартным выводом обработчика «A» и стандартным вводом обработчика «B» через неименованный канал pipe. Конвейерные соединения (3) и (4) обеспечивают передачу данных от обработчика «B» на сервере alpha к обработчику «C» на сервере beta. Оба соединения используют сокеты, открытые для межсерверного взаимодействия на серверах alpha и

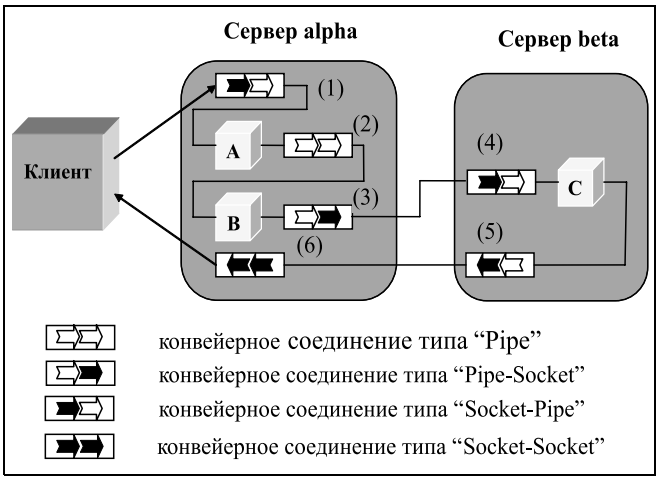


Рис. 2. Пример структуры конвейерных соединений *n*

beta соответственно. Наконец, соединения (5) и (6) обеспечивают возврат клиенту всех данных со стандартного вывода обработчика «C». Так как возврат данных непосредственно клиенту с сервера beta невозможен, в качестве посредника используется сервер alpha. Отметим, что конвейерные соединения (1) и (6) пользуются одним и тем же сокетом для чтения данных из сети и для записи данных в сеть соответственно. То же можно сказать и о соединениях (4) и (5), а также о соединениях (6) и (3).

Корректность обработки клиентских данных в последовательности обработчиков, запущенных в мультисерверной среде, целиком определяется точностью работы созданных конвейерных соединений. Если каждое конвейерное соединение, независимо от его типа, обеспечивает передачу всех байтов со своего входа на выход без потерь и нарушения очередности, то результат обработки будет таким же, как в случае запуска всех обработчиков на одном сервере.

ЗАКЛЮЧЕНИЕ

Как уже отмечалось, описанный подход базируется на определенном сходстве между локальным каналом pipe и сетевым TCP-каналом между удаленными программами: оба типа каналов реализуют буферизованные потоки данных, 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 фундаментальные различия в физической природе каналов оказались незаметны для пользователя.

Как видно из примера, приведенного в § 2, рассмотренная технология может быть применена в гетерогенной серверной среде: в настоящее время сервер RECS реализован на платформах Windows и Unix/Linux.

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 одномашинного конвейера).

Отметим, что описанный подход базируется на создании отдельного процесса (задачи) для запуска каждой из программ-обработчиков. К сожалению, использование программных потоков (нитей) для той же цели оказывается невозможным, так как все программные потоки в пределах одной задачи разделяют одни и те же стандартный ввод и стандартный вывод (здесь имеется прямая аналогия с технологией CGI, испытывающей то же ограничение). Отсюда вытекает существенный недостаток описанного подхода — проигрыш в скорости обработки технологиям типа WS и WCF, имеющим возможность опираться на программные нити. Этот проигрыш связан с накладными расходами на управление процессами на серверах и проявляется тем в большей степени, чем короче время выполнения информационного запроса. Поэтому область эффективного применения описанного подхода включает в себя взаимодействия, связанные с достаточно большими объемами передаваемой информации и с достаточно длительной обработкой (например, поиск информации в СУБД, удаленное формирование отчетов и т. п.).

ЛИТЕРАТУРА

1. *Келли-Бутл С.* Введение в Unix. — М.: ЛОРИ, 1995. — 596 с.
2. *Таненбаум Э.* Современные операционные системы. — СПб.: Питер, 2002. — 1040 с.
3. *Гофф М.* Сетевые распределенные вычисления: достижения и проблемы. — М.: КУДИЦ-ОБРАЗ, 2005. — 320 с.
4. *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.
5. *Wang V., Salim F., Moskovits P.* The definitive guide to HTML5 WebSocket. — N.-Y.: Apress, 2013. — 208 p.
6. *Уайт Т., Hadoop.* Подробное руководство. — СПб.: Питер, 2013. — 672 с.
7. *Хант К.* TCP/IP. Сетевое администрирование. — СПб.: Питер, 2007. — 816 с.
8. *Асратян Р.Э.* Интернет-служба для поддержки распределенных вычислений // Информационные технологии. — 2006. — № 12. — С. 60—66.
9. *Снейдер Й.* Эффективное программирование TCP/IP. Библиотека программиста. — СПб.: Символ-Плюс, 2002. — 320 с.

Статья представлена к публикации руководителем РРС В.Ю. Столбовым.

Асратян Рубен Эзрасович — канд. техн наук,
вед. науч. сотрудник, Институт проблем управления им. В.А. Трапезникова РАН, г. Москва, ✉ rea@ipu.ru.