УДК 004.415.53

К.А. Лаврентьев,

старший преподаватель кафедры информационных систем и технологий Хабаровского государственного университета экономики и права, серверный программист ООО «Интерсол»

МОДУЛЬ СИНХРОНИЗАЦИИ СТАРТА ЗАПРОСОВ ДЛЯ НАГРУЗОЧНОГО СТЕНДА

Нагрузочное тестирование информационных систем с использованием имитационных моделей требует реализации модуля распределения запросов на многие клиентские машины тестирующей системы. В статье рассматриваются способы обеспечения автоматического старта, описывается реализация такого старта с использованием одного из методов. Предлагается архитектура тестирующей системы и проводятся эксперименты для верификации предлагаемой архитектуры.

Ключевые слова: *сокет, служба, одновременный старт, асинхронность, потоки, транзакции, сервер, нагрузочное тестирование, производительность, моделирование нагрузки, распределение заданий.*

Load testing of information systems with simulation models requires the implementation of the module of requests distribution for many client machines of the testing system. The article describes the ways to ensure automatic start, describes the implementation of that start using one of the methods. The architecture of the testing system is proposed. Corresponding experiments are carried out.

Keywords: *socket, service, simultaneous start, asynchrony, threads, transactions, server, load testing, performance, load modeling, job distribution.*

Введение

Заключительным этапом разработки или модернизации многопользовательского клиент-серверного приложения является нагрузочное тестирование производительности. Главная цель такого тестирования – установление предельных значений параметров рабочей нагрузки на систему, приемлемой для предметной области, в которой эксплуатируется система. Среди множества этих параметров – пропускная способность, загруженность ресурсов и т.д., основным считается показатель времени отклика на запрос. Этот показатель также называется психологическим временем ожидания, и его величина обратно пропорциональна желанию пользователю работать с системой. Логично предположить, что значение этого показателя увеличивается вместе с увеличением нагрузки на систему. Для установления предельного времени отклика

разработанной системы в процессе нагрузочного тестирования систему подвергают нагрузкам с помощью отправки N-запросов и замеряют время отклика. Обычный состав тестового стенда – это сервер, который в зависимости от архитектуры разработанной системы может быть как сервером приложений, так и сервером базы данных (БД), и клиент, с которого происходит отправка запросов. В большинстве случаев клиент и сервер – это виртуальные машины. Для имитации рабочей нагрузки необходимо послать серверу определённое количество (N) запросов за t мс и найти среднее время ожидания каждого запроса.

Каждая многопользовательская информационная система (ИС) способна принять разное количество запросов в час, поэтому посчитать среднее количество можно только приблизительно. Например, Google и Yandex обрабатывают в час примерно 220 млн запросов [1]. Компании поменьше, например система заказа авиабилетов, обрабатывает примерно 70 млн запросов. Таким образом, можно сказать, что для имитации рабочей нагрузки современной ИС тестовый стенд должен сгенерировать и обработать не менее 50 млн запросов.

Целью проведения экспериментов на созданном стенде было получение информации о возможности отправки всего заданного количества запросов с клиента на сервер и по возможности получение среднего времени ожидания при заданной рабочей нагрузке. В результате проведения экспериментов были получены следующие статистические данные:

– архитектура тестового стенда не позволяет сгенерировать и отправить на сервер необходимое количество запросов (среднее количество сгенерированных запросов – 45 % от общего числа), это связано с тем, что клиент в представленном тестовом стенде один и ему не хватает оперативной памяти для обработки всех потоков запросов;

– количество запросов, поступивших на сервер, составляет в среднем 3,2 % от общего числа запросов. Причина этого автору видится в том, что у клиентского приложения существует буфер для хранения запросов перед отправкой, и в нём остаются неотправленные запросы после аварийного завершения программы из-за первой проблемы.

Таким образом, после проведения экспериментов можно обозначить проблему: невозможно адекватно провести имитацию рабочей нагрузки при использовании архитектуры тестового стенда с одним клиентом и одним сервером. Невозможность имитировать рабочую нагрузку, используя всего один клиент, связана с техническими ограничениями клиента. Имитация рабочей нагрузки состоит из запросов, которые отправляются серверу клиентом. Время выполнения каждого запроса делится на авторизацию,

отправку запроса, ожидание результата и получение результатов. Ожидание результата занимает большую часть времени выполнения запроса, и именно из-за этого необходимо внедрение в тестовый стенд нескольких клиентов. Внедрение нескольких клиентов необходимо потому, что один клиент в тестовом стенде не позволяет сгенерировать всю необходимую рабочую нагрузку из-за большого времени ожидания результата и нехватки оперативной памяти для обработки всех запросов. В случае добавления в тестовый стенд нескольких клиентов возникают следующие проблемы:

1) нужно обеспечивать одновременный старт запросов с нескольких клиентских машин.

2) необходимо продумать алгоритм распределения количества запросов на каждую клиентскую машину.

Данное исследование будет посвящено решению первой проблемы, так как она является первостепенной. Решение же второй проблемы – алгоритм распределения количества запросов, как и исследование причин того, почему количество поступивших запросов на сервер так мало, является темой отдельных работ и требует проведения множества экспериментов.

Основная часть

Для проведения эффективного нагрузочного тестирования необходимо имитировать реальную нагрузку на сервер информационной системы. Как было сказано выше, для объективной имитации необходимо обеспечить в тестовом стенде несколько клиентских машин, которые будут моделировать нагрузку на сервер [2]. Для управления стартом запросов на сервере необходим единый планировщик старта запросов. Задача этого планировщика – обеспечить старт запросов на каждой клиентской машине в заданное время начала эксперимента. Такой планировщик может быть реализован с помощью разных технологий и с применением различных моделей. В работе А.С. Хританкова «Модели и алгоритмы распределения нагрузки» [3] рассматриваются различные модели реализации единого планировщика старта запросов. В работе [3] автор приходит к выводу, что наиболее эффективным алгоритмом для распределения нагрузки являются системы массового обслуживания, что подтверждает выбор СМО для тестирования производительности системы в данном исследовании. В работе А.Ю. Быкова «Задача распределения заданий между клиентом и сервером в имитационном моделировании на основе технологии облачных вычислений и результаты экспериментов по её решению» [4] автор описывает постановку задачи распределения заданий (запросов) при моделировании нагрузки на сервер. В работе [5] исследуется проблема балансировки нагрузки между серверами и делается вывод о необходимости наличия в системе единого планировщика для балансировки нагрузки.

Для автоматического старта запросов на клиентских машинах существует несколько способов:

1) отправка каждые N секунд запроса на сервер с клиента, который вернёт информацию о том, нужно ли начинать отправку запросов;

2) установка в параметрах сервера времени и параметров эксперимента и отправка этих данных клиенту при его первом обращении к серверу;

3) использование специальных объектов – сокетов. Сокет – это программный интерфейс, служащий для обмена данными между процессами, причем процессы могут находиться как на одной машине, так и на распределённых по сети машинах.

Первый способ является крайне затратным как по нагрузке на сеть, так и на процессор сервера. На данный момент развития ИС не имеет смысла использовать этот способ, так как по большей части все клиент-серверные ИС работают по принципу отсоединённого клиента, а такой способ не укладывается в данную концепцию. Кроме того, для реализации этого способа нужно хранить параметры эксперимента на каждой клиентской машине, что ещё больше усложняет схему тестового стенда.

Второй способ укладывается в концепцию отсоединённого клиента, но при этом имеет низкую надёжность: существует вероятность, что в заданный момент старта эксперимента сервер будет недоступен или занят и не сможет обработать запросы от клиентов.

Третий способ тоже укладывается в концепцию отсоединённого клиента и при этом позволяет следить за стартом и выполнением эксперимента в реальном времени, а значит, имеет высокую надёжность по сравнению со вторым способом. Сравнительный анализ каждого из способов решения задачи представлен в таблице 1.

Таблица 1 – Сравнительный анализ способов решения задачи

Способ / Критерий	**Производительность**	**Надёжность**	**Особенности реализации**
1-й способ	Постоянная отправка запросов на разрешение начать тестирование сильно нагружает сеть и процессор сервера. Производительность тестирующей системы снижается на 3 % для каждого клиента	При реализации автоматического старта с помощью этого способа можно быть уверенным, что клиенту поступит информация о старте запросов. Данный способ обеспечивает должную надёжность автоматического старта	Реализация автоматического старта запросов с помощью этого способа требует хранения параметров экспериментов на каждой клиентской машине. При изменении параметров необходимо будет поменять их на каждой клиентской машине

Продолжение таблицы 1

2-й способ	Реализация автоматического старта запросов с использованием этого способа никак не влияет на производительность тестирующей системы	При реализации автоматического старта с помощью этого способа нельзя быть уверенным, что клиент начнёт отправку запросов в назначенное время. Клиент может аварийно завершить работу, сломается сеть и т.д. Данный способ не обеспечивает надёжность автоматического старта	Реализация автоматического старта запросов с помощью этого способа не влечёт никаких особенностей. При первом запросе клиента нужно отправить ему вместе с информацией об успешном подключении информацию о времени старта запросов
3-й способ	Так как требуется выделить память и процессорное время для обработчика сокетов, то производительность тестирующей системы снизится на 3 % по сравнению с системой без поддержки автоматического старта запросов	При реализации автоматического старта с помощью этого способа можно быть уверенным в старте запросов на каждом клиенте, так как с каждым клиентом поддерживается фоновое соединение по запросу. Данный способ обеспечивает должную надёжность автоматического старта	Реализация автоматического старта запросов с помощью этого способа требует разработки специального серверного модуля – обработчика сокетов и клиентского классашлюза для работы с обработчиком

Таким образом, после проведения сравнительного анализа всех методов обеспечения автоматического старта запросов можно сказать, что самым оптимальным является третий способ: использование специальных объектов – сокетов. Чтобы подтвердить это утверждение, необходимо провести имитационные эксперименты, используя для одновременного старта при имитации нагрузки каждый из способов. Целью проведения экспериментов будет оценка погрешности одновременного старта запросов. Такая погрешность будет у каждого способа, это связано с техническими ограничениями системы: передача данных по сети, открытие форм тестирования и т.д. Эксперимент можно считать удачным и способ одновременного старта запросов оптимальным, если погрешность старта запросов с использованием способа составит не более 1 мс и будет минимальной. Такое

значение погрешности связано с тем, что необходимо обеспечить одновременный старт запросов на каждой клиентской машине. Погрешность в 1мс покрывает временные издержки процессора и клиентской операционной системы на получение сигнала от обработчика сокетов и запуск старт запросов.

Тестовый стенд, который имитирует рабочую нагрузку и использует первый способ для одновременного старта, представляет собой сервер и множество клиентов. Так же, как и в стенде с одним клиентом, происходит обычная отправка запросов на сервер, но в этом случае каждые N-секунд посылается специальный запрос для определения возможности начать тестирование.

Тестовый стенд, использующий второй способ для одновременного старта запросов, – это тот же сервер и множество клиентов, которые посылают серверу запросы. Только в этом случае при обращении клиента к серверу (например, при авторизации на сервере БД) клиенту передаются параметры старта запросов.

В тестовом стенде, построенном с использованием сокетов, также существует множество клиентов и сервер, кроме того, есть ещё специальный веб-сервер – обработчик сокетов. В задачи этого обработчика входит хранение подключений клиентов и отправка им запросов от сервера. Краткий алгоритм работы автоматического старта запросов с использованием сокетов выглядит следующим образом:

– клиент подключается к веб-серверу (обработчику сокетов) и получает в ответ информацию о том, что он успешно зарегистрирован. Обработчик сокетов в это время сохраняет в своей памяти уникальный id клиента и его адрес в сети. После этого подключение между клиентом и обработчиком сокета замораживается и не использует ресурсы системы;

– при возникновении запроса к клиенту (например, на сервере наступило событие, которое генерирует старт запросов) сервер передаёт эту информацию обработчику сокетов, а тот, в свою очередь, отправляет её всем зарегистрированным клиентам;

– клиенты получают от обработчика сокетов информацию о старте запросов и начинают эксперимент. При этом во время проведения эксперимента каждый клиент может обмениваться информацией с обработчиком сокетов и, таким образом, можно осуществлять логирование экспериментов на сервере.

Схема тестового стенда, построенного с использованием сокетов, показана на рисунке 1.

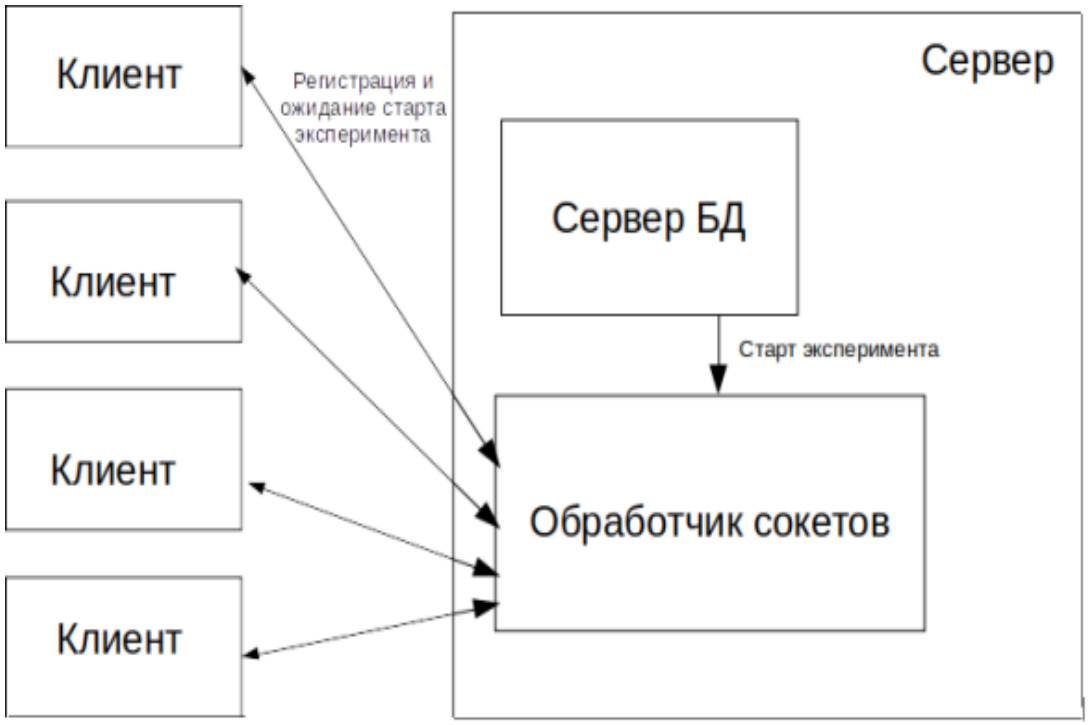


Рисунок 1 – Схема тестового стенда, построенного с использованием сокетов

Для реализации хранилища сокетов подойдет любой веб-ориентированный язык программирования и любой веб-сервер. В данной работе в качестве серверной операционной системы используется Microsoft Windows, значит, в качестве веб-сервера используется IIS [6]. Для разработки хранилища сокетов используется язык javascript с применением фреймворка Express (Node.JS). Выбор этого языка и фреймворка связан с его простой и удобством разработки именно таких приложений: для реализации хранилища сокетов у Node.JS есть библиотека socket.io [7], которая позволяет выполнять все действия по регистрации и отправке сообщений клиентам. Отправка сообщений клиентам происходит по протоколу HTTP в JSON-формате, для того чтобы клиент смог зарегистрироваться у обработчика сокетов и принять сообщение, необходимо написать специальный шлюз. Этот шлюз является обычным классом в клиентском приложении и должен быть написан на языке, на котором написан клиент. В данной работе клиент написан на языке C#, для реализации класса-шлюза используется библиотека WebSocketSharp [8].

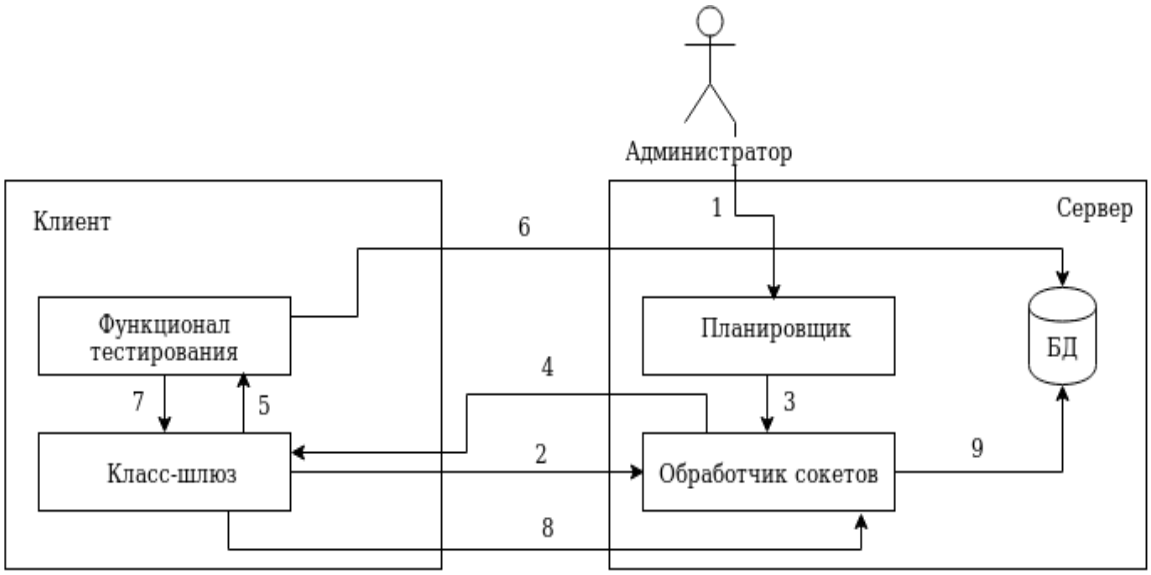
Класс-шлюз должен работать в фоновом режиме, экземпляр этого класса создаётся при запуске программы и в дальнейшем ожидает команд от обработчика сокетов. Все методы этого класса работают в отдельных потоках так как только таким образом можно обеспечить фоно-

вый режим работы. Данный класс-шлюз имеет три открытых метода, это необходимое количество методов для поддержки сокетов в клиентской программе.

Первый метод (Execute) служит для инициализации слушателя сокетов в клиентской программе и инициализации переменных класса. При инициализации слушателя сокетов инициализируются методы открытия и закрытия соединения с обработчиком сокетов и метод получения сообщения от обработчика сокетов.

Второй метод класса (PrepareMessageForParse) нужен для парсинга, пришедшего от обработчика сокетов сообщения и выполнения необходимых действий, в данном случае старта запросов. Этот метод принимает в качестве входного параметра сообщение (message), пришедшее от обработчика сокетов и извлекает из сообщения информацию о том, нужно ли осуществить старт запросов или нет. Результатом работы этого метода будет логическая переменная.

Третий метод (SendLogableMessage) необходим для получения от тестирующего модуля сообщений о каком-либо событии (ошибка, успешное завершение запроса и т.д.) и отправки этого сообщения обработчику сокетов. Этот метод принимает два параметра – сообщение (message) и тип (logableType). Схема работы класса-шлюза представлена на рисунке 2.



1 - Установка параметров эксперимента
2 - Регистрация клиента
3 - Команда от планировщика на старт запросов
4 - Отправка клиенту команды на старт запросов
5 - Команда классу для тестирования на начало эксперимента
6 - Отправка запросов серверу БД
7 - Логирующее сообщение о статусе отправки запроса
8 - Отправка логирующего сообщения классу-шлюзу
9 - Запись логирующего сообщения в БД

Рисунок 2 – Схема работы класса-шлюза

Для корректной имитации автоматического старта запросов в клиентском приложении с использованием класса-шлюза необходимо обеспечить запуск тестирующего приложения в режиме сервиса (daemon в unix-системах). Сервис (служба) в ОС Windows – это приложение, автоматически запускаемое при старте системы и работающее в фоновом режиме. Для корректной имитации автоматического старта запросов необходимо запускать тестирующее приложение именно в режиме сервиса, чтобы предотвратить использование ресурсов системы во время простоя тестирующего приложения. При запуске тестирующего приложения в режиме сервиса возникает проблема контроля отправки запросов. Связана эта проблема с тем, что программа, запущенная в режиме сервиса, работает в фоновом режиме, а значит, не взаимодействует с графическим интерфейсом пользователя, и невозможно визуально оценить и контролировать работу тестирующего приложения. Для решения этой проблемы нужно доработать класс-шлюз и обработчик сокетов на приём и запись логирующих сообщений. При возникновении какого-либо события на клиенте (старта запросов, окончание отправки запросов, ошибка и т.д.) класс-шлюз отправляет сообщение об этом событии обработчику сокетов. Обработчик сокетов, в свою очередь, записывает полученное сообщение в БД. Таким образом, на сервере станет храниться лог работы всех клиентских приложений и будет возможность контроля проведения тестов. Было проведено 250 экспериментов с использованием 3 клиентских машин (1 виртуальная и 2 физические) и одним сервером баз данных. Результаты экспериментов для каждого способа автоматического старта запросов представлены в таблице 2.

Таблица 2 – Результаты экспериментов по определению оптимального способа автоматического старта

	1-й способ	**2-й способ**	**3-й способ**
Погрешность старта, мс	1,4	1,2	1,2

Как видно из результатов, минимальные значения погрешности имеют второй и третий способы, это связано с тем, что они очень похожи и имеют сходный способ работы. Даже при одинаковых значениях погрешности для реализации одновременного старта запросов в тестовом стенде имеет смысл выбрать третий способ, так как он имеет преимущества по сравнению со вторым (см. таблицу 1), в

частности большую надёжность. Отклонение значения погрешности от желаемой 1 мс связано с передачей данных о начале старта запросов на сервер с помощью логирующих сообщений. Таким образом, можно сказать, что описываемый класс-шлюз и обработчик сокетов, как и предлагаемый способ реализации автоматического старта, обеспечивают автоматический старт запросов на клиентских машинах в точное время и логирование процесса тестирования в реальном времени в фоновом режиме.

Список использованных источников

1 https://searchengineland.com/google-now-handles-2-999-trillion-searches-per-year-250247

2 Костюков А. А. Моделирование реальной нагрузки для тестирования производительности сервер / А. А. Костюков // Science Time. 2016. № 1 (25). С. 238–243.

3 Хританков А. С. Модели и алгоритмы распределения нагрузки / А. С. Хританков // Информационные технологии и вычислительные системы. 2009. № 3. С. 33–48.

4 Быков А. Ю. Задача распределения заданий между клиентом и сервером в имитационном моделировании на основе технологии облачных вычислений и результаты экспериментов по её решению / А. Ю. Быков // Инженерный вестник. 2014. № 1. С. 7.

5 Marco Conti, Enrico Gregori, Willy Lapenna. Client-side content delivery policies in replicated web services : parallel access versus single server approach. Performance Evaluation, Volume 59, Issues 2–3, pages 137–157, 2005.

6 https://docs.microsoft.com/ru-ru/iis/get-started

7 https://socket.io/docs/

8 http://sta.github.io/websocket-sharp/