МИНОБРНАУКИ РОССИИ

**федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Вологодский государственный технический университет»
(ВОГТУ)**

Кафедра автоматики и вычислительной техники

ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Методические указания

к выполнению лабораторной работы «Инструментарий для проведения нагрузочного тестирования программного обеспечения»

Факультет электроэнергетический

Специальность: 230105 - программное обеспечение вычислительной техники и автоматизированных систем

Вологда 2013

УДК 681.3.06

Технологии разработки программного обеспечения:
Методические указания к выполнению лабораторной работы «Инструментарий для проведения нагрузочного тестирования программного обеспечения». - Вологда, ВоГТУ, 2013. - 28 с.

Методические указания содержат описание и методику выполнения лабораторной работы по указанной дисциплине. Для лабораторной работы указывается цель работы, приводится необходимый теоретический материал, методика проведения экспериментов. Имеются также варианты заданий и контрольные вопросы по теме работы.

Утверждено редакционно – издательским советом ВоГТУ.

Составители: Сергушичева А.П., канд.техн. наук, доц. каф. АВТ,
Сергушичева М.А., канд.техн. наук, ассистент каф.ИСиТ,
Малахова Д.А., студентка-магистрант

Рецензент: Швецов А.Н., декан ФЗДО, доктор техн. наук, профессор

ВВЕДЕНИЕ

Одним из важнейших процессов при создании программных приложений является тестирование. Тестированием называют деятельность, выполняемую для оценки качества программного обеспечения, которая в общем случае базируется на обнаружении дефектов и проблем в программных системах. Тестирование программных систем состоит из динамической верификации поведения программ на конечном (ограниченном) наборе тестов, выбранных соответствующим образом из обычно выполняемых действий прикладной области и обеспечивающих проверку соответствия ожидаемому поведению системы. Почти все виды тестирования (а приемочное и регрессионное особенно) предполагают проведение тестировщиками большого количества одинаковых тестов. В связи с этим, автоматизация тестирования помогает сэкономить множество человеко-часов.

Одним из инструментов для автоматизации тестирования является программный продукт JMeter, разрабатываемый Apache Jakarta Project. Этот свободно распространяемый инструмент предназначен для проведения нагрузочного тестирования, В данной лабораторной работе рассматриваются его возможности и порядок применения.

1. ЦЕЛЬ РАБОТЫ

Цель работы*:* изучение основных принципов нагрузочного тестирования и основных показателей (метрик) производительности и приобретение навыков работы с инструментарием для тестирования производительности.

2. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

2.1. Нагрузочное тестирование

Современное программное обеспечение должно работать бесперебойно при значительных нагрузках. Проблемы, связанные с плохой производительностью, могут стать причиной отказа клиентов от использования программного продукта. В связи с этим, проведение качественного нагрузочного тестирования должно стать обязательным, для обеспечения стабильности работы приложений.

2.1.1. Основные показатели производительности

Основными показателями производительности приложения являются.

1. Потребление ресурсов центрального процессора (CPU, %). Эта метрика, показывает, сколько времени из заданного определённого интервала было потрачено процессором на вычисления для выбранного процесса. В

современных системах важным фактором является способность процесса работать в нескольких потоках, чтобы процессор мог производить вычисления параллельно. Анализ истории потребления ресурсов процессора может объяснять влияние на общую производительность системы потоков обрабатываемых данных, конфигурации приложения и операционной системы, мультипоточности вычислений и других факторов.

2. Потребление оперативной памяти (Memory usage, Mb). Метрика, показывает количество памяти, использованной приложением. Использованная память может делиться на три категории:

- Virtual — объём виртуального адресного пространства, которое использует процессор. Этот объём не обязательно подразумевает, использование соответствующего дискового пространства или оперативной памяти. Виртуальное пространство конечно и процесс может быть ограничен в возможности загружать необходимые библиотеки.
- Private — объём адресного пространства, занятого процессором и не разделяемого с другими процессами.
- Working Set — набор страниц памяти, недавно использованных процессом. В случае, когда свободной памяти достаточно, страницы остаются в наборе, даже если они не используются. В случае, когда свободной памяти остаётся мало, использованные страницы удаляются.

При работе приложения память заполняется ссылками на объекты, которые, в случае неиспользования, могут быть очищены специальным автоматическим процессом, называемым «сборщиком мусора» (англ. Garbage Collector). Время затрачиваемое процессором на очистку памяти таким способом может быть значительным, в случае, когда процесс занял всю доступную память (в Java — так называемый «постоянный Full GC») или когда процессу выделены большие объёмы памяти, нуждающиеся в очистке. На время, требующееся для очистки памяти, доступ процесса к страницам выделенной памяти может быть заблокирован, что может повлиять на конечное время обработки этим процессом данных.

3. Потребление сетевых ресурсов. Эта метрика не связана непосредственно с производительностью приложения, однако её показатели могут указывать на пределы производительности системы в целом.

Например, серверное приложение, обрабатывая запрос пользователя, возвращает ему видео-поток, используя сетевой канал в 2 мегабит. Требование гласит, что сервер должен обрабатывать 5 запросов пользователей одновременно. Нагрузочное тестирование показало, что эффективно сервер может предоставлять данные только 4 пользователям одновременно, так как мультимедиа-поток имеет битрейт в 500 килобит. Очевидно, что предоставление этого потока 5 пользователям одновременно невозможно в силу превышения пропускной способности сетевого канала, а значит, система не удовлетворяет заданным требованиям

производительности, хотя при этом потребление ей ресурсов процессора и памяти может быть невысоким.

4. Работа с дисковой подсистемой (I/O Wait). Работа с дисковой подсистемой может значительно влиять на производительность системы, поэтому сбор статистики по работе с диском может помогать выявлять узкие места в этой области. Большое количество чтений или записей может приводить к простаиванию процессора в ожидании обработки данных с диска и в итоге увеличению потребления CPU и увеличению времени отклика.

5. Время выполнения запроса (request response time, ms). Время выполнения запроса приложением остаётся одним из самых главных показателей производительности системы или приложения. Это время может быть измерено на серверной стороне, как показатель времени, которое требуется серверной части для обработки запроса. На клиентской стороне определяют время, которое требуется на сериализацию/десериализацию, пересылку и обработку запроса. Следует иметь ввиду, что не каждое приложение для тестирования производительности может измерить оба этих времени.

2.1.2. Нагрузочное тестирование и его виды

Нагрузочное тестирование (Load Testing) или **тестирование производительности** (Performance Testing) - это автоматизированное тестирование, имитирующее работу определенного количества пользователей на каком либо общем (разделяемом ими) ресурсе. При этом определяют, собирают и анализируют показатели производительности и времени отклика программно-технической системы (устройства) на внешний запрос, проверяют их соответствие требованиям. Приветствуется измерение производительности системы на ранней стадии разработки. Нагрузочное тестирование на первых стадиях готовности архитектурного решения ('Proof-of-Concept'-тестирование) призвано определить его состоятельность.

Основная цель нагрузочного тестирования заключается в том, чтобы, создав определённую ожидаемую в системе нагрузку (посредством виртуальных или реальных пользователей), наблюдать за показателями производительности системы. Например, веб-сервис с функциональностью корзины покупателя рассчитан на 100 одновременно работающих пользователей, таким образом, что 25 из них просматривают товар и выходят из системы; 25 – добавляют товар в корзину, оформляют его и выходят из системы; 25 – используют функцию возврата товара и выходят из системы; 25 – входят в систему и не проявляют никакой активности. Нагрузочное тестирование, позволяющее удостовериться, что данная система готова к выходу в эксплуатацию, должно эмулировать вышеописанный типичный сценарий работы с веб-сервисом. При этом для анализа могут сниматься

показатели производительности системы в целом или каждого ее узла в частности.

В качестве критериев успешности нагрузочного тестирования выступают требования к производительности системы, сформулированные (и записанные в соответствующем документе) на стадии разработки функциональных спецификаций. Однако, если такие требования отсутствуют, то первое нагрузочное тестирование будет являться пробным (exploratory load testing) и основываться на разумных предположениях об ожидаемой нагрузке и потреблении аппаратной части ресурсов.

Практика моделирования ожидаемого применения приложения с помощью эмуляции работы нескольких пользователей одновременно больше всего подходит для мультипользовательских систем, использующих клиент-серверную архитектуру (например, веб-серверов). Однако и другие типы программных продуктов могут быть протестированы подобным способом. Например, текстовый или графический редактор можно заставить прочесть очень большой документ; а финансовый пакет — сгенерировать отчёт на основе данных за несколько лет. Наиболее адекватно спроектированный нагрузочный тест даёт более точные результаты.

Можно выделить следующие **виды** нагрузочного тестирования:

1. Тестирование производительности (Performance testing). Для определения масштабируемости приложения под нагрузкой требуется:

- измерить время выполнения выбранных операций при определенных интенсивностях выполнения этих операций
- определить количество пользователей, одновременно работающих с приложением
- определить границы приемлемой производительности при увеличении нагрузки (при увеличении интенсивности выполнения этих операций)
- исследовать производительности на высоких, предельных, стрессовых нагрузках

2. Стрессовое тестирование (Stress Testing). Позволяет проверить насколько приложение и система в целом работоспособны в условиях стресса, а также оценить способность системы к регенерации, т.е. к возвращению к нормальному состоянию после прекращения воздействия стресса. Стрессом в данном контексте может быть повышение интенсивности выполнения операций до очень высоких значений или аварийное изменение конфигурации сервера. Также одной из задач при стрессовом тестировании может быть оценка деградации производительности, таким образом, цели стрессового тестирования могут пересекаться с целями тестирования производительности. Не существует чёткой границы между нагрузочным и стресс-тестированием, однако эти виды тестирования отвечают на разные бизнес-вопросы и используют различную методологию.

3. Объемное тестирование (Volume Testing). Задачей объемного тестирования является получение оценки производительности при увеличении объема информации в базе данных приложения. При этом измеряют время выполнения выбранных операций при определенных интенсивностях выполнения этих операций и (или) определяют количество пользователей, одновременно работающих с приложением.

4. Тестирование стабильности или надежности (Stability / Reliability Testing). Задачей тестирования стабильности (надежности) является проверка работоспособности приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки. Времена выполнения операций могут играть в данном виде тестирования второстепенную роль. При этом на первое место выходит отсутствие утечек памяти, перезапусков серверов под нагрузкой и другие аспекты, влияющие именно на стабильность работы.

2.1.3. Основные принципы нагрузочного тестирования

Основные принципы нагрузочного тестирования обобщают эмпирическую информацию, полученную при тестировании производительности в целом, и применимы к любому типу такого тестирования:

1. **Запросы уникальны.** Даже сформировав реалистичный сценарий работы с системой на основе статистики ее использования, необходимо понимать, что всегда найдутся исключения из этого сценария. Например, существует пользователь, обращающийся к отличным от всех остальных, уникальным страницам веб-сервиса.

2. **Время отклика системы,** в общем случае, **подчиняется функции нормального распределения**. Это означает, что на основе достаточного количества измерительной информации, можно определить вероятность с которой отклик системы на запрос попадёт в тот или иной интервал времени.

3. **Время отклика системы зависит от** степени ее **распределённости**, т.е. на разброс значений времени отклика системы влияют одновременно количество запросов, приходящихся на каждый узел системы, и количество узлов, каждый из которых добавляет некоторую случайную величину задержки при обработке запросов.

4. **Существует разброс времени отклика системы.** При достаточно большом количестве измерений величины времени обработки запроса в любой системе всегда найдутся запросы, время обработки которых превышает определённые в требованиях максимумы. При этом, чем больше суммарное время проведения эксперимента тем выше окажутся новые максимумы.

5. Необходимая **точность воспроизведения профилей нагрузки** тем дороже, чем больше компонент содержит система. Часто невозможно учесть все аспекты профиля нагрузки для сложных систем. Чем сложнее система, тем

больше времени будет затрачено на проектирование, программирование и поддержку адекватного профиля нагрузки для неё, что не всегда является необходимостью. Оптимальный подход в данном случае заключается в балансировании между стоимостью разработки теста и покрытием функциональности системы, в результате которого появляются допущения о влиянии на общую производительность той или иной части тестируемой системы.

2.1.4. Этапы проведения нагрузочного тестирования

1. Анализ требований и сбор информации о тестируемой системе.
2. Разработка модели нагрузки
3. Выбор инструмента для нагрузочного тестирования
4. Создание и отладка тестовых скриптов
5. Проведение тестирования
6. Анализ результатов
7. Подготовка, отправка и публикация отчета по проведенному нагрузочному тестированию

2.2. Инструментарий для тестирования производительности

Следует отметить, что для большинства видов тестирования производительности используется один и тот же инструментарий, умеющий выполнять типовые задачи. В отличие от инструментов для автоматизации регрессионного тестирования, работающих на уровне объектов графического пользовательского интерфейса, инструменты для нагрузочного тестирования работают на уровне протокола. Например, имеется стандартный интернет-браузер, выполняющий функцию перехода по указанной ссылке при нажатии кнопки. В данном случае для автоматизации регрессионного тестирования необходимо написать скрипт, передающий браузеру клик мышью и нажатие кнопки, в то время как при создании скрипта для нагрузочного тестирования нужно написать передачу гиперссылки от браузера для нескольких пользователей, включая уникальные для каждого из них имя пользователя и пароль.

2.2.1. Автоматизация **нагрузочного тестирования**

Автоматизация не сокращает этап подготовки к тестированию, а, наоборот, увеличивает его. Преимущества автотестирования станут очевидны только после выпуска первой стабильная версии приложения. Средства автоматизации тестирования могут существенно сократить сроки проверки версий: автотесты прогоняют ночью, а на следующий день анализируют выявленные проблемы. Благодаря автоматизации можно не просто ускорить процесс тестирования, но и увеличить тестовое покрытие за счет большего количества перебираемых комбинаций входных данных, что в свою очередь позволяет снизить

требования к квалификации разработчиков – с большей вероятностью их ошибки будут обнаружены на этапе тестирования. Также автоматизация тестирования позволяет если не избежать, то значительно уменьшить синдром «замыленного глаза», когда тестировщик перестает замечать ошибки при выходе новых версий.

Существуют различные инструменты для обнаружения и исследования проблем в разных узлах системы: приложениях, базах данных, сети, обработки на клиентской стороне, балансировки нагрузки.

Инструменты нагрузочного тестирования являются более сложными по сравнению со средствами автоматизации функционального тестирования. Они фактически «перехватывают» трафик между тестируемым приложением и сервером и представляют его в виде, удобном для работы. Для инструментов нагрузочного тестирования требуется поддержка протокола, который использует тестируемым приложением, наличие встроенных средств мониторинга параметров серверов, возможность гибкой настройки сценариев нагрузочного тестирования, наличие средств анализа результатов и построения отчетов о нагрузочном тестировании.

Практически все производители продуктов для автоматизированного тестирования предлагают средства нагрузочного тестирования. Лидерами на рынке средств автоматизированного нагрузочного тестирования являются HP с продуктом LoadRunner и IBM с продуктами Robot и Performance Tester, поддерживающими множество протоколов (включая терминальные протоколы). Большинство средств нагрузочного тестирования работают лишь с Web-приложениями.

Помимо собственно средств тестирования существуют так называемые средства поддержки процесса тестирования, позволяющие вести учет требований и тест-кейсов, проводить анализы покрытия требования тестами, управлять ходом выполнения тестирования, вести учет обнаруженных дефектов и т.п. Лидирует в данной области Web-приложение HP Quality Center – единый инструмент управления процессом тестирования (включая управление требованиями и дефект-менеджмент), интегрируемый со средствами функционального и нагрузочного тестирования HP QuickTest Professional и LoadRunner. С данным инструментом конкурирует продукт Rational Quality Manager (RQM) от IBM, представляющий собой Web-приложение на платформе Jazz.

В числе **коммерческих инструментов** для автоматизированного нагрузочного тестирования: HP Performance Center, включая HP LoadRunner (Hewlett-Packard (Mercury Interactive)), Rational Performance Tester (IBM Rational), Silk Performer (Borland (Segue)), LoadComplete Web Load Testing (SmartBear), NeoLoad (Neotys). Существуют для автоматизированного нагрузочного тестирования и **бесплатные инструменты,** например, Jmeter, Grinder. Наиболее популярные инструменты для нагрузочного тестирования представлены в таблице 1.

Таблица 1 – Инструменты для нагрузочного тестирования

ПО	Наименование производителя	Комментарии
OpenSTA	'Open System Testing Architecture'	Свободно распространяемое программное обеспечение для нагрузочного/стресс тестирования, лицензированное GNU GPL. Использует распределённую архитектуру приложений, основанную на CORBA. Доступна версия под Windows, хотя имеются проблемы с совместимостью с Windows Vista. Поддержка прекращена в 2007 году.
IBM Rational Performance Tester	IBM	Основанное на среде разработки Eclipse ПО, позволяющее создавать нагрузку больших объёмов и измерять время отклика для приложений с клиент-серверной архитектурой. Требует лицензирования
JMeter	Открытый проект Apache Jakarta Project	Основанный на Java кроссплатформенный инструментарий, позволяющий производить нагрузочные тесты с использованием JDBC / FTP / LDAP / SOAP / JMS / POP3 / HTTP / TCP соединений. Даёт возможность создавать большое количество запросов с разных компьютеров и контролировать процесс с одного из них
HP LoadRunner	HP	Инструмент для нагрузочного тестирования, изначально разработанный для эмуляции работы большого количества параллельно работающих пользователей. Также может быть использован для unit- или интеграционного тестирования
SilkPerformer	Micro Focus	
Visual Studio Load Test	Microsoft	Visual Studio предоставляет инструмент для тестирования производительности включая load / unit testing
LoadComplete	SmartBear	

Ниже кратко представлены некоторые инструменты для автоматизированного тестирования:

- **IBM Rational Performance Tester** – инструмент нагрузочного и стрессового тестирования, с помощью которого можно выявлять проблемы системной

производительности и их причины. Позволяет создавать тесты без написания кода и не требует навыков программирования. Обеспечивает гибкие возможности моделирования и эмуляции различных пользовательских нагрузок. Выполняет сбор и интеграцию данных о серверных ресурсах с данными о производительности приложений, получаемыми в режиме реального времени.

- **HP LoadRunner** – программный продукт для автоматизации нагрузочного тестирования широкого набора программных сред и протоколов. Поддерживает SOA, работу с Web-сервисами, Ajax, RDP, SQL, продуктами Citrix, платформы Java, .Net, а также все основные ERP- и CRM-приложения от PeopleSoft, Oracle, SAP и Siebel. Пакет HP LoadRunner включает в себя более 60 мониторов сбора данных о тестируемой инфраструктуре и предоставляет детальную диагностику по работе приложений.

- **OpenSTA** (Open Systems Testing Architecture) – больше, чем приложение для тестов, это открытая архитектура, проектируемая вокруг открытых стандартов. Проект создан в 2001 году группой компаний CYRANO, которая поддерживала коммерческую версию продукта, но CYRANO распался, и сейчас OpenSTA распространяется как приложение с открытым кодом под лицензией GNU GPL, работает в Windows NT 4.0SP5/2000/XP. Для работы требует Microsoft Data Access Components (MDAC), который можно скачать с сайта корпорации. Текущий инструментарий позволяет провести нагрузочное испытание HTTP/HTTPS сервисов, хотя его архитектура способна на большее. OpenSTA позволяет создавать тестовые сценарии на специализированном языке SCL (Script Control Language). Для упрощения создания и редактирования сценариев используется специальный инструмент Script Modeler. Выбираем Tools – Canonicalize URL, запустится веб-браузер. Просто ходим по сайту, собирая ссылки, которые будут сохранены в скрипт. Все параметры запроса поддаются редактированию, возможна подстановка переменных. Структура теста и заголовки будут выводиться во вкладках в панели слева. Тесты удобно объединять в наборы. Настройки прокси задаются в самом скрипте, поэтому можно указать несколько серверов. Реализована возможность организации распределенного тестирования. Каждая из машин такой системы может выполнять свою группу заданий, а repository host осуществляет сбор и хранение результатов. После установки на каждой тестирующей системе запускается сервер имен, работа которого обязательна. Поддерживается аутентификация пользователей на веб-ресурсе и установление соединений по протоколу SSL. Параметры работы нагружаемой системы можно контролировать с помощью SNMP и средств Windows NT. Результаты тестирования, включающие время откликов, количество переданных байт в секунду, коды ответа для каждого запроса и количество ошибок выводятся в виде таблиц и графиков. Использование большого числа фильтров позволяет отобрать необходимые результаты. Результат можно экспортировать в CSV-файл. Возможности по выводу отчетов несколько

ограничены, но по ссылкам на сайте можно найти скрипты и плагины, упрощающие, в том числе, анализ полученной информации.

- Apache JMeter (jakarta.apache.org/jmeter) является Java-приложением с открытым кодом, предназначен для нагрузочного тестирования не только веб-приложений и их отдельных компонентов (скрипты, сервлеты, Java объекты и др.), но также FTP-серверов, баз данных (с использованием JDBC) и сети. Функциональность расширяется с помощью плагинов. Поддерживается SSL (через Java Secure Sockets Extension). Возможно проведение тестов, как с использованием графического интерфейса, так и из командной строки. Использование Java подразумевает кроссплатформенность, поэтому JMeter уверенно работает в различных *nix-системах, в Windows от 98 и некоторых других ОС. Распространяется под Apache License.
В JMeter предусмотрены механизмы авторизации виртуальных пользователей, поддерживаются пользовательские сеансы, шаблоны, кэширование и последующий offline анализ результатов теста; функции позволяют сформировать следующий запрос, основываясь на ответе сервера на предыдущий. Есть возможность проводить распределенные тесты. В этом случае один из компьютеров является сервером (bin/jmeter-server.bat), который управляет клиентами и собирает итоговую информацию. Для работы достаточно запустить ApacheJMeter.jar или в консоли jmeter.bat (Windows) или jmeter.sh (*nix).

- WAPT (Web Application Testing) позволяет испытать устойчивость веб-сайта и других приложений, использующих веб-интерфейс, к реальным нагрузкам. Разрабатывается новосибирской компанией SoftLogica LLC. Это одна из самых простых в использовании программ обзора. Для проведения простого теста даже не нужно заглядывать в документацию, интерфейс прост, но не локализован. Работает под управлением Windows от 98, поддерживается и Vista. Для проверки WAPT может создавать множество виртуальных пользователей, каждый с индивидуальными параметрами. Поддерживается несколько видов аутентификации и куки. Сценарий позволяет изменять задержки между запросами и динамически генерировать некоторые испытательные параметры, максимально имитируя таким образом поведение реальных пользователей. В запрос могут быть подставлены различные варианты HTTP-заголовка, в настройках можно указать кодировку страниц. Параметры User-Agent, X-Forwarded-For, IP указываются в настройках сценария. Значения параметров запроса могут быть рассчитаны несколькими способами, в том числе, определены ответом сервера на предыдущий запрос, используя переменные и функции. Поддерживается работа по защищенному протоколу HTTPS (и все типы прокси-серверов). Созданные сценарии, сохраняемые в файле XML-формата, можно использовать повторно. Кроме стандартных Performance и Stress, в списке присутствуют несколько других

тестов, позволяющих определить максимальное количество пользователей и тестировать сервер под нагрузкой в течение долгого периода.

- NeoLoad (www.neotys.com) - еще одна система, позволяющая провести нагрузочное тестирование веб-приложений. Написана на Java, работает на компьютерах под управлением Windows NT/2000/XP, Linux и Solaris. В отчете можно получить подробную информацию по каждому загруженному файлу. NeoLoad весьма удобен для оценки работы отдельных компонентов (AJAX, PHP, ASP, CGI, Flash, апплетов и пр.). Возможна установка времени задержки между запросами (thinktime) глобально и индивидуально для каждой страницы. Тестирование проводится как с использованием весьма удобной графической оболочки, так и с помощью командной строки (используя заранее подготовленный XML-файл). Поддерживает работу с протоколом HTTPS, с HTTP и HTTPS прокси, basic веб-аутентификацию и cookies, автоматически определяя данные во время записи сценария, и затем проигрывает во время теста. Для работы с различными профилями для регистрации пользователей могут быть использованы переменные. При проведении теста можно задействовать дополнительные мониторы (SNMP, WebLogic, WebSphere, RSTAT и Windows, Linux, Solaris), позволяющие контролировать и параметры системы, на которой работает веб-сервер. При помощи NeoLoad можно проводить и распределенные тесты. Один из компьютеров является контролером, на остальные устанавливаются генераторы нагрузки (loadGenerator). Контролер распределяет нагрузку между loadGenerator и собирает статистику. Очень удобно реализована работа с виртуальными пользователями. Пользователи имеют индивидуальные настройки, затем они объединяются в Populations (должна быть создана как минимум одна Populations), в Populations можно задать общее поведение (например, 40% пользователей популяции посещают динамические ресурсы, 20% читают новости). Виртуальные пользователи могут иметь индивидуальный IP-адрес, полосу пропускания и свой сценарий теста.

- Корпорация Microsoft предлагает целых два продукта, позволяющих протестировать веб-сервер под нагрузкой. Это**Microsoft Application Stress Tool** и **Web Capacity Analysis Tool**. Первый распространяется как отдельный продукт и имеет графический интерфейс. Второй входит в состав комплекта инструментов **Internet Information Services 6.0 Resource Kit Tools**, работает из командной строки. **MAST** более наглядный, в создании теста поможет простой мастер создания тестов, возможна работа с cookies, регулировка нагрузки по разным URL. Сценарий тестирования может быть создан вручную или записан с помощью веб-браузера и при необходимости отредактирован. В **WAST** уровень нагрузки (stress level) регулируется путем задания количества нитей, осуществляющих запросы к серверу, а число виртуальных пользователей рассчитывается как произведение числа нитей на число сокетов, открытых каждой из нитей. По окончании теста получаем простой

отчет в текстовой форме, в котором дана информация по числу обрабатываемых запросов в единицу времени, среднему времени задержки, скорости передачи данных на сервер и с сервера, количеству ошибок и т.д. Отчет можно экспортировать в CSV-файл. Никаких возможностей по статистической обработке не предусмотрено, то есть с его помощью можно только оценить работу при определенных условиях.

- Selenium это объектно-ориентированное JavaScript приложение, которое может анализировать файлы определенной структуры для того, чтобы находить в них команды для манипуляции браузером и команды для выполнения определенных действий и проверок. Этот инструмент позволяет записывать и воспроизводить скрипты, представляющие собой обычные HTML-страницы с одной таблицей, содержащей команды.

- BrowserMob - это набор сервисов, которые предоставляются через интернет, которые дают возможность мониторинга и нагрузочного тестирования веб сайтов, позволяет проверить скорость загрузки целевого сайта из 4 разных мест и показывающий много подробных метрик загрузки для каждого из них.

- Testing Anywhere – это универсальный и легкий в использовании инструмент автоматизации, который позволяет автоматизировать разные типы тестирования. Запись GUI сценариев и простота использования без особых навыков программирования дают значительное преимущество в создании тестов, а встроенный редактор тестов позволяет с легкостью модифицировать записанные тесты.

- Яндекс.танк – новый инструмент для нагрузочного тестирования (внутренняя разработка яндекса). Этот консольный инструмент пока не имеющийет графического интерфейса, но дает довольно полную картину в этой самой консоли. Позволяет нагружать одну страницу или сразу список урлов (которые будут запрашиваться примерно в одинаковых соотношениях). Можно самому составить список запросов со своими заголовками. Например, можно реализовать нагрузку от запросов от анонимных пользователей и залогиненных. Через скрипт (php, python, bash, etc) залогиниться на сайте и получить нужные куки. Сгенерировать в нужном формате данные для танка и запустить тест. Танк позволяет создать 3 вида нагрузки: 1) постоянная (указывается количество запросов и время), 2) линейно-нарастающая (указывается начальное и конечное значение и время), 3) ступенчато-нарастающая (указывается начальное и конечное значение, шаг увеличения нагрузки и время). Указанные виды нагрузки можно комбинировать в одном тесте. Синтаксис команд прост и понятен.

2.2.2. JMeter

JMeter - один из быстрых и надежных инструментов для автоматизации нагрузочного тестирования, Хотя изначально JMeter разрабатывался Apache

Jakarta Project как средство тестирования web-приложений, в настоящее время он способен проводить нагрузочные тесты для JDBC-соединений, FTP, LDAP, SOAP, JMS, POP3, IMAP, HTTP и TCP.

Основными преимуществами JMeter являются:

— использование Java. Соответственно получаем независимость от операционной системы как плюс и большое потребление RAM как минус;

— свободное распространение. На начальном этапе стоит использовать бесплатные инструменты. Это позволит лучше понять, какие именно тесты нужны, дать ориентировочную оценку аппаратным ресурсам для проведения таких тестов, понять достаточно ли возможностей бесплатных инструментов для решения поставленных задач. Переходить на платные инструменты следует тогда, когда более четко определены задачи по тестированию;

— простота и высокая скорость обучения;

— универсальность и обширные возможности (применим не только для веб-приложений);

— возможность работы как через GUI, так и из консоли;

— удобное масштабирование (есть готовое решение JMeter SaaS на AWS);

— расширяемость (поддерживает плагины сторонних разработчиков, позволяет дополнять инструмент новыми функциями);

— широкая распространенность и хорошая поддержка (в Internet есть много ресурсов по JMeter: руководство пользователя онлайн, пользовательские форумы, веб-примечания и т.д.);.

— параллелизм (создание множества потоков и настройка каждого потока в отдельности);

— мобильность (может работать на любых JVM).

JMeter работает, действуя как клиент клиент-серверного приложения. Он способен измерять время отклика, и все другие ресурсы сервера, такие как загрузка ЦП и использование памяти. В программе реализованы механизмы авторизации виртуальных пользователей, поддерживаются пользовательские сеансы. Организовано логирование результатов теста и разнообразная визуализация результатов в виде диаграмм, таблиц и т. п. JMeter имеет встроенный прокси-сервер, который предназначен для записи сессий, но можно использовать и внешний. Перед началом тестирования необходимо составить тестовый план, описывающий серию заданий, которые необходимо выполнить **JMeter**. Он должен содержать одну или несколько групп потоков (Thread Groups) и другие элементы:

- Логические контроллеры (Logic controllers);
- Типовые контроллеры (Sample generating controllers);
- Слушатели (Listeners);
- Таймеры (Timers);

— Соответствия (Assertions);
— Конфигурационные элементы (Configuration elements).

Работа с системой начинается с добавления группы потоков (Edit - Add - Thread Group). В ее настройках указываем название, количество запускаемых потоков, то есть виртуальных пользователей (Number of threads), время задержки между запуском потоков (Ramp-Up Period), количество циклов выполнения задания (Loop Count), здесь же можно определить выполнение задания по расписанию (Sheduler). Далее, щелкая в созданную группу, необходимо добавить образец запроса (Sampler), выбрав его из списка. Для нагрузочного тестирования или проверки работоспособности сервера достаточно выбрать HTTP Request (Add -Sampler - HTTP Request). Здесь указываем название, IP-адрес и порт веб-сервера, протокол, метод передачи данных (GET, POST), параметры переадресации, передачу файлов на сервер. Настраиваем и жмем на Run. Вывод результата осуществляется с помощью Listeners, каждый по-своему выводит результат. Например, Aggregate Graph выводит суммарные результаты теста в виде таблицы и графика.

Тесты для JMeter создаются визуально и имеют древовидную структуру в окне редактирования теста. Основной сценарий тестирования JMeter (план тестирования) может включать создание цикла, который моделирует последовательные запросы к серверу с предопределенным интервалом и группой потока, которая моделирует параллельную загрузку.

При создании теста JMeter предлагает несколько типов компонент:

Samplers - основные элементы, которые непосредственно общаются с тестируемым приложением, например http sampler для обращения к веб-приложению

Logic controllers - элементы, позволяющие группировать другие элементы в циклы, группы параллельного запуска, и т.д.

Assertions - элементы, выполняющие контроль. С их помощью вы можете проверить текст, который вы ожидаете на веб-странице или, например, указать что вы ожидаете ответ от сервера не более чем 2 секунды. Если какой-либо Assert не будет удовлетворен, тест будет иметь негативный результат.

Являясь удобным инструментом для запуска и мониторинга тестов, Jmeter предоставляет хорошие возможности для просмотра отчетов. Содержание отчета настраивает тестировщик. В отчет можно включать различные таблицы и графики. Типичный пример отчета в графической форме представлен на рисунке 1. Элемент Graph Results строит графики на основании данных, поступающих в ходе тестирования. На графике линиями разного цвета отражены результаты проведенного тестирования: черным – время каждого запроса; синим – текущее среднее время всех запросов; сиреневым – медиана; красным – текущее стандартное отклонение; зеленым – текущая интенсивность потока (число обрабатываемых сервером запросов в минуту).

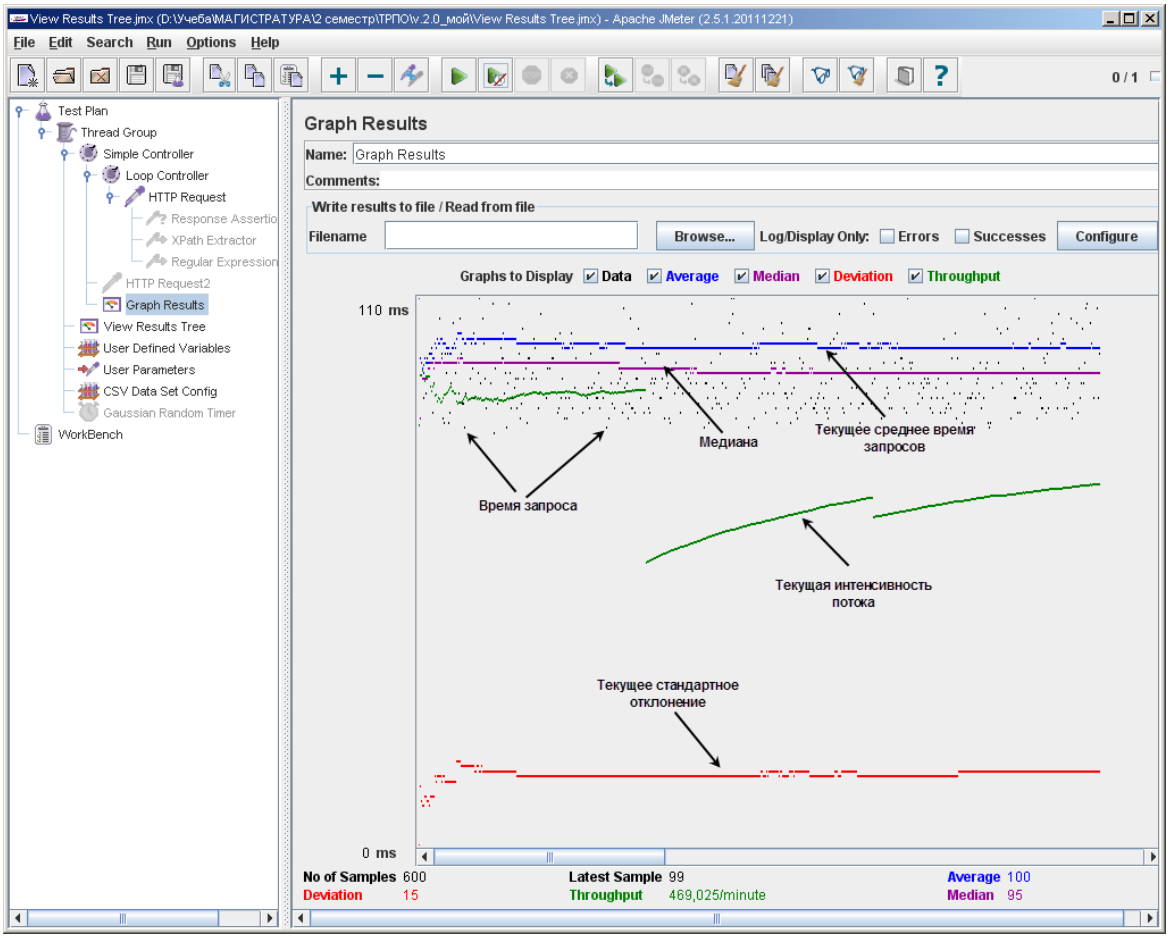

Рис. 1 - Пример отчета JMeter

3. ПРИМЕНЕНИЕ JMETER

3.1. Настройка JMeter

Для работы JMeter на компьютере должна быть установлена Java. Папку JMETER_MODIFICATION скопируйте, например, на диск C:/temp. Предлагаемый пример можно выполнить с помощью указаний JMeter.swf (файл открывается любым браузером). Рекомендуется следующая последовательность настройки JMeter.

1. В папке JMETER_MODIFICATION необходимо открыть папку bin и с помощью любого текстового редактора в файле jmeter.bat отредактировать строчку :

set JM_LAUNCH=c:\PROGRA~1\Java\jdk1.5.0_11\bin\java.exe ,

вписав в нее путь до java.exe, и сохранить файл. Обратите внимание, что в пути к файлу не должно быть никаких пробелов. Так, например, название папки «Program Files» необходимо заменить на «PROGRA~1».

2. Запустите файл jmeter.bat. Должно открыться главное окно JMeter (рис.2).

В поле «Name» введите название рабочей области, например «Trpo_workBench».

Примечание: WorkBench - это временный элемент. Дело в том, что интерфейс Jmeter построен на “перетаскивании” с места на место элементов дерева сценария. Вы создаете дерево плана, а затем можете его части перемещать на Workbench и обратно. Выполняется в ходе тестирование только содержимое Test Plan, а содержимое Workbench даже не сохраняется на диск.

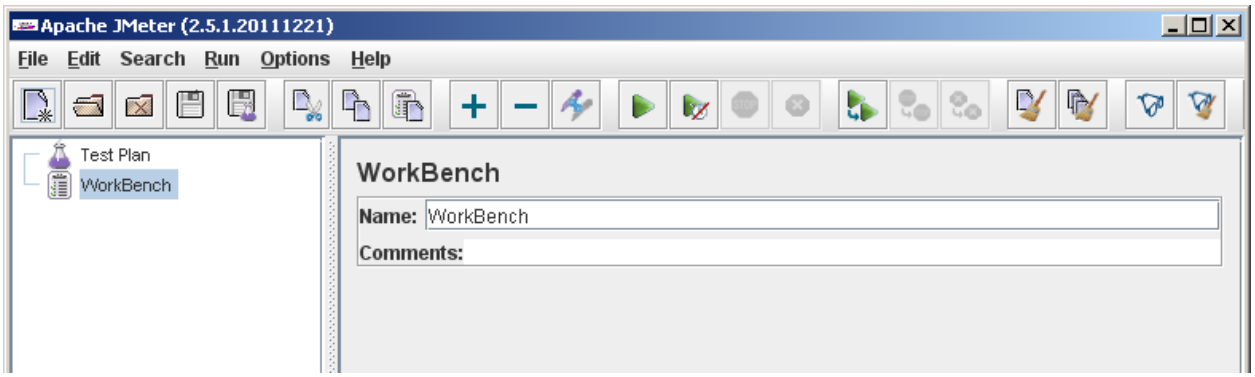

Рис.2 – Окно JMeter

3. Переходим в Test Plan и изменяем название, например, на «Trpo_TestPlan»

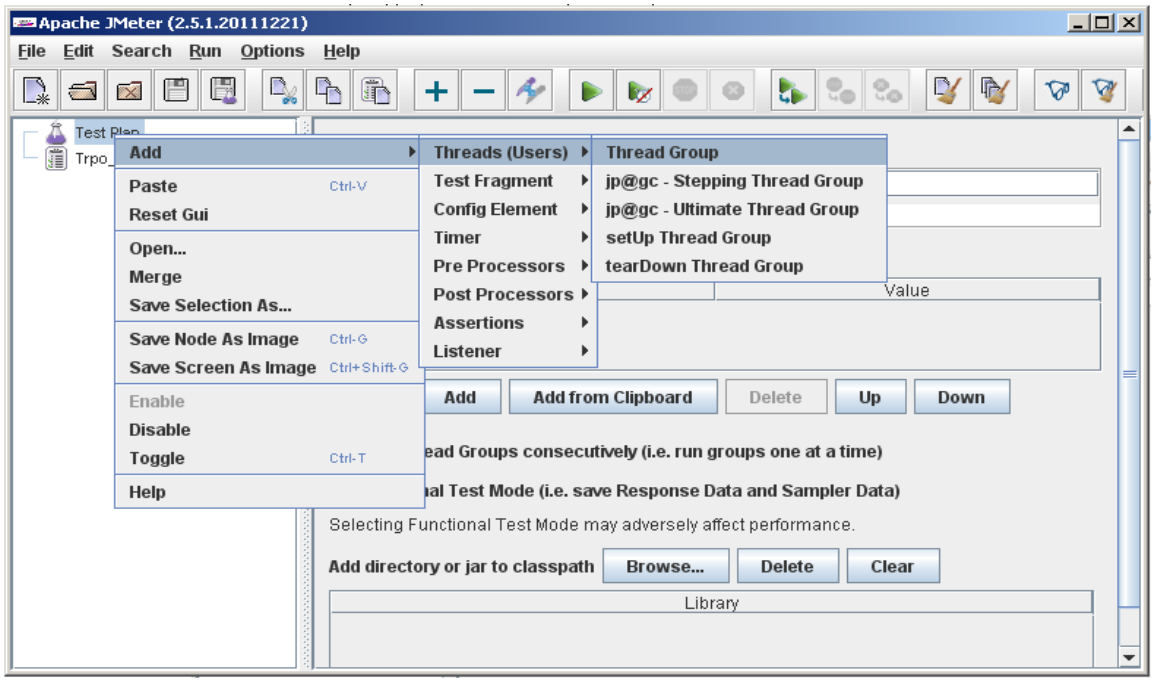

Рис.3 – Добавление группы виртуальных пользователей

4. Правой кнопкой мыши открываем контекстное меню TestPlan и добавляем группу виртуальных пользователей, как показано на рисунке 3, и изменяем ее название, например на «Trpo_ThreadGroup»

Данная настройка контролирует количество пользователей, которые будут заходить на сайт (Number of Threads), время, за которое зайдут в систему

все пользователи(Ramp-up Period) и количество повтора для теста (Loop Count).

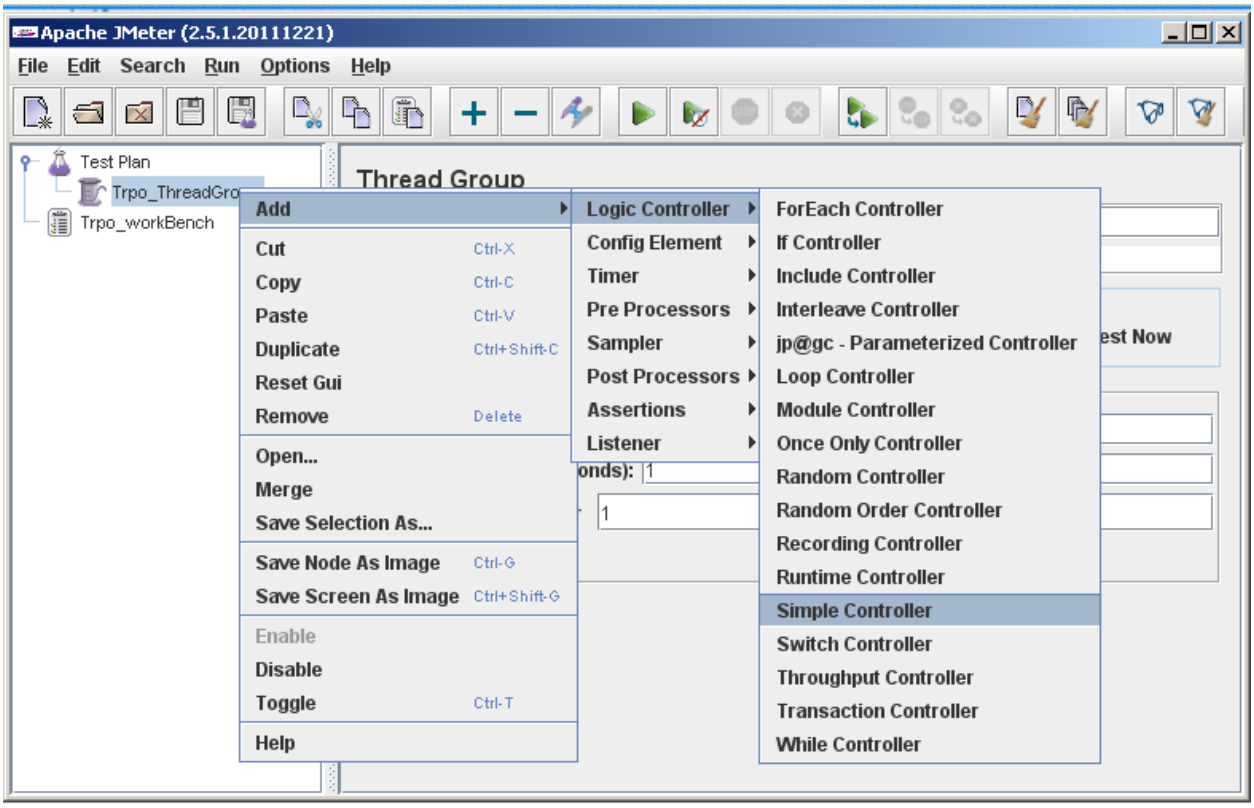

Рис.4 – Добавление контроллера

5. Выбрав только что созданную группу виртуальных пользователей, нужно добавить в нее контроллер (рис.4), который будет использовать прокси-сервер. Задаем ему имя, например «Trpo_SimpleController» (Trpo_ThreadGroup – Add – Logic Controller – Simple Controller).

6. Для элемента рабочей области (Trpo_workBench) добавляем прокси-сервер, с помощью которого будет происходить запись теста (рис.4). Задаем для него имя, например, «Trpo_ProxyServer»

Для исключения конфликтов в поле Port необходимо ввести любой свободный порт, например 8090.

Примечание: Для того, чтобы посмотреть какие порты заняты переходим в командную строку, открываем консоль (команда cmd), далее набираем в командной строке команду: **netstat -ano** – на экране отобразится список всех активных подключений с номерами занятых портов.

В выпадающем списке Target Controller выбираем созданный ранее Simple Controller (Trpo_SimpleController) – в него будут записывать действия пользователя.

Далее на этой же вкладке нажимаем кнопку Start для запуска сервера.

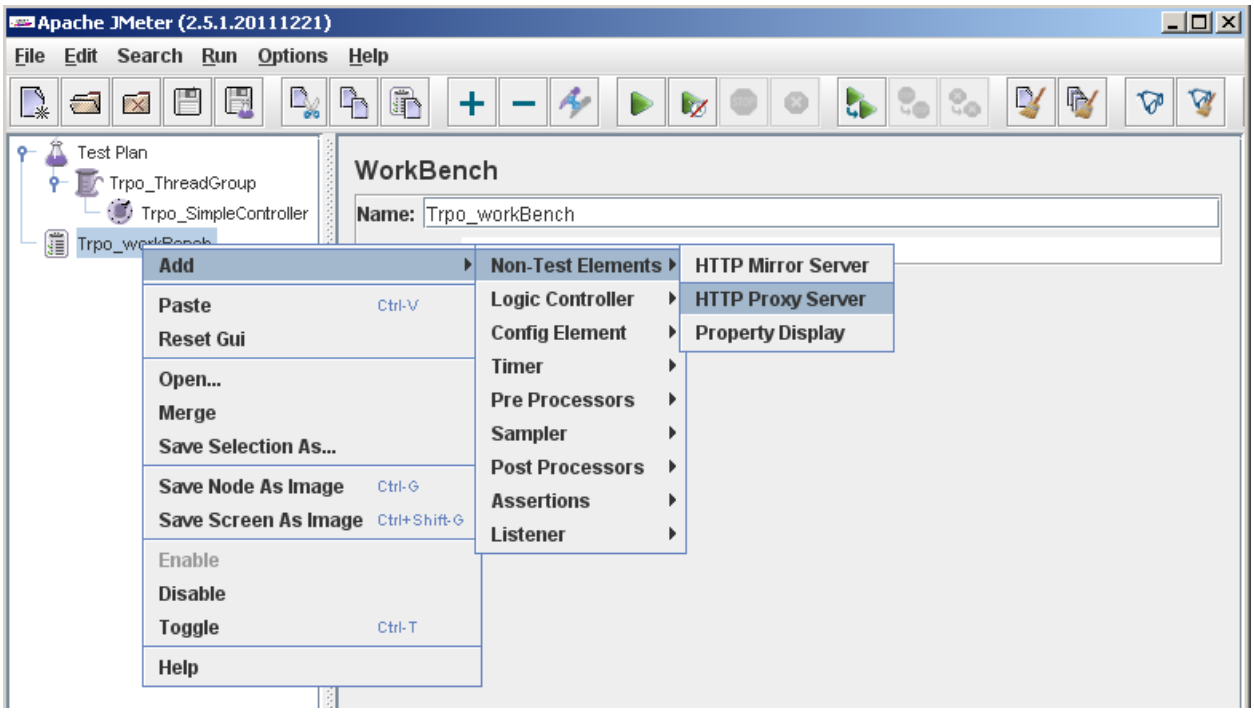

Рис.5 – Добавление прокси-сервера

3.2. Настройка браузера

В FireFox заходим в Инструменты – Настройки – Дополнительные – вкладка Сеть – кнопка Настроить. Выбираем «Ручная настройка прокси-сервера». В поле «Порт» вводим номер порта, указанного в настройках прокси-сервера Jmeter. В поле «HTTP прокси» вводим «127.0.0.1» и ставим галочку «Использовать этот прокси-сервер для всех протоколов».

Примечание: Убедитесь, что поле «Не использовать этот прокси сервер для» не заполнено никакими значениями.

Сохраняем настройки.

Для настройки прокси в Internet Explorer необходимо зайти в меню Сервис – Свойства обозревателя – вкладка Подключения – Настройка LAN – поставить галочку «Использовать прокси-сервер для подключений LAN», в поле «Адрес» вводим 127.0.0.1 в поле «Порт» вводим номер порта, указанного в настройках прокси-сервера Jmeter.

3.3. Пример работы

1. В FireFox открываем любой сайт, например, сайт практикума по программированию (atpp.vstu.edu.ru -> Практикум по программированию). Ходим по ссылкам сайта – в это время все действия записываются, для их дальнейшего воспроизведения JMeter.

2. Переходим в JMeter , раскрываем созданный Simple Controller и видим все записанные запросы (рис.6):

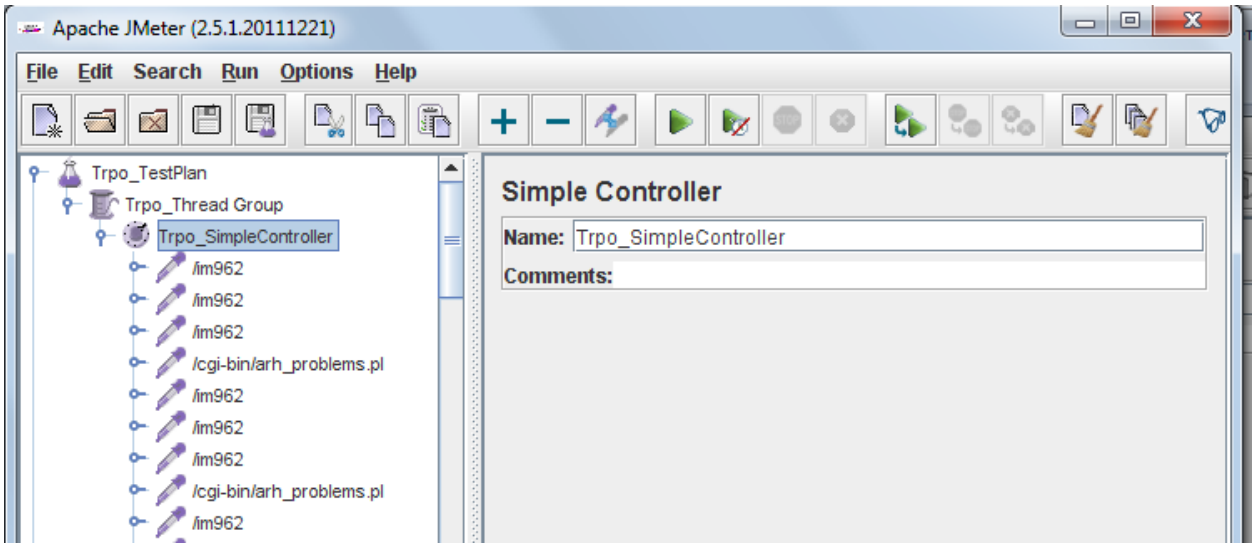

Рис.6 – Записанные запросы

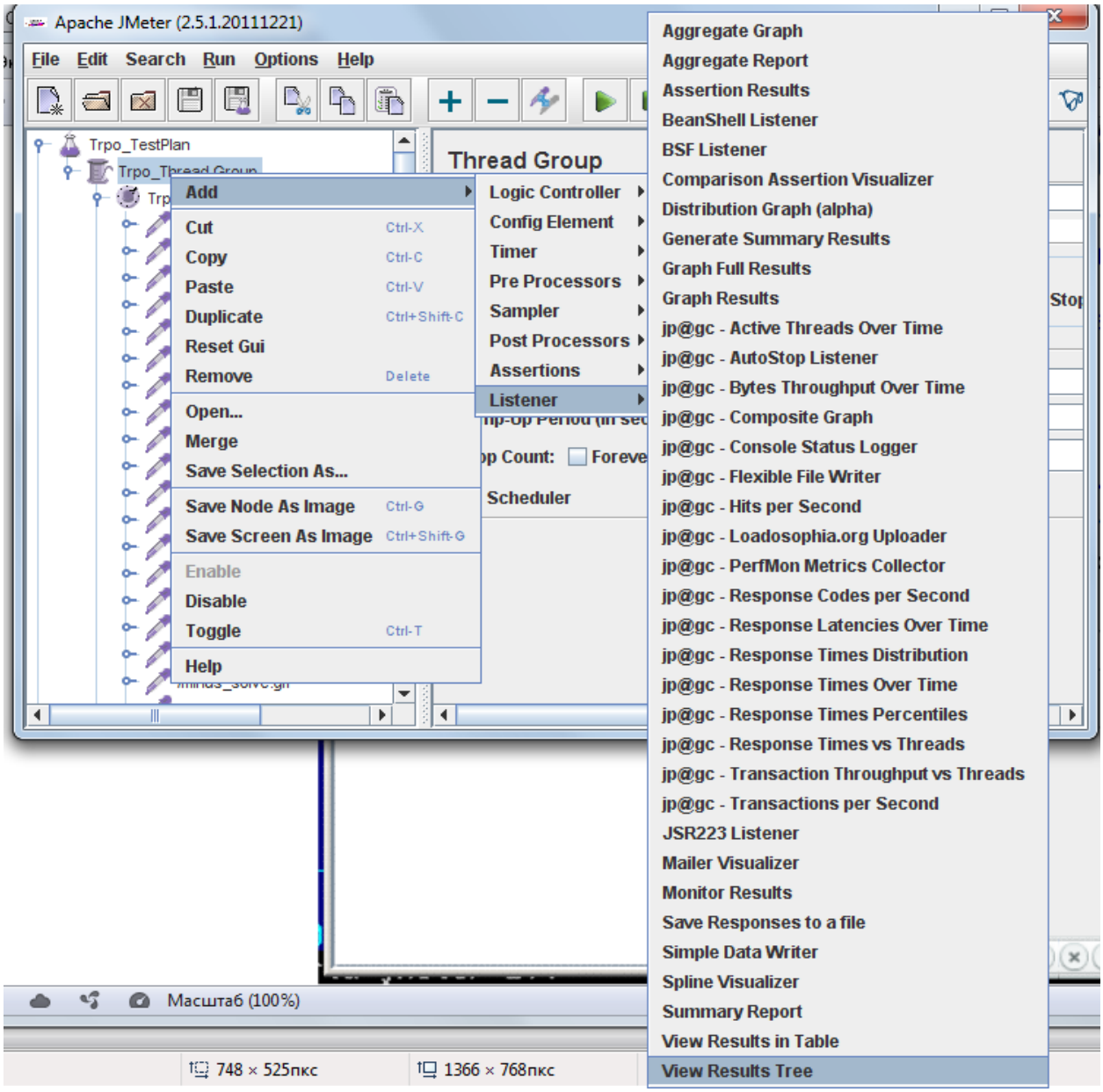

Рис.7 – Добавление Listener’ов

Теперь необходимо проанализировать все запросы, при необходимости некоторые запросы можно удалить, если, по вашему мнению, они не влияют на результаты тестирования (например, запросы рисунков). После этого можно отключить в браузере настройку прокси-сервера, а также остановить его в JMeter (кнопка Stop на вкладке элемента HTTP Proxy Server).

3. Для сбора информации о прохождении тестов в группу виртуальных пользователей необходимо добавить Listener'ы, например, View Results Tree - Trpo_ThreadGroup->Add->Listener->View Results Tree (рис.7). Данные элементы позволяют определить успешность тестов.

Примечание: Из-за маленького разрешения экрана пункт View Results Tree может не отображаться на экране. Для его выбора воспользуйтесь кнопками вверх-вниз на клавиатуре.

4. Сохраняем проект и запускаем сценарий, нажав на иконку Start (на панели инструментов JMeter – зеленый треугольник) или пункт меню Run->Start В результате Jmeter эмулирует действия, совершаемые пользователем в п.1. Меняя настройки (например, количество пользователей) можно сымитировать ситуацию одновременной работы на сайте нескольких человек, и, как результат, определить справляется ли сайт с заданной нагрузкой.

5. Результаты теста можно просмотреть в окне View Results Tree (рис.8).

При нажатии на запрос в правой панели отобразиться общая информация о запросе, сам запрос (вкладка Request), а также ответ на данный запрос (вкладка Response data). Запросы с ошибкой в окне View Results Tree отображаются красным цветом (ответом на такие запросы может быть страница с 404 ошибкой).

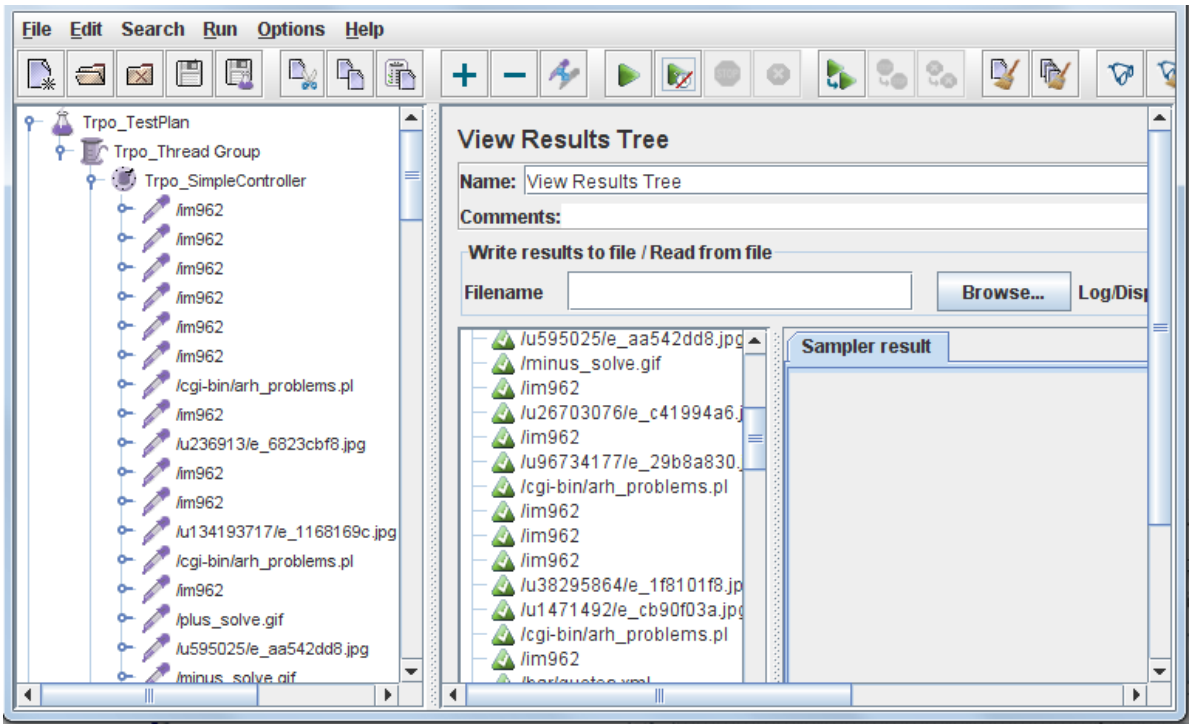

Рис.8 – Окно View Results Tree

4. ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

1. Ознакомиться с теоретической частью.
2. Настроить и запустить программу JMeter, ознакомиться с функциями этой программы и проделать предложенный пример.
3. Выполнить задание для самостоятельной работы в соответствии с вариантом, указанным преподавателем.
4. Составить отчет по выполненной работе.

5. ВАРИАНТЫ ЗАДАНИЙ

5.1. Задания для самостоятельной работы

1. Создание запроса

Напишите запрос для получения страницы текста определенной задачи в проверяющей системе самостоятельно. Для этого в блок Simple Controller (дерево задач, левая панель) добавьте новый элемент – HTTP Request: Simple Controller->Add->Sampler-> HTTP Request (рис.9).

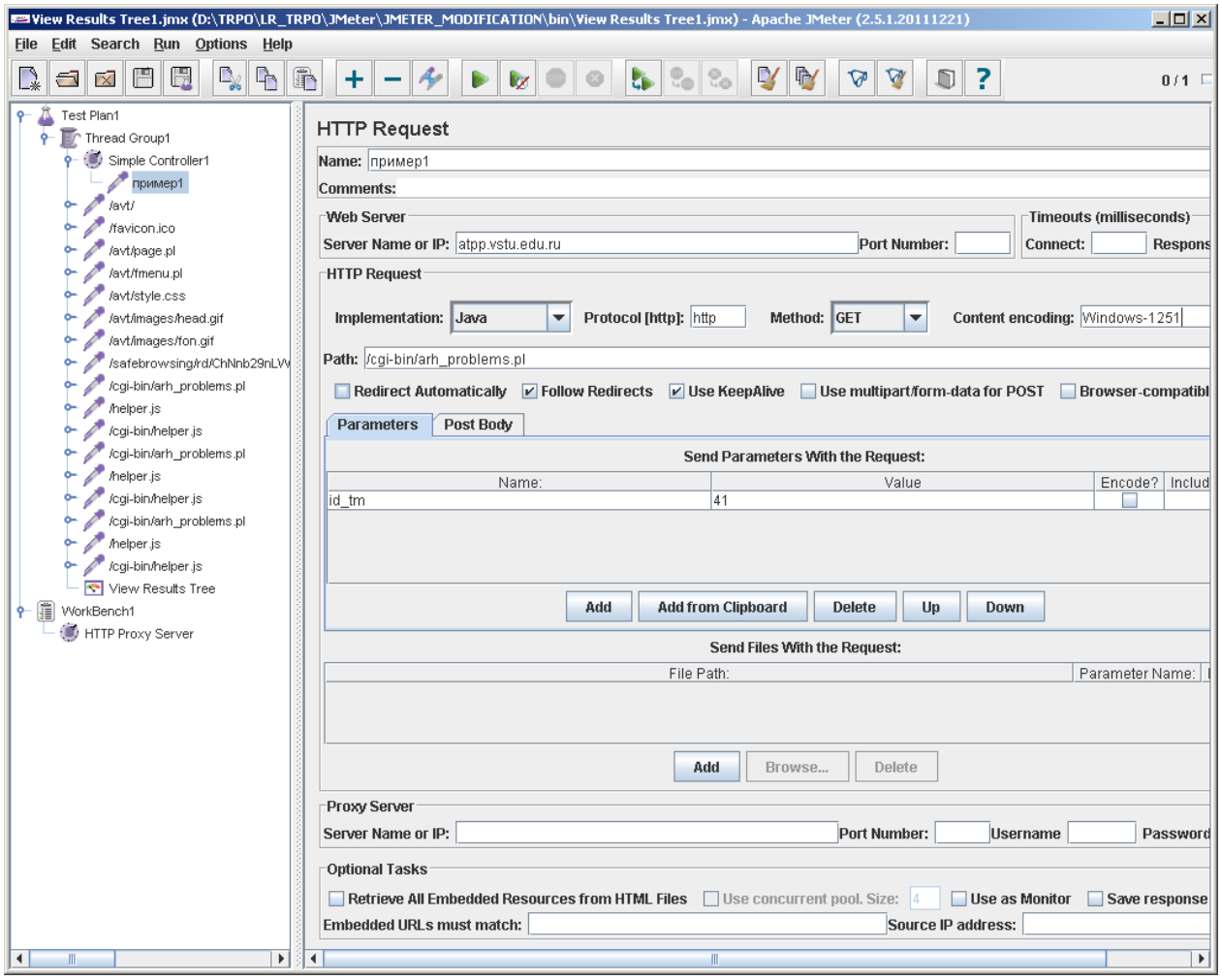

Рис.9 – Окно HTTP Request

Заполните поля запроса, по аналогии с полями запросов, сформированными прокси-сервером (дерево задач, левая панель: открывается при щелчке мыши). Добавьте посылаемые в запросе параметры в блок «Send Parameters With the Request» - заполните имя и значение параметра в поле name и value соответственно (для страницы текста задачи отправляемым параметром будет id_prb). Запустите тест и посмотрите его результат в View Results Tree. Убедитесь, что в ответ на ваш запрос вы получили корректную html страницу.

2. Циклы

Сделайте так, что бы ваш запрос выполнялся несколько раз.

Для этого правой кнопкой мыши щелкните по созданному запросу. В появившемся меню выберите Insert Parent->Logic Contriller-> Loop Controller, задайте число повторов запроса. Запустите тест и убедитесь, что ваш запрос выполнился заданное число раз.

3. Использование переменных в запросе:

В группу элементов Thread Group добавьте User Defined Variables (Thread Group ->Add->Config Element-> User Defined Variables) (рис10).

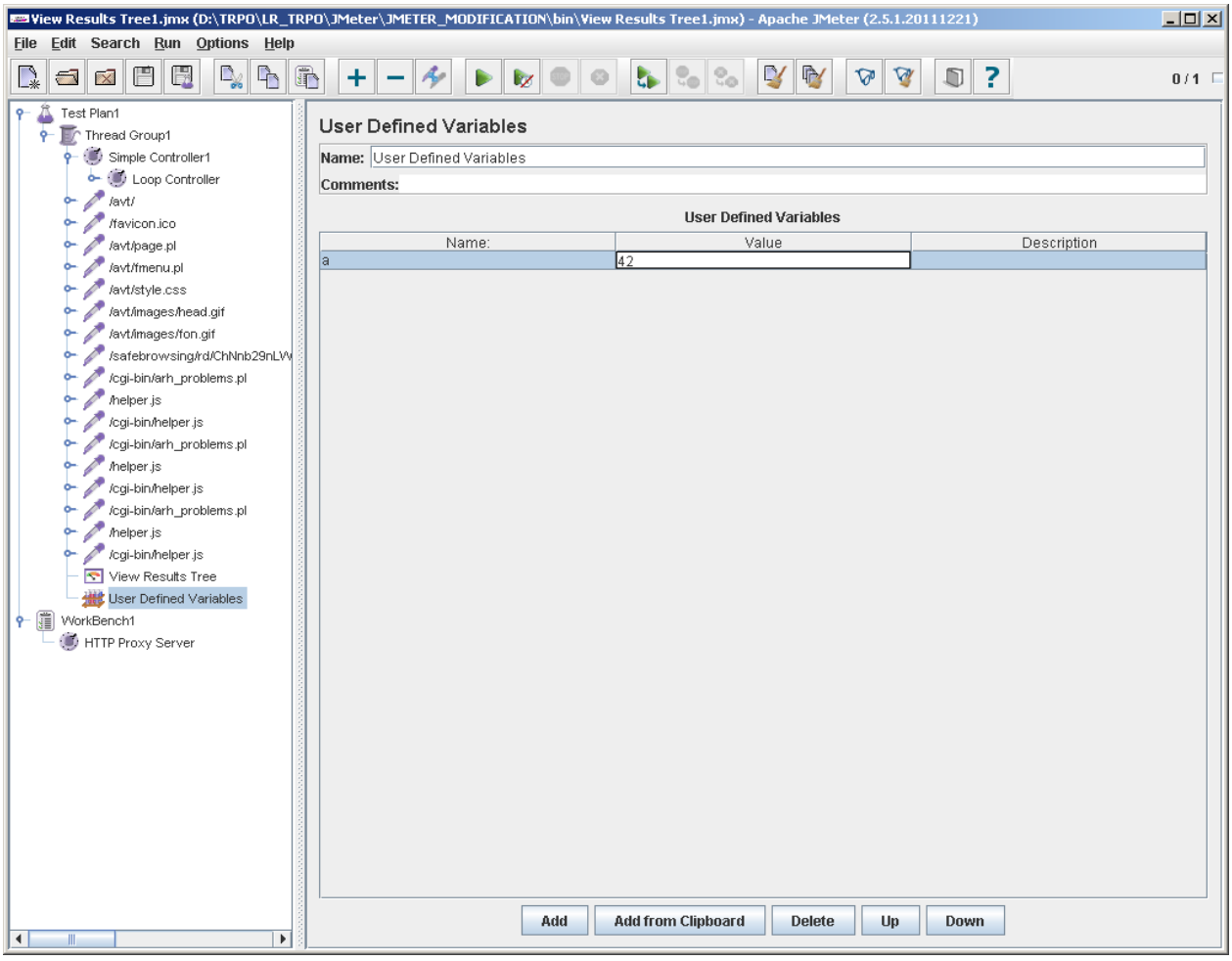

Рис.10 – Окно User Defined Variables

Добавьте в него название и значение переменной – пусть это будет идентификатор какой-нибудь задачи на сайте проверяющей системы (название переменной может быть любым, значение переменной – номер задачи, которая будет отправляться в запросе).

В добавленном запросе HTTP Request в качестве значения идентификатора задачи впишите название переменной в формате **${**название_переменной}(рис11).

Запустите тест и убедитесь, что в запросе отправилось значение указанной переменной.

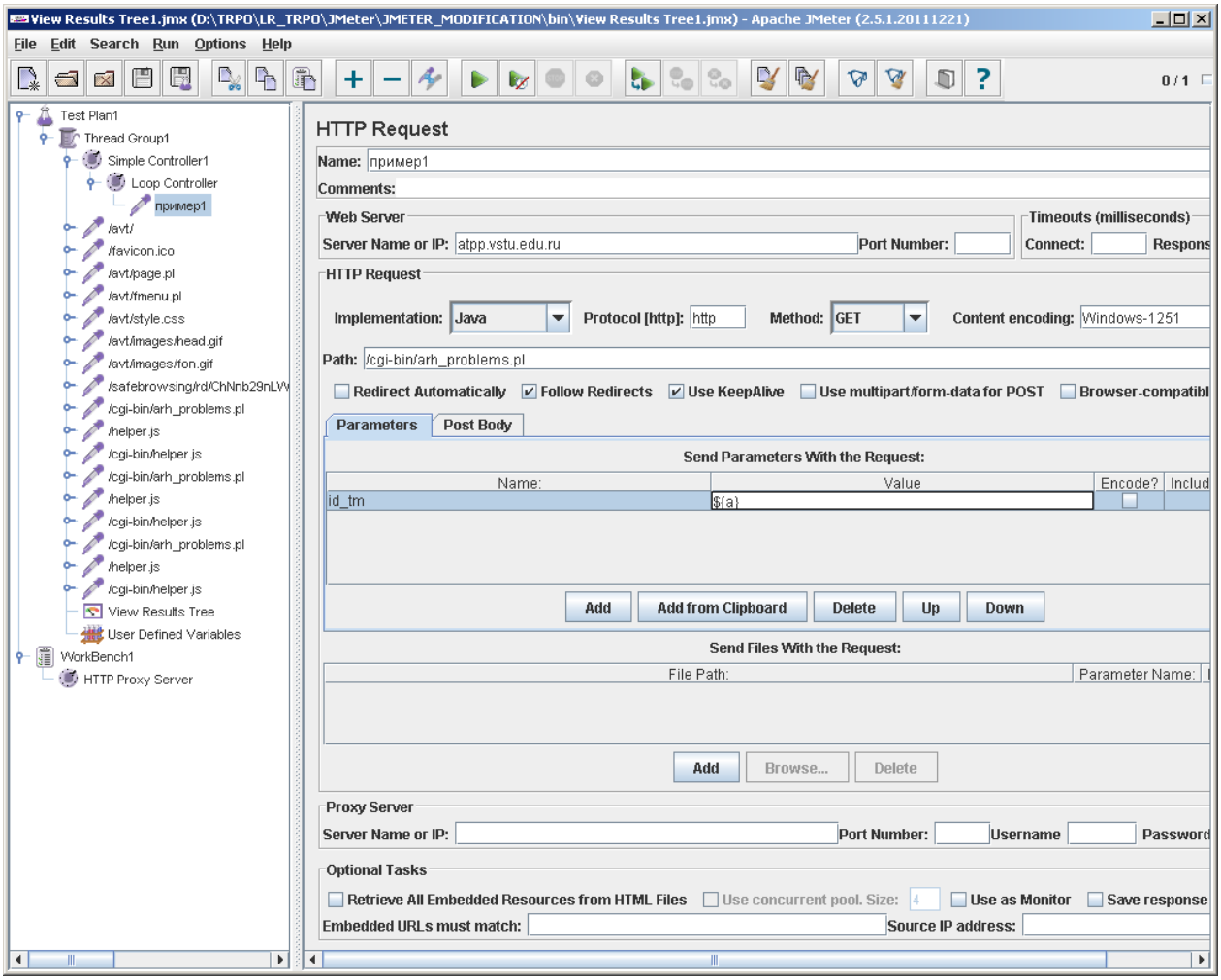

Рис.11

4. Задание переменных для разных потоков (пользователей):

В группу элементов Thread Group добавьте элемент User Parameters. Добавьте в него еще одного пользователя User_2, после чего добавьте название и значения новой переменной (по аналогии с предыдущим заданием). В качестве значений укажите разные идентификаторы задач для двух разных пользователей (потоков) – это сымитирует ситуацию просмотра разных задач двумя пользователями.

В созданном запросе HTTP Request в качестве значения параметра укажите название новой переменной (по аналогии с предыдущим заданием).

В настройках элемента Thread Group укажите 2х пользователей и запустите тест. Убедитесь, что в запросе для первого пользователя отправилось первое значение переменной, для второго – второе.

В настройках элемента Thread Group укажите 3х пользователей и посмотрите результат выполнения теста.

5. Выбор номера задачи из текстового файла:

Создайте txt-файл и перечислите несколько идентификаторов задач (каждый идентификатор с новой строки). В элемент Thread Group добавьте CSV Data Set Config (рис12). Впишите название текстового файла (с указанием расширения) и название переменной, в которой будут храниться номера задач. Убедитесь, что строка «Delimiter» не заполнена.

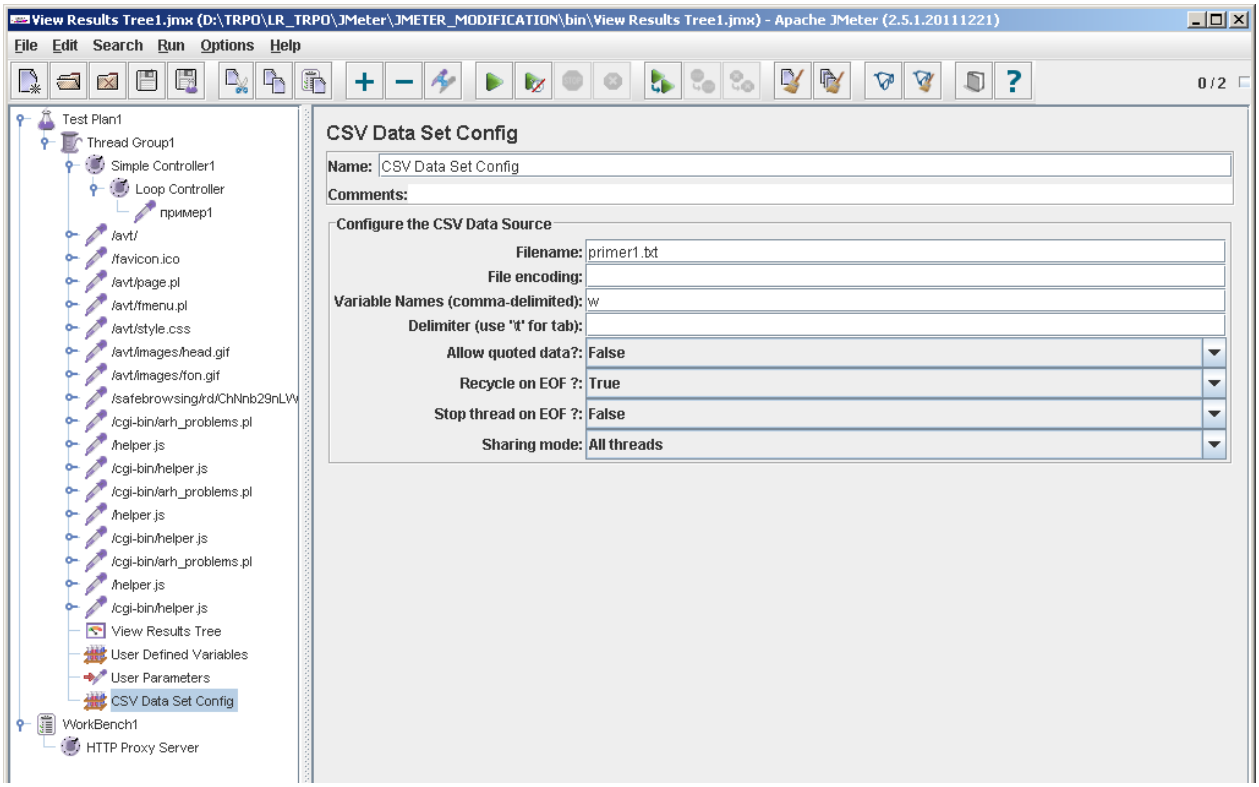

Рис.12 – Окно CSV Data Set Config

В добавленном запросе HTTP Request в качестве значения параметра впишите название переменной вида ${название_переменной}.

Запустите тест и убедитесь, что для первого пользователя в запросе отправилось значение первой переменной в указанном файле, для второго пользователя – значение второй переменной и т.д.

6. Таймеры

Для увеличения «реалистичности» теста необходимо добавить таймеры, которые будут управлять задержками между операциями. Для этого расставьте задержки (delay) в jmeter скрипте между логическими шагами.

Для этого добавьте в группу Thread Group таймер Gaussian Random Timer и запустите тест.

Увеличьте значения параметров на вкладке Gaussian Random Timer в 10 раз и снова запустите тест – убедитесь, что паузы между запросами стали больше.

7. Проверка корректности ответа

Для созданного запроса добавьте элемент Response Assertion.В блоке Patterns to test можно перечислить текст, который должен обязательно быть в ответе на запрос, что бы можно было сказать о правильности ответа.

Например, это может быть небольшая фраза из текста задачи.

Запустите тест – убедитесь, что в View Results Tree ваш запрос отображается зеленым цветом.

Снова зайдите в Response Assertion и укажите в блоке Patterns to test значение, которого не должно быть в ответе на запрос. Запустите тест – убедитесь, что ваш запрос отображается красным цветом. Посмотрите, что отображается на вкладке Response data вашего запроса.

8. Просмотр результатов на разных графиках и таблицах

Добавьте Listener-ы View Results in Table и Graph Results.

Проанализируйте результаты тестов по таблице и графику.

9*. Сохранение переменной

Значения, полученные в ответе на запрос можно сохранять в переменные и использовать эти переменные в следующих запросах. Найти нужные значения можно, например, с помощью регулярных выражений (подробнее о регулярных выражениях см. в папке «регулярные выражения»)

Внизу на странице текста задачи существуют ссылки на другие популярные задачи.

Необходимо сохранить значение параметра id_prb одной из таких ссылок в переменную и отправить эту переменную в следующем запросе – результатом должна быть страница с текстом задачи.

Для этого в запрос добавьте элемент Regular Expression Extractor.

В поле Reference введите имя переменной, в которую будет сохранено найденное в ответе значение. В поле Template впишите 1.

В поле Regular Expression введите выражение, при этом ту часть, которая должна сохраниться в переменной, заключите в круглые скобки.

Например, если в поле Regular Expression записать «id=(test)», то в случае если строка «id=test» будет найдена в тексте ответа, то в значение переменной запишется только «test».

После чего создайте еще один запрос HTTP Request, при этом в качестве значения идентификатора задачи введите название переменной регулярного выражения.

Запустите тест.

Убедитесь, что оба запроса прошли корректно и в их параметрах отправились верные идентификаторы задач.

Зайдите на страницу задачи первого запроса через браузер. Убедитесь, что ссылка на задачу, идентификатор которой отправился во втором запросе, присутствует на странице.

5.2. Список сайтов для нагрузочного тестирования

1 http://nazva.net/
2 http://eruditor.ru/z/
3 http://www.intuit.ru

6.ТРЕБОВАНИЯ К ОТЧЕТУ

Отчет должен содержать:

1. Цель работы.
2. Краткие теоретические сведения (1 стр.).
3. Задание.
4. Копии экранов с комментариями к выполненным действиям.
5. Выводы по работе. В выводах следует проанализировать полученные результаты тестирования.

7. КОНТРОЛЬНЫЕ ВОПРОСЫ

1. С какой целью проводится нагрузочное тестирование?
2. Перечислите основные показатели производительности.
3. В чем различие понятий "тестирование производительности" и "стрессовое тестирование "?
4. Что дает автоматизация нагрузочного тестирования?
5. Чем отличаются инструменты нагрузочного тестирования от средств автоматизации функционального тестирования?
6. Приведите примеры инструментов для нагрузочного тестирования.
7. Что относят к основным преимуществам JMeter?
8. Какие показатели производительности можно определить с помощью JMeter?
9. Какие настройки возможны в JMeter?