

Тема 6

«Тестирование производительности и нагрузочное тестирование»

Виды, направления и цели тестирования приложений под нагрузкой

Терминология

Прежде чем приступить к изучению определений и всей темы, предлагаю ознакомиться с базовой терминологией нагрузочного тестирования. Итак:

Виртуальный пользователь (virtual user) – процесс (программа), выполняющий некоторые заданные операции.

Итерация (iteration) – повтор операции.

Интенсивность выполнения операции (operation intensity) – частота выполнения операции (количество итераций) в единицу времени.

Нагрузка (load, loading) – совокупность выполняемых с системой операций.

Производительность (performance) – количество операций, выполняемых в единицу времени.

Масштабируемость (scalability) – способность системы увеличивать производительность пропорционально добавлению ресурсов.

Основные определения

Тестирование производительности (performance testing) – исследование «скоростных показателей» приложения при различной по характеру и количественных показателях нагрузке. *Наиболее общее определение, во многом покрывающее нижеследующие.*



Нагрузочное тестирование (load testing) – исследование способности приложения сохранять заданные показатели качества при нагрузке в допустимых пределах и некотором превышении этих пределов (определение «запаса прочности»). *Иногда выступает синонимом «тестирования производительности», но это не всегда правомерно.*



Стрессовое тестирование (stress testing) – (в контексте данной темы) исследование поведения приложения при «ненормальных» изменениях нагрузки (в нештатных условиях).



Объёмное тестирование (volume testing) – исследование производительности приложения при обработке различных объёмов данных.



Основные определения (пояснения)

Для лучшего понимания рассмотрим эти определения на примере автомобиля.

Тестирование производительности — с какой скоростью машина может ехать с 1-2-3-4-5-ю пассажирами, в гору, с горы, на разных видах топлива, как быстро она разгоняется и т.п.



Нагрузочное тестирование — может ли машина при некоторых установленных параметрах загрузки, топлива и т.п. разогнаться до 100 км/ч за указанное время и сохранять такую скорость на протяжении указанного маршрута. А если её перегрузить?



Стрессовое тестирование — зима, -70, если её завести и на непрогретом двигателе с 15-ю пассажирами подниматься в гору на 5-й передаче...



Объёмное тестирование — при каком количестве пассажиров (груза) начнутся проблемы со скоростными характеристиками.

Основные определения (пояснения 2)



И ещё раз – на примере принтера.

Тестирование производительности – за какое время принтер напечатает N страниц вот таких, таких и таких документов.

Нагрузочное тестирование – если на протяжении пяти часов непрерывно печатать...

Стрессовое тестирование – бумага со скрепками, кусок картона, 15000 одновременно посланных на печать документов (все с наивысшим приоритетом), печатать непрерывно трое суток...

Объёмное тестирование – послать на печать маленький txt-документ, 500-страничный doc-документ, 27-гигабайтный psd-документ.

Основные определения (пояснения 3)

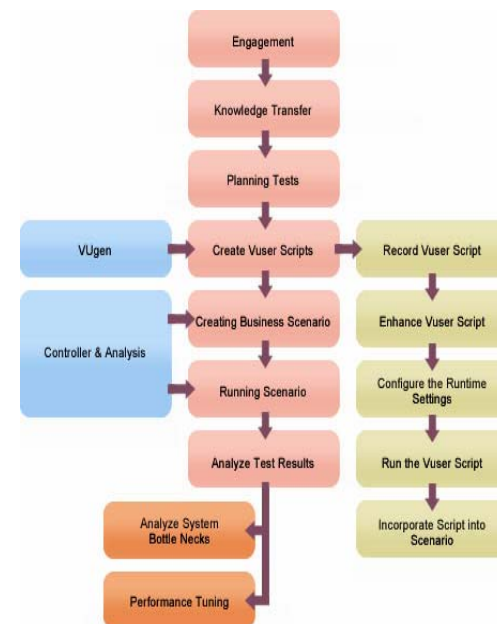
И ещё раз ☺ – на примере сайта.

Тестирование производительности – каково максимальное время генерации страницы при 10-20-500 пользователях, если их число резко меняется, если они выполняют такие-то действия...

Нагрузочное тестирование – заявленные в спецификации 100 пользователей одновременно сидят на сайте и выполняют некоторые действия... как ведёт себя сайт? А если пользователей будет 120?

Стрессовое тестирование – 500-1000-5000 пользователей, DoS/DDoS атака, наплыв спамеров...

Объёмное тестирование – в базе новостей 10000000 записей, на форуме 1000000 сообщений...



Цели...

Цели тестирования производительности:

оценка времени выполнения выбранных операций при определённой интенсивности и очередности выполнения этих операций;

оценка реакции приложения на изменение количества пользователей, одновременно работающих с приложением;

оценка границ интенсивности и видов нагрузки, при которых производительность приложения выходит за рамки приемлемой;

исследование производительности на низких, средних, высоких, предельных и стрессовых значениях нагрузки;

оценка показателей масштабируемости приложения.



Цели...



Цели нагрузочного тестирования:

оценка скорости реакции приложения **на различные значения нагрузки** в допустимых пределах;

оценка **использования** **приложением** **системных ресурсов** **при различных значениях нагрузки** **в допустимых пределах**;

оценка **изменения** **со временем поведения приложения** **при сохранении** **некоторой допустимой нагрузки** **длительное время**.

Цели...

Цели стрессового тестирования:

оценка реакции приложения на нестандартные и стрессовые случаи изменения нагрузки, в т.ч.:

- резкое непредсказуемое изменение интенсивности нагрузки;
- значительное превышение предельно допустимой нагрузки;
- интенсивное использование функций приложения, являющихся «узким местом» в производительности.



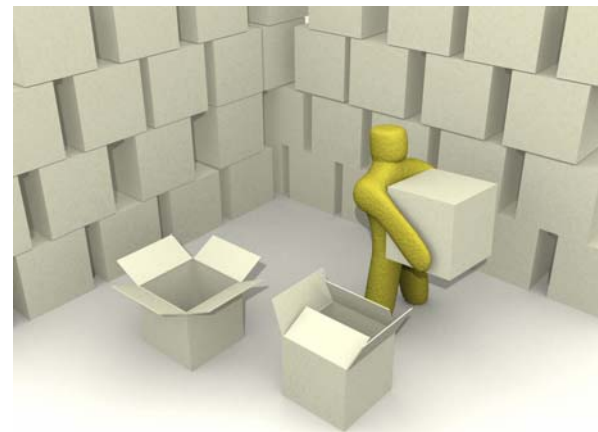
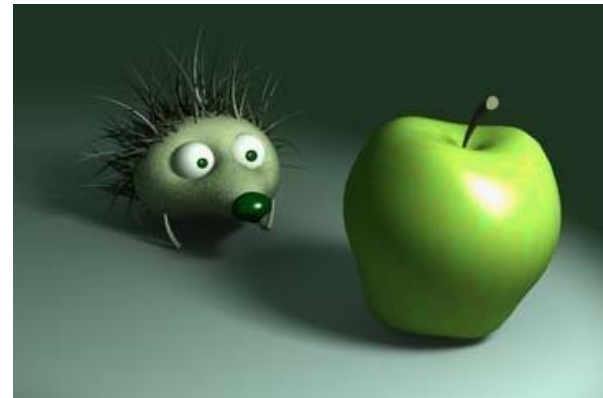
Цели...

Цели объёмного тестирования:

оценка показателей
производительности приложения в
случаях приёма, обработки и генерации
данных различного объёма и с
различными показателями
вычислительной сложности обработки;

оценка способности приложения
обрабатывать большие объёмы данных
в условиях высокой загрузки системных
вычислительных ресурсов;

оценка способности приложения
обрабатывать большие объёмы данных
при недостатке оперативной памяти.



Цели...

Цели любого тестирования, так или иначе затрагивающего производительность:

- определение **лучшей архитектуры** системы, выбор наилучшей **платформы, средств и языков** реализации;
- определение **оптимального способа хранения** файлов;
- оценка и **оптимизация схемы БД** в контексте повышения производительности;
- определение **узких мест** системы;
- оценка **максимальной и минимальной производительности** системы и условий их достижения;
- определение **характера увеличения времени отклика** системы при увеличении нагрузки;
- определение **максимального числа одновременно работающих пользователей**, превышение которого делает использование системы невозможным;
- определение **влияния конфигурации системы на производительность**;
- оценка показателей **масштабируемости системы**;
- оценка **архитектуры и настройки сети заказчика** требованиям производительности.



Почему это так важно

Важность тестов производительности объясняется просто: нет производительности – нет качества.

Низкую производительность замечает почти любой пользователь, что приводит к его недовольству.

Низкая производительность уменьшает количество полезных операций в единицу времени, что приносит убытки заказчику.

Низкая производительность может стать причиной выхода приложения из строя, проблем с безопасностью и т.п.

Низкая производительность может быть обусловлена некоторыми скрытыми причинами, влияющими и на остальные показатели качества системы (отказоустойчивости, восстанавливаемости, живучести и т.п.)



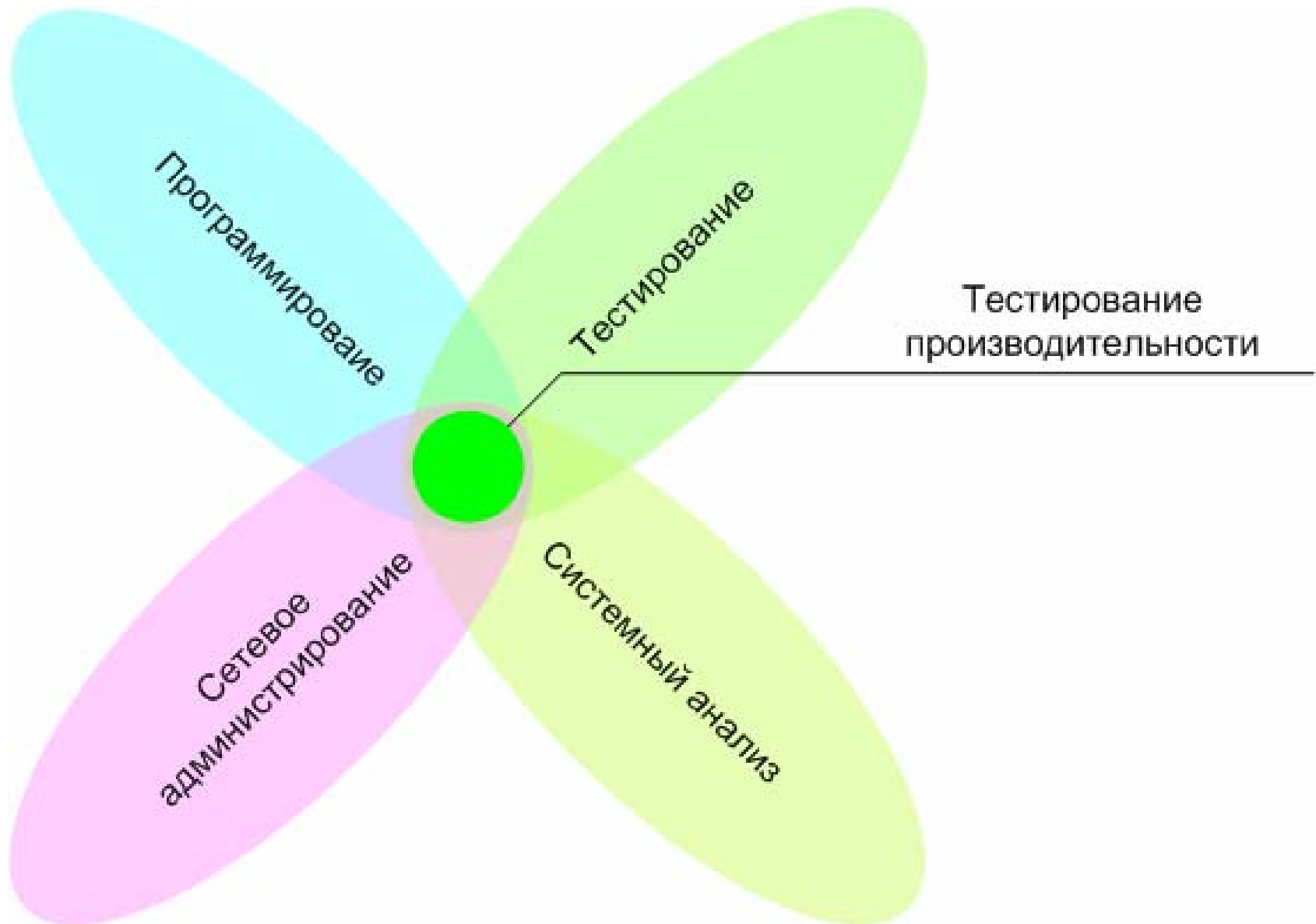
Необходимые навыки

При проведении тестов производительности (*) крайне желательно **обладать следующими навыками** (сочетать их все в одном специалисте сложно, потому хорошо бы, чтобы в команде были все соответствующие специалисты):

- **программирование** – разработка скриптов и устранение проблем в приложении;
- **тестирование** – грамотное проведение тестирования с чётко обозначенными целями и способами;
- **системный анализ** – эффективная обработка накопленных данных, формулировка выводов и рекомендаций;
- **сетевое администрирование** – решение вопросов организации работы некоторых тестов, анализ проблем с производительностью, вызванных настройками сети.

* Здесь и далее за основу материалов темы взяты разработки Владимира Марченко (“EPAM Systems”).

Необходимые навыки



Составляющие производительности

Производительность складывается из следующих характеристик:

- время отклика;
- мощность;
- стабильность;
- масштабируемость.



Множественность тестов

Провести лишь «пару тестов» для полноценной оценки производительности недостаточно.

Почему?

Потому, что показатели **производительности** **зависят от многих факторов**, которые можно полноценно учесть только при выполнении достаточно большого количества разнообразных тестов.



Основные тесты производительности



Наращивание нагрузки (ramp up) – позволяет определить т.н. «**точку насыщения**» (**saturation point**) – показатель нагрузки, при которой производительность системы начинает ухудшаться. Проведение нескольких ramp up тестов с увеличением доступных аппаратных ресурсов позволяет определить, насколько система **масштабируема (scalability)**.



Низко-, средне- и высоконагруженная работа (low-, mid-, high-load) – позволяет оценить время отклика (**response time and latency**) системы.



Тест на выживаемость (longevity test) – показывает способность системы работать длительное время под высокой нагрузкой.

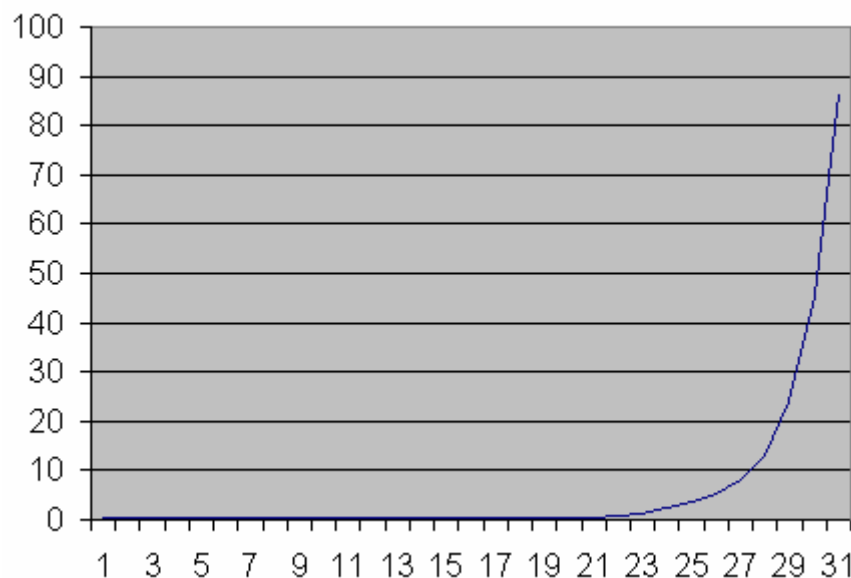


Тест «часа пик» (rush hour test) – позволяет оценить реакцию системы на резкое изменение нагрузки.

Ramp up test

Реальная история: скрипт, генерирующий дерево комментариев к статье, и основанный на многоуровневых JOIN-запросах в среднем обрабатывает за 0.5 секунды, если уровень вложенности комментариев не превышает 21. Затем время работы выглядит так:

- 22 уровня = 0.81 секунды
- 23 уровня = 1.12 секунды
- 24 уровня = 2.53 секунды
- 25 уровней = 3.52 секунды
- 26 уровней = 5.47 секунды
- 27 уровней = 8.05 секунды
- 28 уровней = 13.16 секунды
- 29 уровней = 24.15 секунды
- 30 уровней = 45.52 секунды
- 31 уровень = 86.61 секунды



Ramp up test

Можно ли сказать, что мы нашли точку насыщения?

Отчасти. Мы нашли точку насыщения для одного пользователя (отсутствуют конкурирующие запросы) и конкретного программного и аппаратного окружения. Для выяснения полной картины **следует провести тесты на разном оборудовании и при разном количестве одновременно работающих пользователей.**

Однако одно мы выяснили: на данном оборудовании алгоритм становится неработоспособным при уровне вложенности комментариев, превышающем 21.



Ramp up test

Ещё один пример. Система обрабатывает запрос одного пользователя за одну секунду, используя 10% системные ресурсы, – запросы двух пользователей – тоже за секунду, используя 20% системных ресурсов. И так до 10-ти пользователей, когда система всё ещё справляется за одну секунду, используя 100% ресурсов.

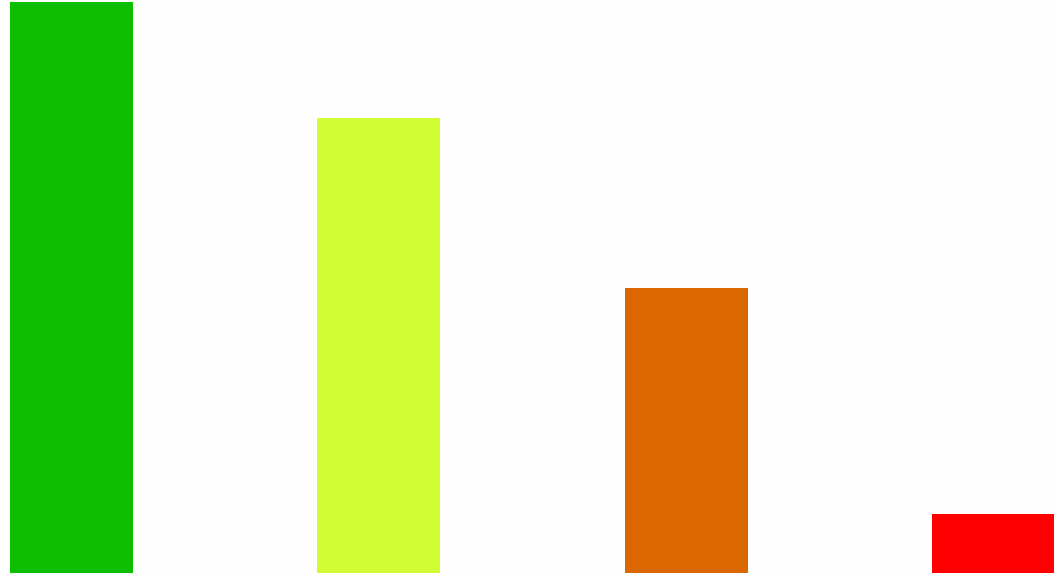
Но вот – одновременно пришло 11 запросов. **Что произойдёт?**



Ramp up test

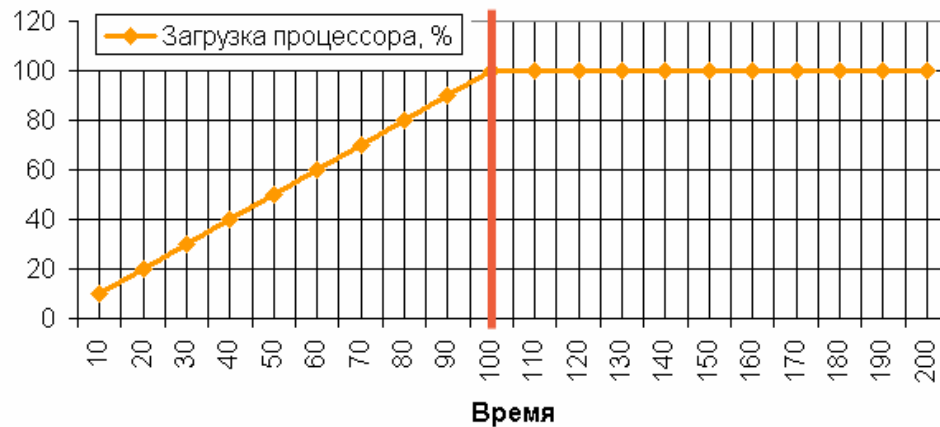
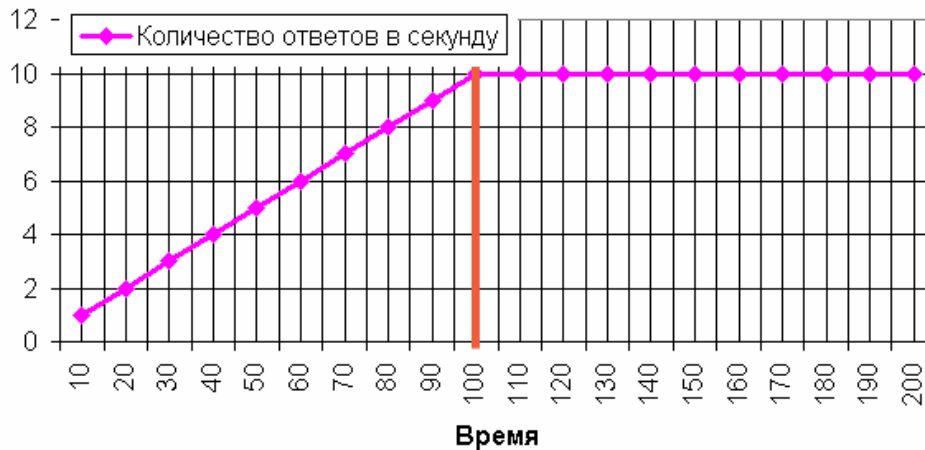
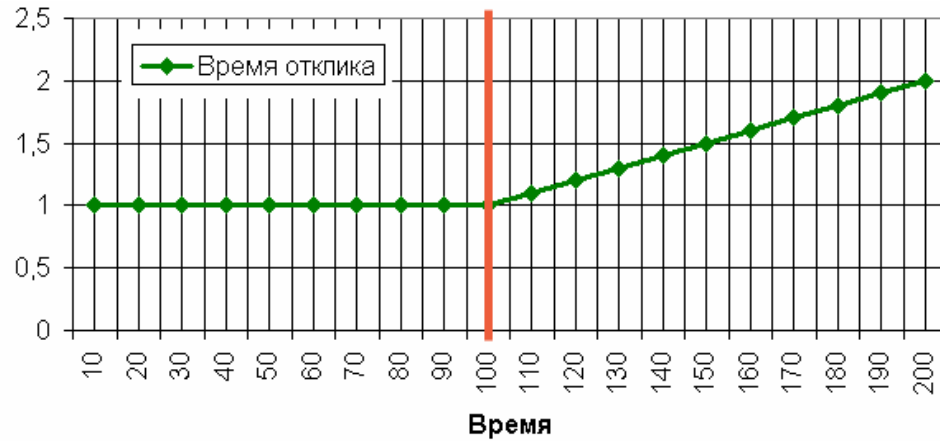
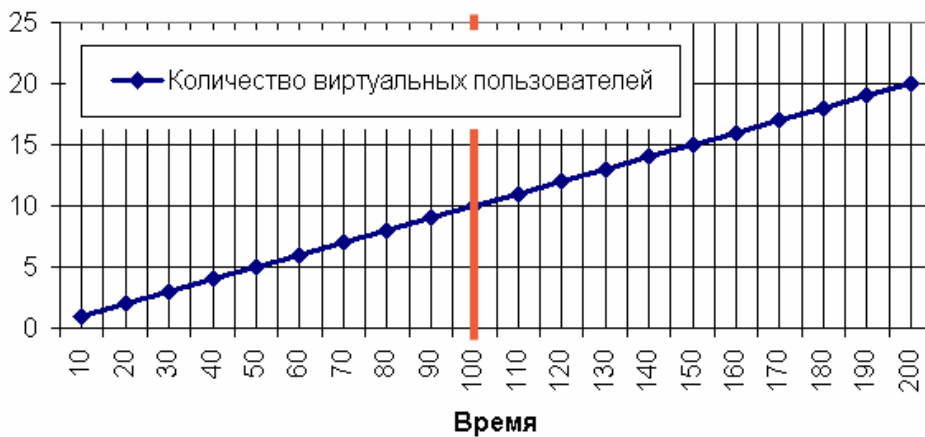
Итак, на примере этой гипотетической системы рассмотрим варианты поведения системы:

- **хороший**
- **приемлемый**
- **плохой**
- **ужасный**



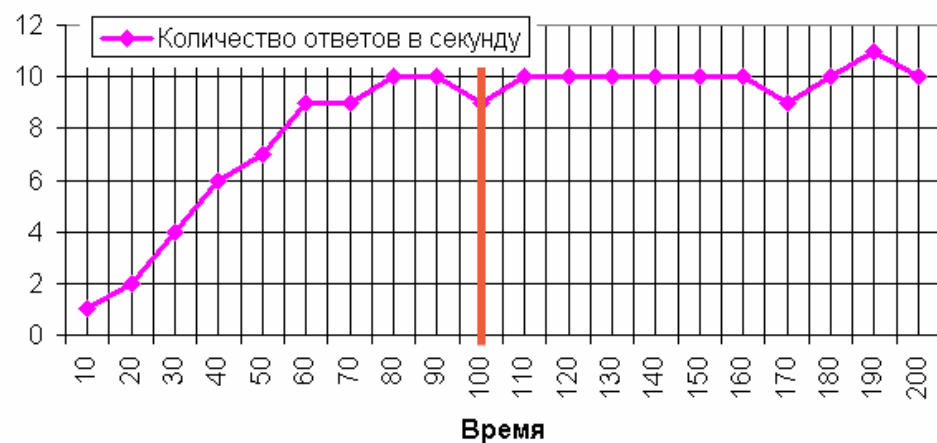
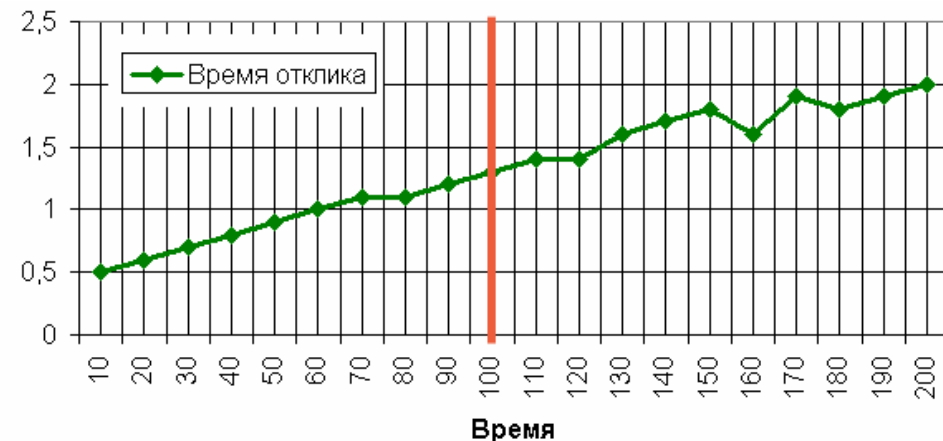
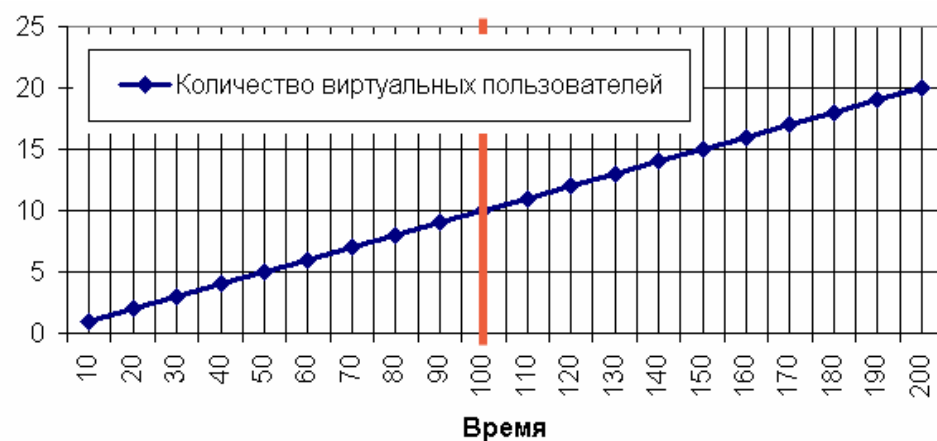
Ramp up test: хороший вариант

Увы, этот вариант стоит скорее называть «идеальным», т.к. в реальной жизни он не встречается.



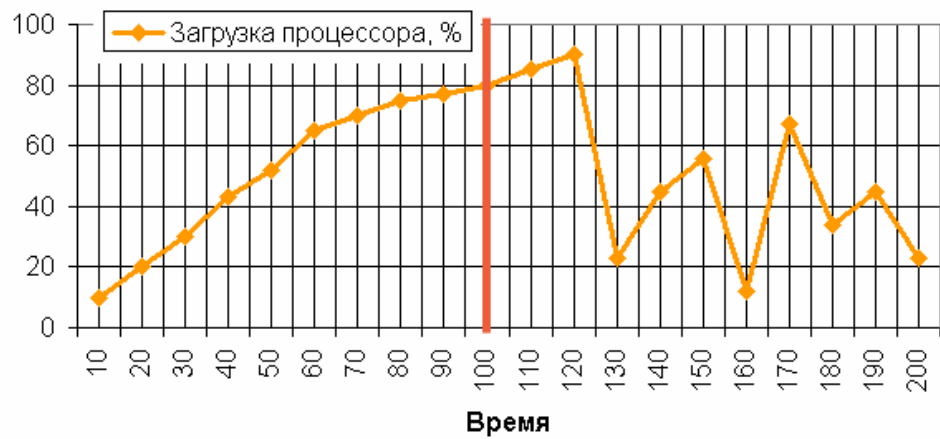
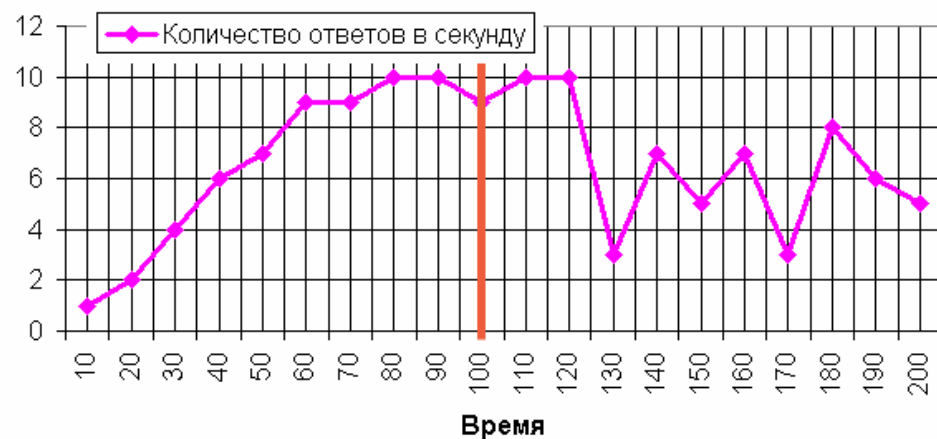
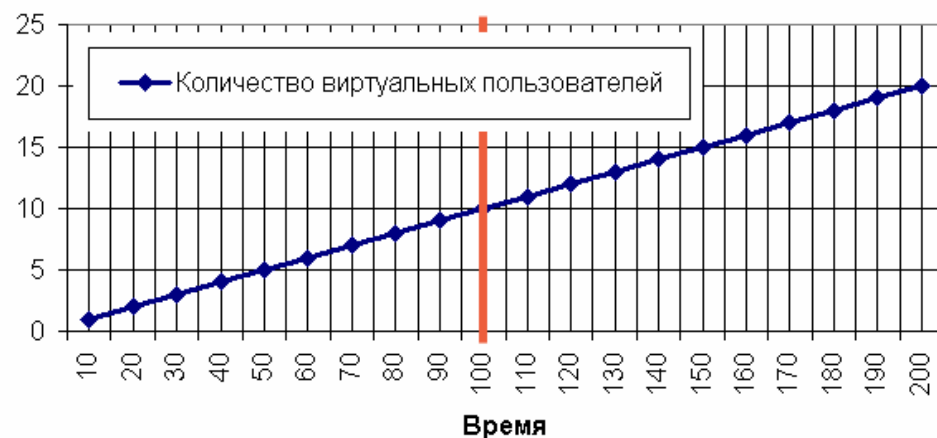
Ramp up test: приемлемый вариант

С такой системой уже можно работать. Она **оказывается достаточно устойчивой**.



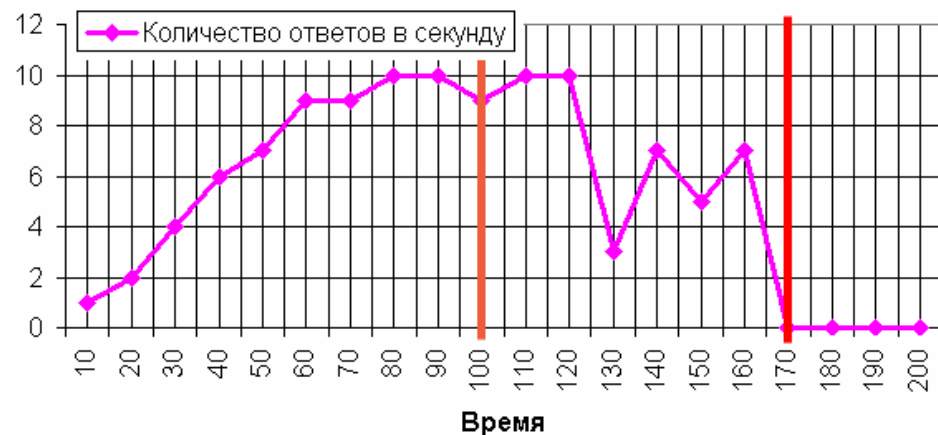
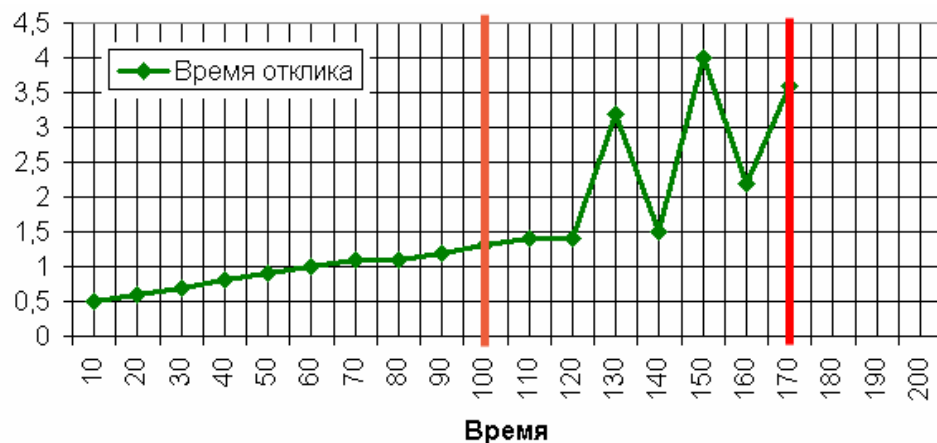
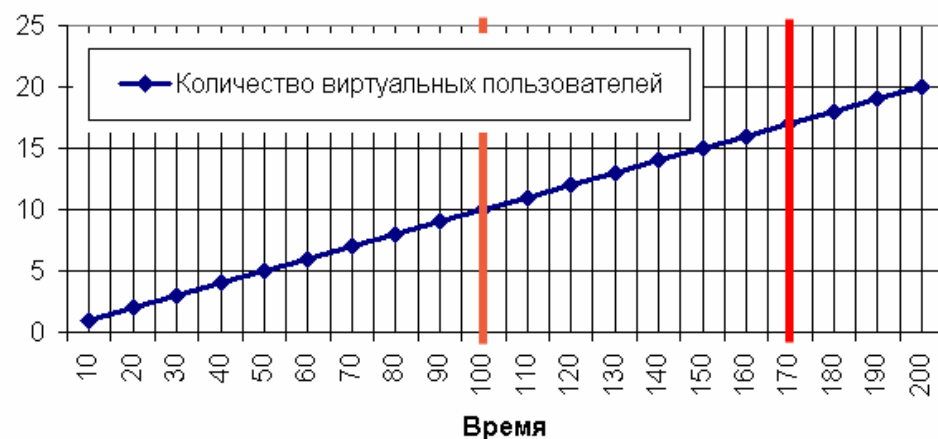
Ramp up test: плохой вариант

Вскоре после достижения точки насыщения **система становится нестабильной.**



Ramp up test: ужасный вариант

Вскоре после достижения точки насыщения **система**
выходит из строя.



Ramp up test: задачи

Итак, в процессе выполнения **ramp up** теста решаются следующие **задачи**:



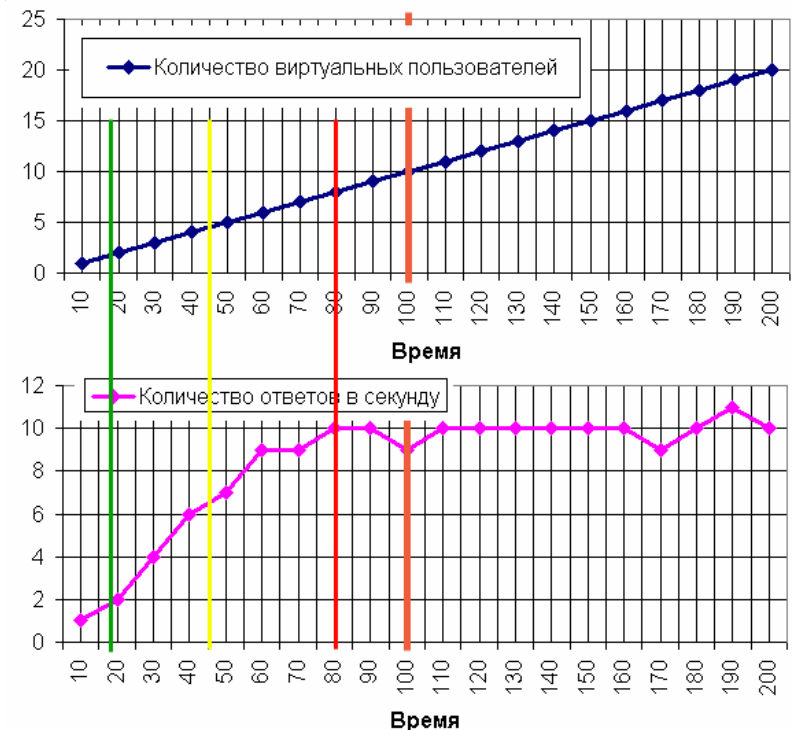
- поиск и **определение параметров точки насыщения**;
- определение **показателей масштабируемости** системы;
- определение **показателей нагрузки**, при которых:
 - система работает заведомо **стабильно**;
 - система работает **нестабильно**;
 - система гарантированно **выходит из строя**;
- определение **ключевых параметров аппаратной и сетевой конфигураций**, влияющих на производительность.
- определение **значений низкой, средней и высокой нагрузки** для low-, mid-, high-load теста.

Low-, mid-, high-load: пример

Возвращаясь к нашим четырём примерам систем рассмотрим «приемлемый вариант» и **определим значения низкой, средней и высокой нагрузки.**

Чаще всего (если говорить о количестве пользователей), берутся варианты в **10-15%, 40-45%, 80-85%** от количества пользователей, приводящих систему к точке насыщения.

Проводя тесты с этими значениями **на протяжении долгого времени**, мы получаем «тестирование на выживаемость» (**longevity test**).



Low-, mid-, high-load: задачи

Тестирование системы под различной нагрузкой позволяет **определить её реакцию на соответствующие условия** эксплуатации.

Для простых систем характерно увеличение числа проблем с увеличением нагрузки, однако для сложных систем это правило может не выполняться.

В любом случае, проводя подобные тесты мы **собираем статистические данные**:

- об **использовании аппаратных и сетевых ресурсов** и характере изменения этого использования;
- о **любых важных** средних, минимальных, максимальных **показателях** и их дисперсии.



Longevity test

Тесты на выживаемость системы показывают её способность **сохранять заданные показатели производительности, находясь длительное время под высокой нагрузкой.**

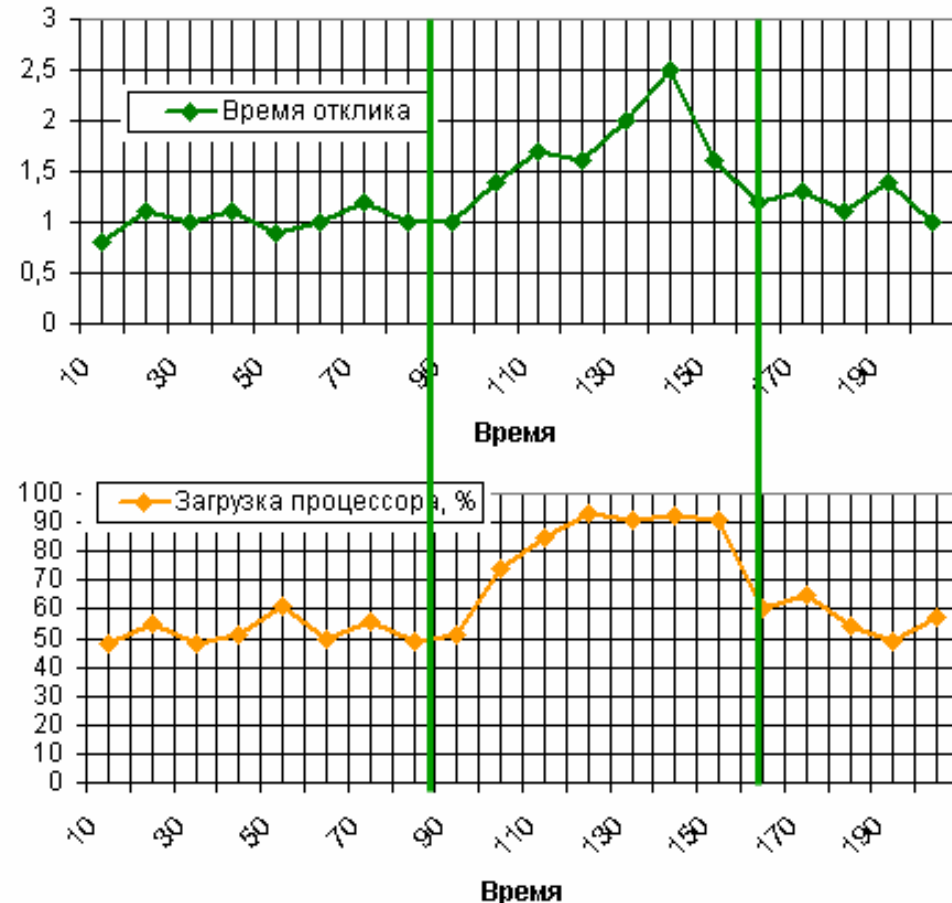
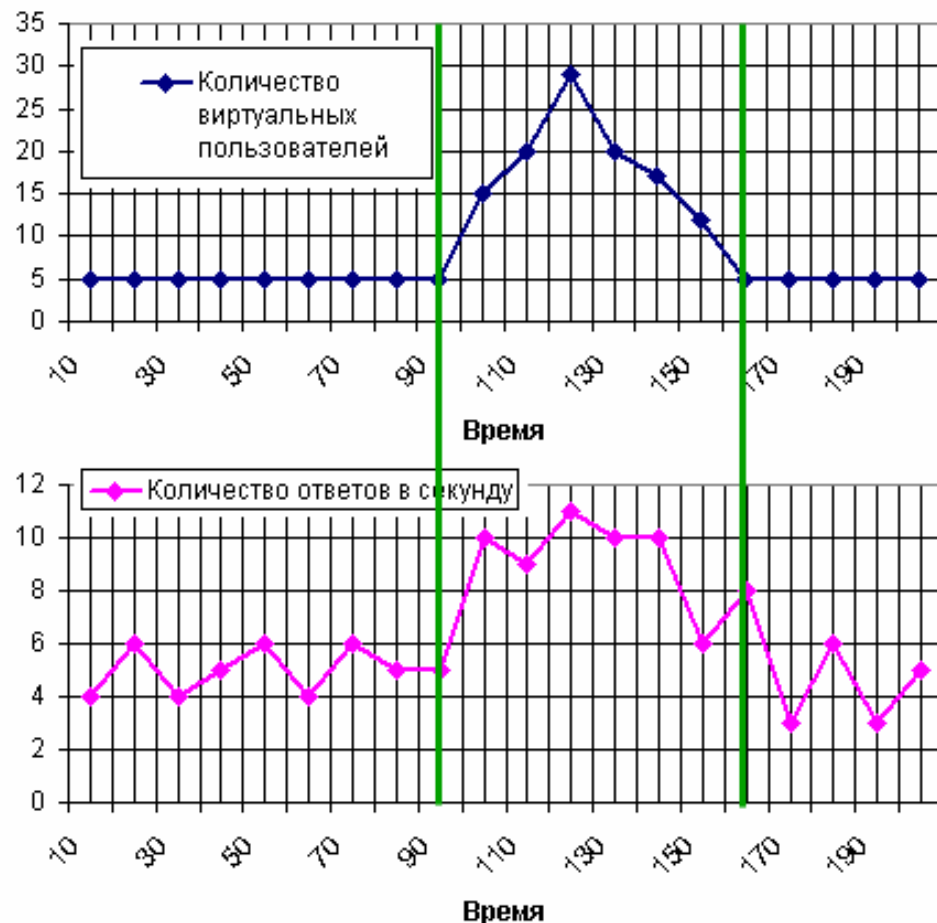
Как правило, для проведения этих тестов систему подвергают нагрузке, сопоставимой с той, что приводит к точке насыщения.

Продолжительность таких тестов – **несколько дней.**



Rush-hour test

Тест «**часа пик**» очень полезен в силу того, факта, что **он идеально отражает реальную жизнь**, когда в силу суточных колебаний интересов аудитории, календарных событий или DOS-атак нагрузка может очень резко изменяться.



Rush-hour test: задачи

Основная задача rush-hour теста – проверить **поведение системы ВО ВРЕМЯ** всплеска нагрузки и **ПСЛЕ НЕГО**.

Часто бывают случаи, когда заметное ухудшение наступает именно после того, как нагрузка вернётся к исходным расчётным показателям.

Данный тест также полезен для **исследования способности системы активировать свои «средства выживания под большой нагрузкой»** (использование кэширования, приоритизация обработки запросов, запуск резервных серверов, блокировка вредоносных запросов и т.п.) – все эти средства, будучи верно реализованными и использованными, позволяют сохранить производительность системы на приемлемом уровне.



Тест для проверки изученного

1. Что такое «тестирование производительности»?
2. Каковы основные задачи ramp-up теста?
3. Каковы цели стрессового тестирования?
4. Что исследует объёмное тестирование?
5. Почему тестирование производительности важно?
6. Из каких характеристик складывается производительность?
7. Что исследует rush hour тест?
8. Какую информацию о системе позволяет собрать longevity тест?
9. Под какой нагрузкой, обычно, проводится longevity тест? Как определить эту нагрузку?
10. Почему для исследования производительности системы следует проводить много разнообразных тестов?

Этапы проведения тестирования производительности

Основные этапы

Тестирование производительности, как и любое другое тестирование, подразумевает **организованную спланированную деятельность**, которая в данном случае включает следующие этапы:

- **сбор** необходимой **информации** о системе;
- **разработка модели** нагрузки;
- **выбор** инструментальных **средств**;
- **создание** и отладка **тестов**;
- **проведение тестирования**;
- **анализ результатов** и **формирование отчётности**.

PLAN FIRST!

Рассмотрим подробнее...

Сбор информации

Этот этап отчасти похож на этап сбора и анализа требований к продукту, однако здесь внимание уделяется только факторам, влияющим на производительность:

- работа с сетью;
- работа с БД;
- формирование интерфейса;
- кэширование;
- вычислительно сложные операции;
- узкие места (bottlenecks);
- обработка больших объёмов данных;
- и вообще – что это за система, как и кто с ней будет работать и какие проблемы с производительностью могут возникнуть.



Разработка модели нагрузки



Модель нагрузки – совокупность сценариев работы с системой, успешное выполнение которых зависит от показателей производительности.

Необходимо **определить**:

- **список тестируемых операций** (ЧТО конкретно мы будем проверять);
- **интенсивность выполнения операций** (нас будут интересовать операции, выполняемые наиболее интенсивно, и операции, сильнее всего влияющие на производительность);
- **зависимость изменения интенсивности** выполнения операций от времени (параллельное выполнение операций; ситуации типа «час пик» и т.п.)

Разработка модели нагрузки

При выборе подвергаемых тестированию операций следует руководствоваться **двумя показателями**:

- снижение производительности этих операций **уменьшает количество полезных действий** пользователя в единицу времени (например, банк может обслужить меньше клиентов);
- снижение производительности этих операций **ставит под угрозу работоспособность** системы.

С технической точки зрения – это операции, **интенсивно потребляющие аппаратные ресурсы**.



Разработка модели нагрузки

В контексте разработки модели нагрузки имеет смысл формировать **несколько профилей нагрузки**.

Профиль нагрузки – характерное для реальной жизни взаимодействие пользователей с системой, взаимозависимое с производительностью системы.

Например:

- «утро, все пришли на работу и зашли на наш новостной сайт» (ситуация «час-пик»);
- «стандартный день, ничего выдающегося»;
- «DDOS-атака»;
- «предпраздничная пора, тысячи покупателей ломанулись в наш интернет-магазин» (проверяется longevity тестированием).
- **ваши идеи?**



Разработка модели нагрузки



Ещё одним интересным решением при моделировании нагрузки является «пересчёт показателей», самым простым примером которого является уменьшение количества виртуальных пользователей, но увеличение интенсивности взаимодействия каждого пользователя с системой.

Т.е., например, 10 пользователей, выполняющих по 100 операций в секунду, создадут нагрузку, сопоставимую со 100 пользователями, выполняющими по 10 операций в секунду.

Однако следует учитывать тот факт, что часть ресурсов системы тратится как раз на «обслуживание отдельно взятого пользователя».

Выбор инструментальных средств

При выборе инструментальных средств для проведения тестирования производительности **следует учитывать следующие факторы:**

- способность средства **реализовывать необходимые профили и модели** нагрузки;
- **сложность формирования тестов** в среде инструментального средства;
- **форму отчётности**, предоставляемую инструментальным средством;
- **требования к аппаратной части** «генератора нагрузки»;
- **СТОИМОСТЬ** инструментального средства;
- **ВОЗМОЖНОСТЬ ИСПОЛЬЗОВАНИЯ ГОТОВЫХ РЕШЕНИЙ** или создания решений, которые можно будет использовать в последующих проектах.



Создание и отладка тестов



Этот этап наиболее зависим от того, **ЧТО и С ПОМОЩЬЮ ЧЕГО** мы собираемся тестировать.

В общем и целом, здесь нет ничего, что принципиально отличалось бы от других видов автоматизированного тестирования.

Следует **хорошо изучить инструментальное средство** и его «скриптовый язык», а также **все сопутствующие разработке тестов специфичные решения** и подход.

После чего тесты разрабатываются и выполняются под наблюдением тестировщика. В случае обнаружения ошибок – тесты корректируются.



Проведение тестирования и отчётность

Проведению тестирования посвящена большая часть нашей темы (особенно – раздел об использовании инструментальных средств), так что позволим себе здесь просто упомянуть, что такой этап существует.

Отчётности же посвящён отдельный (следующий) раздел...



Отчётность о тестировании производительности

Отчётность

После выполнения тестов **наступает время отчётности**, и здесь как никогда важно понимать, что **некачественный отчёт сводит к нулю результаты даже самого гениального тестирования**.



Что писать в отчёт

Отчёт – это ВЫВОДЫ, подкреплённые ФАКТАМИ.

Поэтому не имеет смысла просто приводить кучу цифр (которых может набраться на несколько гигабайт в виде лог-файлов).

Чтобы верно сформулировать выводы, перед проведением теста нужно чётко поставить **ЦЕЛЬ** теста (что мы ищем? зачем мы выполняем этот тест? о чём он нам расскажет?) и уже исходя из целей **делать выводы** о предмете исследования и **приводить РЕКОМЕНДАЦИИ** по улучшению ситуации, если всё не так хорошо, как нам хотелось бы.

Читаемость отчёта улучшается, если он **снабжён графиками, диаграммами и прочими средствами визуализации**, однако и они тоже должны отвечать требованию полезности.

Рассмотрим на примерах...



Что писать в отчёт: примеры

1) Итак, мы решили провести самый первый тест: *«Запустим 250 виртуальных пользователей на три часа.»*
Что это нам даст? НИ-ЧЕ-ГО. Здесь нет чёткой цели, нет поставленной задачи, выводы написать невозможно.

2) Мы решили искать точку насыщения. (Уже хорошо!)
Взяли типичное для данной системы оборудование и провели тест по всем правилам. Что мы пишем в отчёт? **Параметры точки насыщения.** Здесь не важны «средние времена отклика», «средние нагрузки на процессор» и т.п. (помните, что средняя температура по больнице и моргу вполне может быть 36.6 градусов).

Что писать в отчёт: примеры

3) Если мы проводим **rush-hour тест**, нужно описать **поведение системы во всех ключевых моментах этого теста**, обратив особое внимание на все «подозрительные» отклонения от нормы.

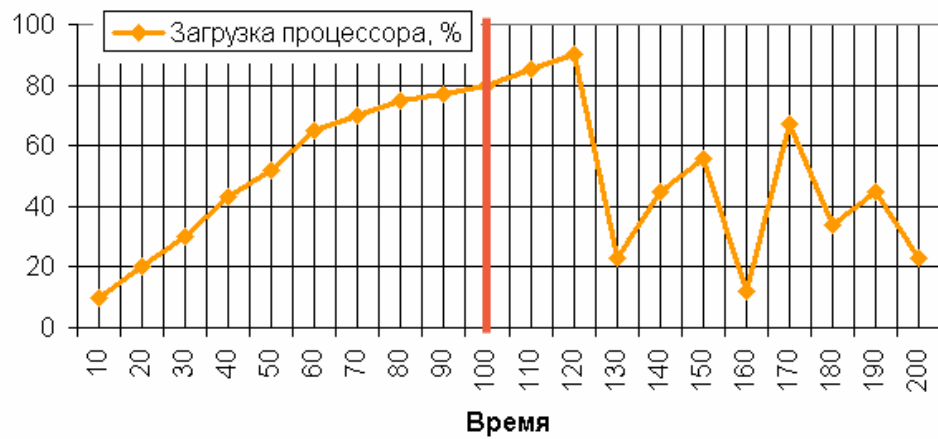
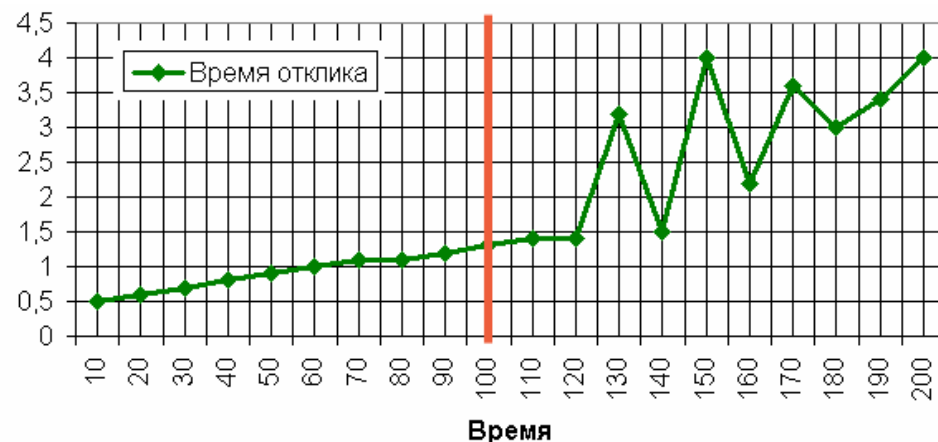
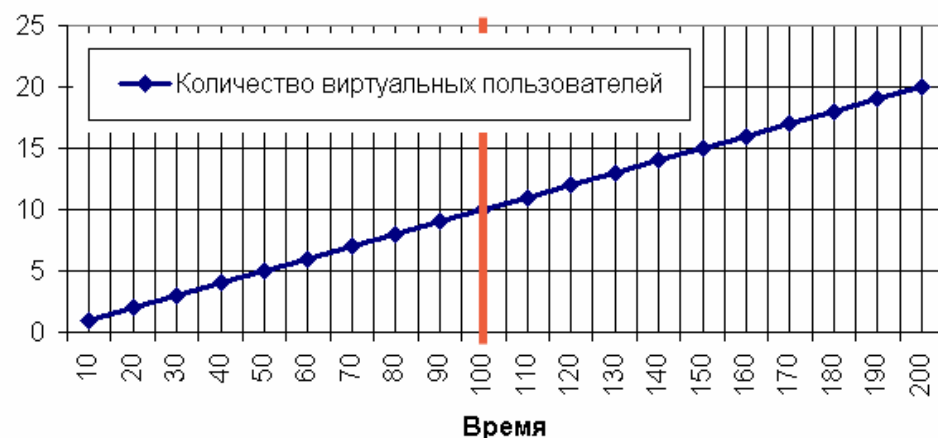
4) Если мы проводим **longevity тест**, то нас будет интересовать: **как долго система смогла выдерживать нагрузку**; как изменялась производительность системы, были ли утечки памяти, как быстро система «съедала» дисковое пространство и т.п. – т.е. всё то, что в конечном итоге «убило бы» систему.

5) Давайте придумаем ещё примеров...



Отчётность: быстрая практика

Давайте вспомним пример «**плохого случая**» ramp up теста. Как вы можете отразить это в отчёте? У вас 10 минут ☺



Тест для проверки изученного

1. Каковы основные этапы тестирования производительности?
2. Какую информацию о системе следует собрать до начала тестирования производительности?
3. Что такое модель нагрузки?
4. Что такое профиль нагрузки? Приведите примеры.
5. Что следует учитывать при выборе инструментальных средств тестирования производительности?
6. На что следует обращать особое внимание при подготовке отчёта о тестировании производительности?
7. Какая основная информация будет представлена в отчёте о group ир тесте?
8. Что такое (в контексте тестирования производительности) «узкое место»?
9. На какие функции системы следует обращать особое внимание при проведении тестирования производительности?
10. Существуют ли системы, для которых не имеет смысла проводить rush hour тест?

Инструментальные средства тестирования производительности

Коммерческие и некоммерческие средства

Наиболее известные **коммерческие** **инструментальные средства** тестирования производительности:

- HP Performance Center (+ HP LoadRunner);
- Rational Performance Tester;
- SilkPerformer;
- TestComplete.



- Наиболее известные **некоммерческие**:
- Apache **JMeter** (его мы сейчас и рассмотрим!)
 - Grinder.



Да, возможности коммерческих средств несопоставимо выше, но цена некоторых из них превышает цену квартиры в элитном районе.

JMeter – описание

JMeter – инструмент для проведения нагрузочного тестирования, разрабатываемый Apache Jakarta Project.

- Способен проводить нагрузочные тесты для JDBC-соединений, FTP, LDAP, SOAP, JMS, POP3, IMAP, HTTP и TCP.
- Интересна возможность создания большого количества запросов с помощью нескольких компьютеров при управлении этим процессом с одного из них.
- Архитектура, поддерживающая плагины сторонних разработчиков, позволяет дополнять инструмент новыми функциями.
- В программе реализованы механизмы авторизации виртуальных пользователей, поддерживаются пользовательские сеансы (сессии).
- Организовано протоколирование результатов теста и разнообразная визуализация результатов в виде диаграмм, таблиц и т. п.

JMeter – установка

Для использования JMeter вам понадобится:

- **JRE** (Java Runtime Environment) – среда для выполнения Java-приложений (да, JMeter написан на Java). Как правило, вопрос решается более глобально с установкой JDK (Java Development Kit) – полноценного набора инструментов для компиляции Java-приложений.



Загрузить JRE и JDK можно отсюда:

<http://www.oracle.com/technetwork/java/javase/downloads/>

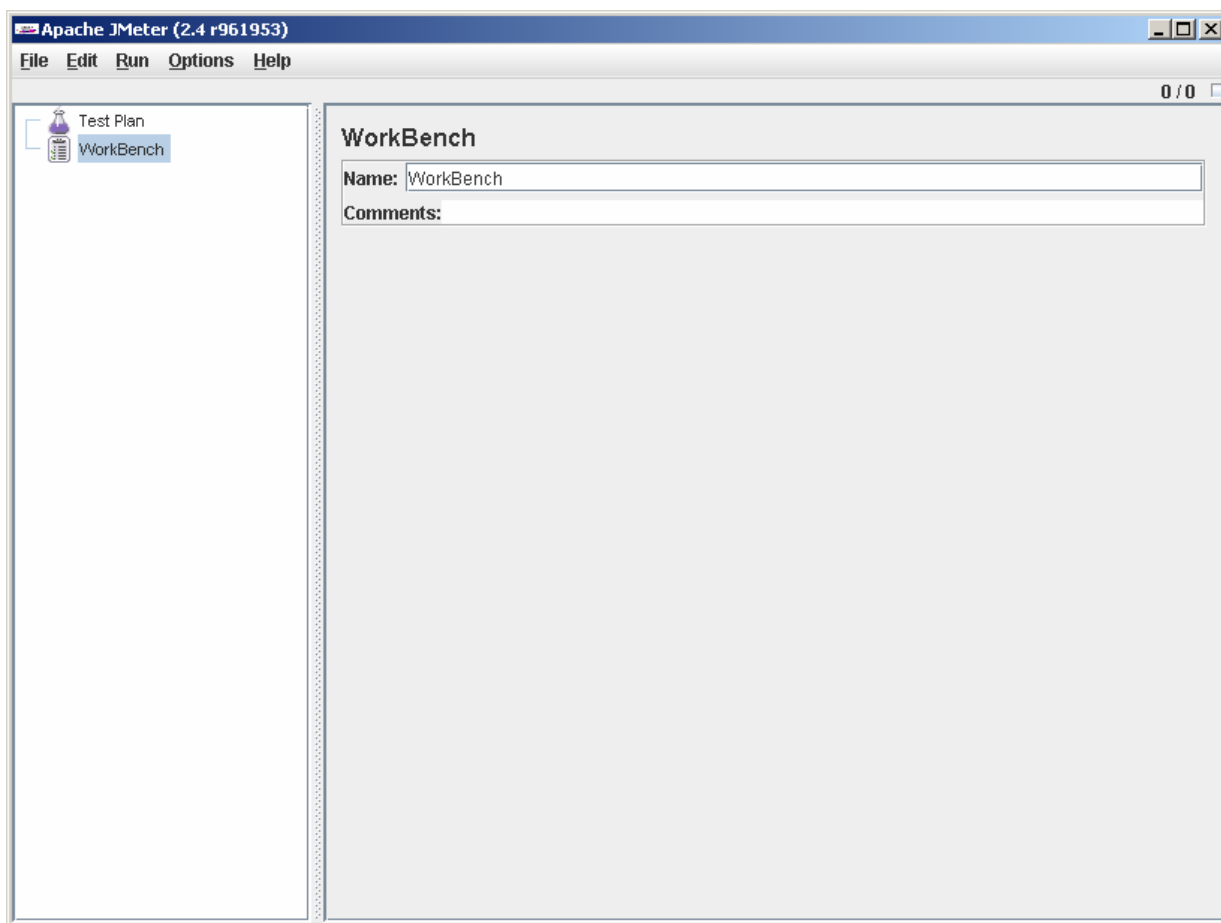
- Сам **JMeter**, который можно загрузить отсюда:
<http://jakarta.apache.org/jmeter/>



JMeter – установка

После установки JRE (или JDK) JMeter **достаточно просто распаковать и запустить**.

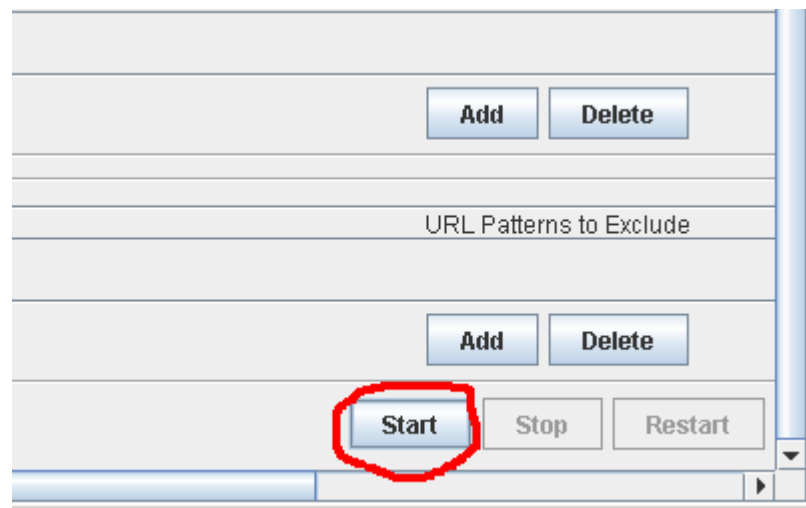
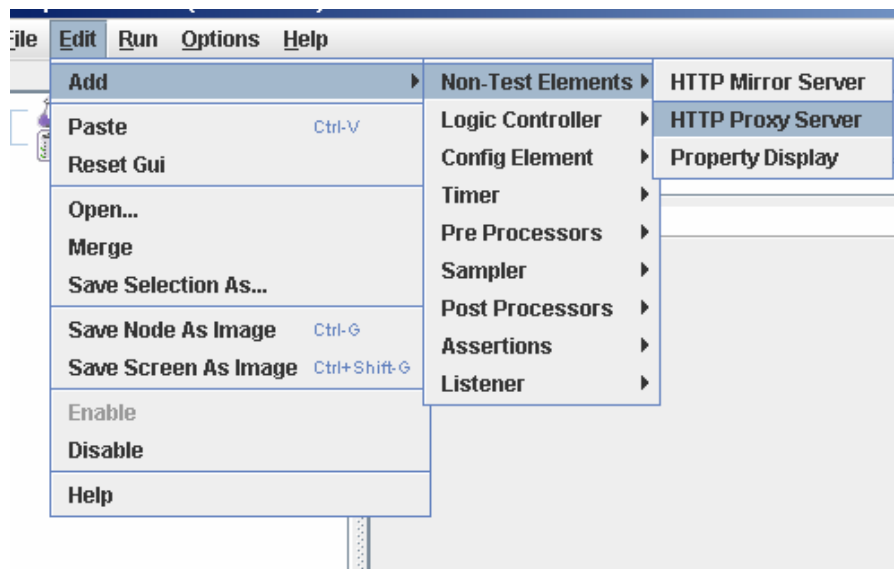
Вы увидите примерно такое окно.



JMeter – основные идеи

JMeter позволяет записывать «скелеты тестов» с применением технологии **Record And Playback**.

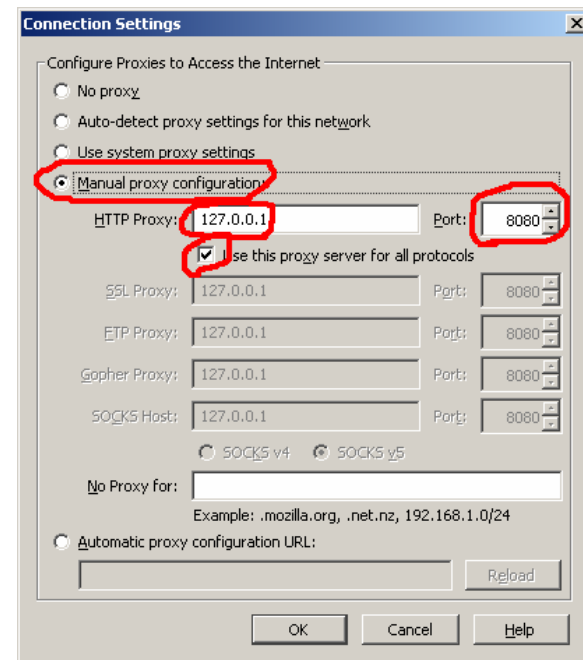
Для использования этой технологии в JMeter необходимо включить прокси-сервер: в меню слева выберите «Workbench», а затем в главном меню «Edit» → «Add» → «Non-Test Elements» → «HTTP Proxy Server». **Нажмите «Start»** внизу появившейся страницы.



JMeter – основные идеи

Настройте свой браузер на использование только что созданного прокси-сервера (если вы не меняли параметры по умолчанию, то это будет: адрес «127.0.0.1» порт «8080»).

Сейчас вы можете выполнять с тестируемым веб-приложением любые действия – JMeter их запишет.

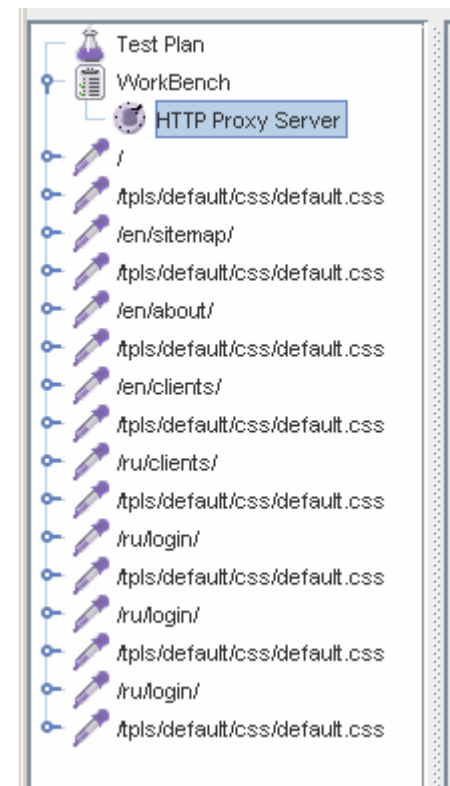


JMeter – основные идеи

После записи теста не забудьте:

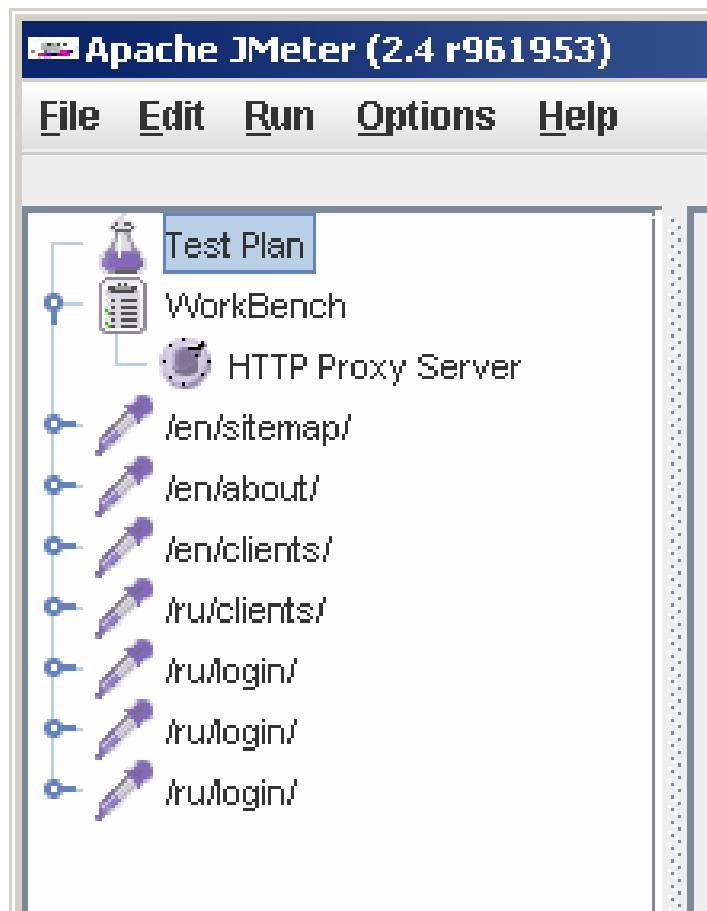
1. Остановить прокси-сервер JMeter'а.
2. Вернуть настройки браузера в состояние, в котором они находились ДО того, как вы перенастроили браузер на использование прокси JMeter'а.

Сейчас у вас записан ваш первый тест:



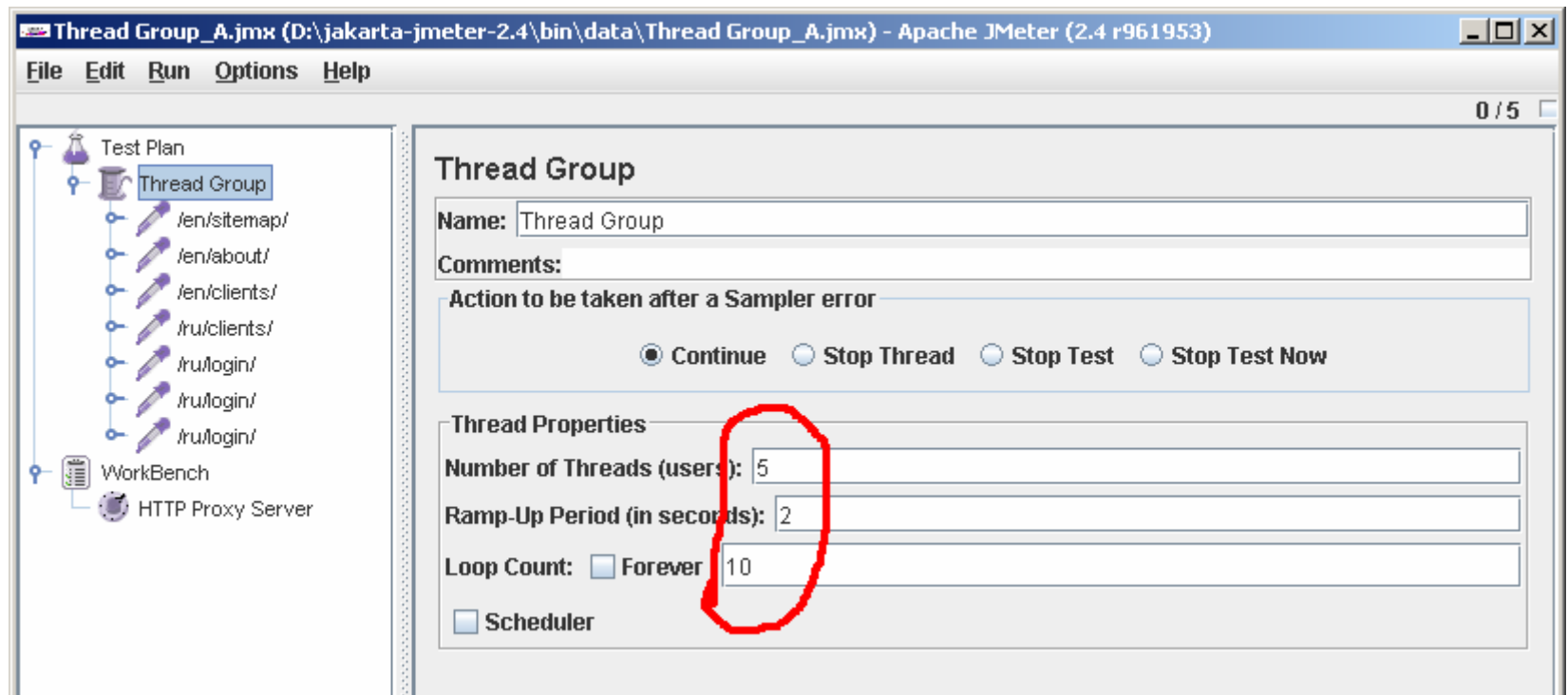
JMeter – основные идеи

Если в тесте оказались записаны «лишние действия», удалите их. Тест должен принять **аккуратный законченный вид**.



JMeter – создание пользователей

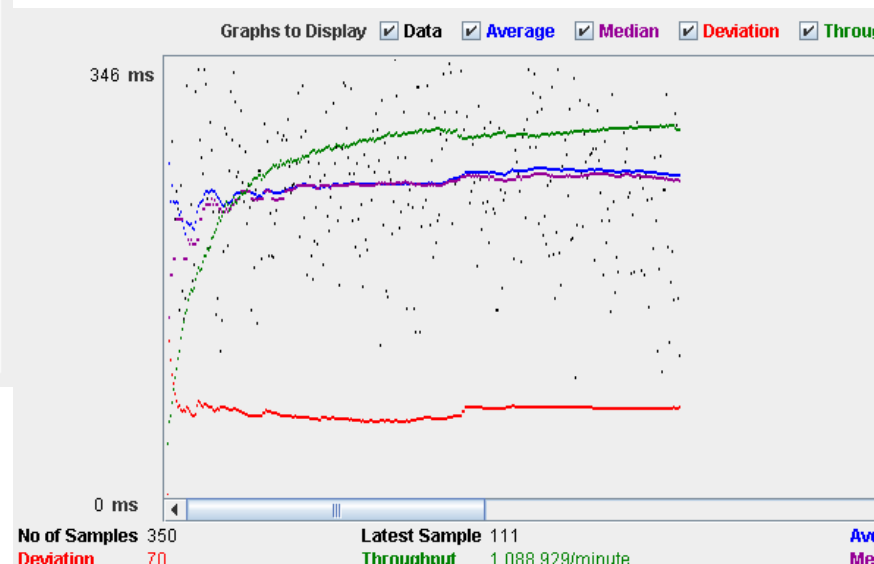
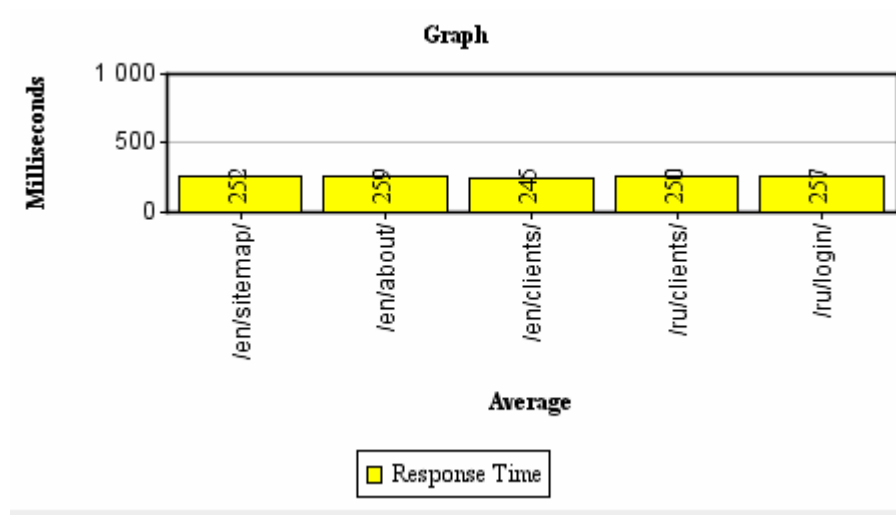
Выбрав в левом меню «Test plan» в главном меню выберите «Edit» → «Add» → «Therads (users)», создайте и настройте **группу пользователей**.



JMeter – создание отчёта

Остаётся только добавить («Edit» → «Add» → «Listeners») в ваш тестовый план **инструмент сбора статистики**. **Все инструменты собирают одинаковые данные**, но по-разному их отображают.

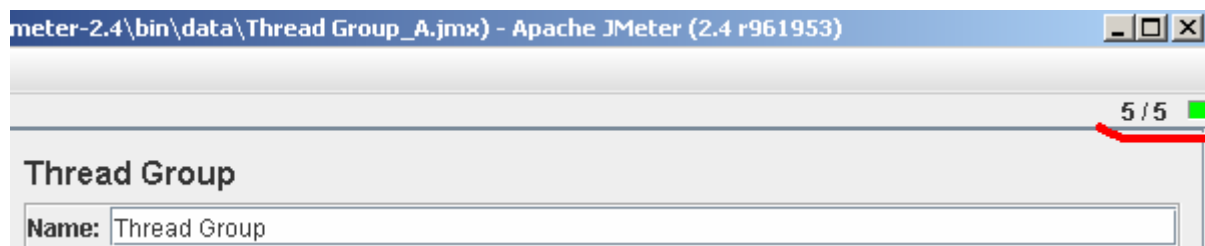
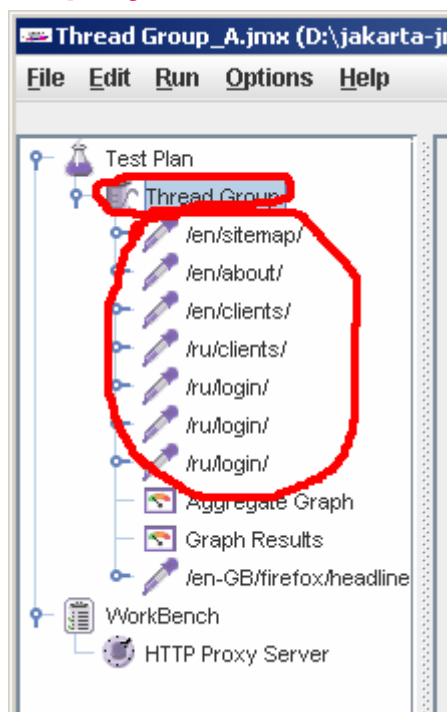
Для начала рекомендуется использовать «Aggregate Graph» и «Graph Results»



JMeter – выполнение теста

Перед тем, как запустить тест, **переместите элементы теста в дочерние элементы созданной группы пользователей.**

И тест можно запускать («Run» → «Start»). В процессе работы теста в правом верхнем углу окна JMeter **будет отображаться зелёный квадрат и количество активных виртуальных пользователей.**



JMeter – где узнать больше

Более **подробную информацию** по JMeter можно получить:

1. Из **видеоролика** «Введение в JMeter.exe»
2. Из **документации** на официальном сайте:
<http://jakarta.apache.org/jmeter/usermanual/>
3. На **практике**, к которой мы скоро приступим.



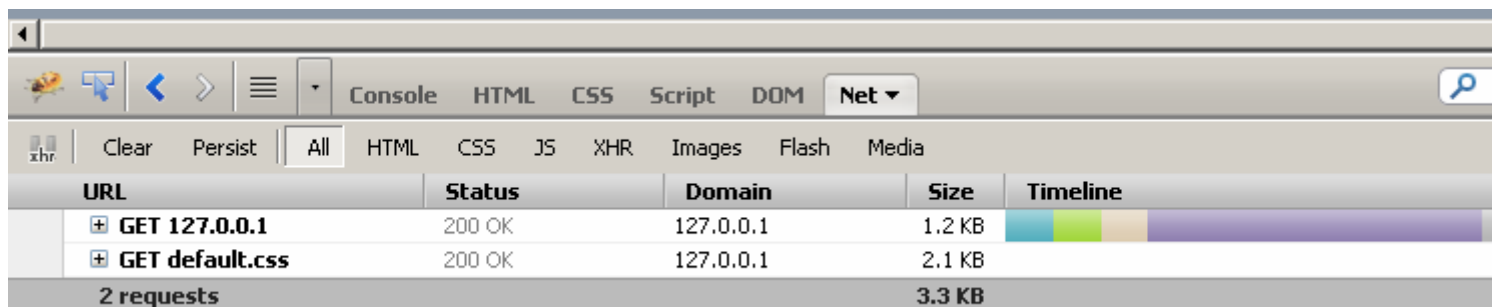
Средства сбора информации о деятельности приложения

Откуда брать информацию

Часто возникает необходимость выяснить, что в некоторый момент времени делает приложение, или что оно вообще делало при выполнении той или иной задачи.

В случае с веб-ориентированными приложениями помогают всевозможные плагины браузера, например, FireBug (мы уже рассматривали ранее).

Можно использовать и более серьёзные инструменты, такие как сетевые снифферы и т.п.



The screenshot shows the Firebug Net panel with two requests. The first request is 'GET 127.0.0.1' with a status of '200 OK', domain '127.0.0.1', and size '1.2 KB'. The second request is 'GET default.css' with a status of '200 OK', domain '127.0.0.1', and size '2.1 KB'. The total size for both requests is '3.3 KB'.

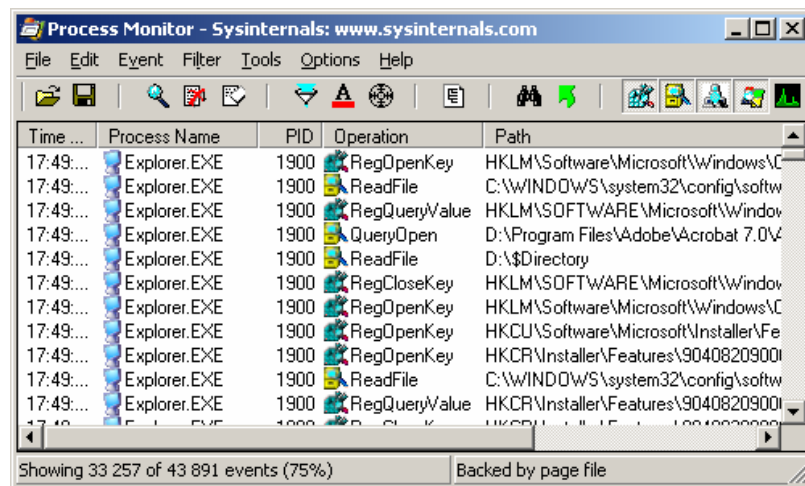
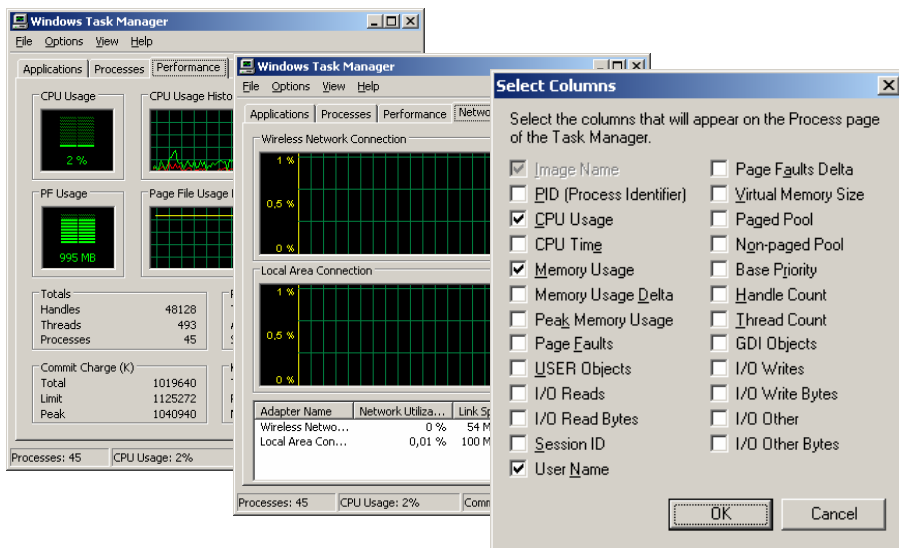
URL	Status	Domain	Size	Timeline
GET 127.0.0.1	200 OK	127.0.0.1	1.2 KB	
GET default.css	200 OK	127.0.0.1	2.1 KB	
2 requests			3.3 KB	

Откуда брать информацию

Если для сбора информации нельзя использовать средства слежения за HTTP-коммуникациями, можно использовать:

1. **Windows Task Manager** – его возможностей хватает для решения элементарных проблем.

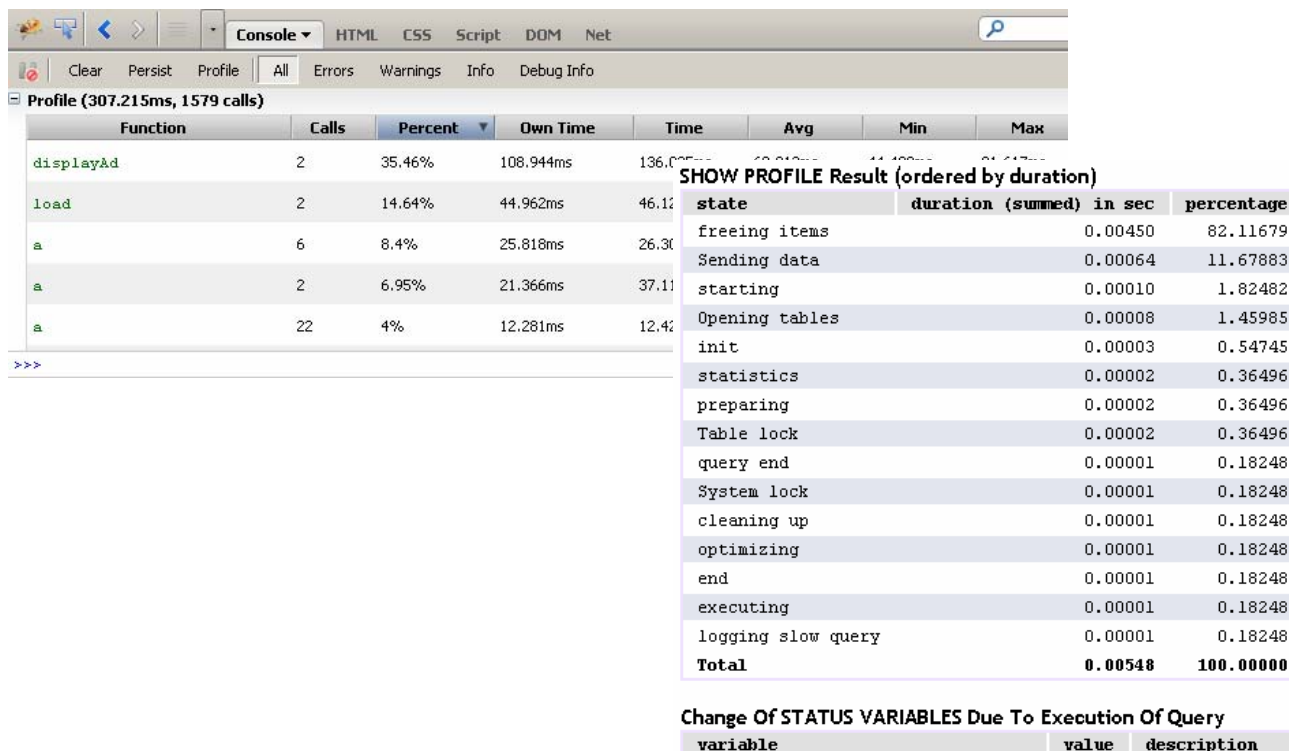
2. **Process Monitor** – бесплатную утилиту, собирающую множество информации о происходящем в системе.



Откуда брать информацию

В некоторых случаях можно воспользоваться т.н. «профилировщиками». Так, например, профилировщик работы JavaScript встроен в FireBug.

Есть профилировщики и в средствах работы с базами данных – например, в SQLyog и phpMyAdmin.



The screenshot shows the Firebug Console with a JavaScript profile and a MySQL SHOW PROFILE result.

JavaScript Profile (307.215ms, 1579 calls)

Function	Calls	Percent	Own Time	Time	Avg	Min	Max
displayAd	2	35.46%	108.944ms	136.075ms	68.037ms	44.400ms	81.617ms
load	2	14.64%	44.962ms	46.112ms	23.056ms	11.111ms	34.962ms
a	6	8.4%	25.818ms	26.300ms	4.387ms	1.111ms	13.617ms
a	2	6.95%	21.366ms	37.112ms	18.556ms	11.111ms	34.962ms
a	22	4%	12.281ms	12.400ms	0.554ms	0.111ms	1.667ms

SHOW PROFILE Result (ordered by duration)

state	duration (summed) in sec	percentage
freeing items	0.00450	82.11679
Sending data	0.00064	11.67883
starting	0.00010	1.82482
Opening tables	0.00008	1.45985
init	0.00003	0.54745
statistics	0.00002	0.36496
preparing	0.00002	0.36496
Table lock	0.00002	0.36496
query end	0.00001	0.18248
System lock	0.00001	0.18248
cleaning up	0.00001	0.18248
optimizing	0.00001	0.18248
end	0.00001	0.18248
executing	0.00001	0.18248
logging slow query	0.00001	0.18248
Total	0.00548	100.00000

Change Of STATUS VARIABLES Due To Execution Of Query

variable	value	description
status_variables	0	0

Profiling	
Status	Time
starting	0.000052
Opening tables	0.000056
System lock	0.000006
Table lock	0.000011
init	0.000024
optimizing	0.000007
statistics	0.000015
preparing	0.000013
executing	0.000005
Sending data	0.000089
end	0.000006
query end	0.000004
freeing items	0.000568
logging slow query	0.000008
cleaning up	0.000005

✓

Showing rows 0 - 3 (~4¹ total, 1 page)

Откуда брать информацию

И, наконец, очень многие программные средства способны вести **собственные протоколы** (лог-файлы).

К слову, **наше тестируемое приложение ТОЖЕ может вести такие лог-файлы**, если обосновать их необходимость перед отделом разработки...

```
127.0.0.1 -- [14/Nov/2008:19:32:22 +0200] "GET /phpmyadmin/css/phpmyadmin.css.php?&js_f
127.0.0.1 -- [14/Nov/2008:19:32:22 +0200] "GET /themes/original/img/error.ico HTTP/1.1"
127.0.0.1 -- [14/Nov/2008:19:32:36 +0200] "POST /phpmyadmin/tbl_create.php HTTP/1.1" 20
127.0.0.1 -- [14/Nov/2008:19:32:36 +0200] "GET /phpmyadmin/css/phpmyadmin.css.php?&js_f
127.0.0.1 -- [14/Nov/2008:19:32:37 +0200] "GET /phpmyadmin/js/keyhandler.js HTTP/1.1" 2
127.0.0.1 -- [14/Nov/2008:19:32:37 +0200] "GET /phpmyadmin/left.php?&server=1&db=pvsi&t
127.0.0.1 -- [14/Nov/2008:19:32:37 +0200] "GET /phpmyadmin/css/phpmyadmin.css.php?&js_f
127.0.0.1 -- [14/Nov/2008:19:33:17 +0200] "POST /phpmyadmin/tbl_create.php HTTP/1.1" 20
127.0.0.1 -- [14/Nov/2008:19:33:17 +0200] "GET /phpmyadmin/css/phpmyadmin.css.php?&js_f
127.0.0.1 -- [14/Nov/2008:19:33:18 +0200] "GET /themes/original/img/error.ico HTTP/1.1"
127.0.0.1 -- [14/Nov/2008:19:33:18 +0200] "GET /phpmyadmin/left.php?db=pvsi HTTP/1.1" 2
127.0.0.1 -- [14/Nov/2008:19:33:18 +0200] "GET /phpmyadmin/css/phpmyadmin.css.php?&js_f
127.0.0.1 -- [14/Nov/2008:19:33:29 +0200] "GET /phpmyadmin/tbl_change.php?db=pvsi&table
127.0.0.1 -- [14/Nov/2008:19:33:30 +0200] "GET /phpmyadmin/css/phpmyadmin.css.php?&js_f
127.0.0.1 -- [14/Nov/2008:19:33:31 +0200] "GET /phpmyadmin/js/tbl_change.js HTTP/1.1" 2
127.0.0.1 -- [14/Nov/2008:19:33:31 +0200] "GET /themes/original/img/error.ico HTTP/1.1"
127.0.0.1 -- [14/Nov/2008:19:34:26 +0200] "POST /phpmyadmin/tbl_replace.php HTTP/1.1" 2
127.0.0.1 -- [14/Nov/2008:19:34:27 +0200] "GET /phpmyadmin/css/phpmyadmin.css.php?&js_f
127.0.0.1 -- [14/Nov/2008:19:36:58 +0200] "GET /pvsi/26.php HTTP/1.1" 200 221
127.0.0.1 -- [14/Nov/2008:19:36:58 +0200] "GET /favicon.ico HTTP/1.1" 404 295
127.0.0.1 -- [14/Nov/2008:19:37:40 +0200] "GET /phpmyadmin/db_details_structure.php?db=
127.0.0.1 -- [14/Nov/2008:19:37:40 +0200] "GET /phpmyadmin/css/phpmyadmin.css.php?&js_f
127.0.0.1 -- [14/Nov/2008:19:37:40 +0200] "GET /phpmyadmin/left.php?&server=1&db=pvsi&t
127.0.0.1 -- [14/Nov/2008:19:37:41 +0200] "GET /phpmyadmin/css/phpmyadmin.css.php?&js_f
127.0.0.1 -- [14/Nov/2008:19:37:41 +0200] "GET /phpmyadmin/tbl_properties_structure.php
127.0.0.1 -- [14/Nov/2008:19:37:41 +0200] "GET /phpmyadmin/css/phpmyadmin.css.php?&js_f
127.0.0.1 -- [14/Nov/2008:19:38:24 +0200] "GET /pvsi/26.php HTTP/1.1" 200 458
```

Time = 0.1102180480957

Тест для проверки изученного

1. Какие инструментальные средства тестирования производительности вы знаете?
2. Каковы основные возможности JMeter?
3. Как с помощью JMeter записать тест?
4. Какие средства сбора информации о деятельности приложения вы знаете?
5. Как собрать информацию о действиях приложения, не затрагивающих HTTP-протокол?
6. Что такое (в контексте тестирования производительности) «профилировщик»?
7. Каким образом можно собрать информацию о деятельности разрабатываемого нами приложения, если существующие «средства слежения» не помогают?
8. Какие тесты производительности вы бы провели для базы данных?
9. Какие тесты производительности следует провести для анализа масштабируемости системы?
10. Чем опасны с точки зрения производительности утечки памяти?