**1 Нагрузочное тестирование**

Современное программное обеспечение просто обязано бесперебойно работать под колоссальными нагрузками. Любого рода проблемы, связанные с плохой производительностью, могут стать причиной отказа клиентов от использования вашего ПО. В связи с этим, **проведение качественного нагрузочного тестирования** должно стать обязательным, для обеспечения стабильности работы ваших приложений.

Теперь давайте ответим на вопрос: "**Что же такое нагрузочное тестирование?**", и постараемся более подробно описать **процесс проведения нагрузочного тестирования**.

**Нагрузочное тестирование** (Load Testing) или **тестирование производительности** (Performance Testing) - это **автоматизированное тестирование**, имитирующее работу определенного количества бизнес пользователей на каком-либо общем (разделяемом ими) ресурсе.

Начиная работу в области нагрузочного тестирования, следует четко понимать, что это не просто запись и прогон (Record and Playback) скриптов, а более сложный процесс:

* **Во-первых**, нагрузочное тестирование - это серьезная исследовательская и аналитическая работа
* **Во-вторых** - это реальное автоматизированное тестирование, требующее серьезных навыков программирования, а также знания сетевых протоколов и различных серверов приложений и баз данных
* **В-третьих** - существуют разные [**виды нагрузочного тестирования**](http://www.protesting.ru/testing/types/loadtesttypes.html), ставящие перед собой [**разные цели**](http://www.protesting.ru/automation/load/loadtestgoals.html)

## 2 Основные виды тестирования производительности

Рассмотрим основные виды нагрузочного тестирования, также задачи стоящие перед ними.

**Тестирование производительности**(**Performance testing**)

Задачей тестирования производительности является определение масштабируемости приложения под нагрузкой, при этом происходит:

* измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций
* определение количества пользователей, одновременно работающих с приложением
* определение границ приемлемой производительности при увеличении нагрузки (при увеличении интенсивности выполнения этих операций)
* исследование производительности на высоких, предельных, стрессовых нагрузках

**Стрессовое тестирование** (**Stress Testing**)

Стрессовое тестирование позволяет проверить насколько приложение и система в целом работоспособны в условиях стресса и 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.

**Объемное тестирование** (**Volume Testing**)

Задачей объемного тестирования является получение оценки производительности при увеличении объемов данных в базе данных приложения, при этом происходит:

* измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций
* может производиться определение количества пользователей, одновременно работающих с приложением

**Тестирование стабильности или надежности** (**Stability / Reliability Testing**)

Задачей тестирования стабильности (надежности) является проверка работоспособности приложения при длительном (многочасовом) тестировании со средним уровнем нагрузки. Время выполнения операций может играть в данном виде тестирования второстепенную роль. При этом на первое место выходит отсутствие [**утечек памяти**](http://alexeybulat.blogspot.com/2008/10/memory-leak.html), перезапусков серверов под нагрузкой и другие аспекты влияющие именно на стабильность работы.

**3 Терминология в нагрузочном тестировании**

Чтобы обсуждать **подходы к нагрузочному тестированию** и проблемы решаемые с его помощью, предлагаем начать с терминологии. Понимая под различными терминами одни и те же сущности можно говорить на "одном языке".

**Нагрузочное тестирование** или **Тестирование производительности** - это **автоматизированное тестирование**, имитирующее работу определенного количества бизнес пользователей на каком-либо общем (разделяемом ими) ресурсе. В качестве примера можно привести работу сотрудников современного банка, в котором все работают с одними и теми же программными приложениями, установленными на банковских серверах. Или использование программного приложения веб магазин, в данном случае посетителями, нагружающими сервера, будут пользователи интернета.

Моделирование нагрузки происходит с помощью специальных продуктов и техник. Что же, чем и как мы собираемся моделировать? Для того чтобы разобраться в этом и нужно определиться с терминологией:

1. **Виртуальный пользователь** (**Virtual User**) - программный процесс, циклически выполняющий моделируемые операции
2. **Итерация** (**Iteration**) – это один повтор выполняемой в цикле операции
3. **Интенсивность выполнения операции** (**Operation Intensity**) - частота выполнения операции в единицу времени, в тестовом скрипте задается интервалом времени между итерациями
4. **Нагрузка** (**Loading**) - совокупное выполнение операций на общем ресурсе (тр./сек, хитов/сек)
5. **Производительность** (**Performance**) - количество выполняемых операций за период времени (N операций за M часов)
6. **Масштабируемость приложения** (**Application Scalability**) - пропорциональный рост производительности при увеличении нагрузки
7. **Профиль нагрузки** (**Performance Profile**) - это набор операций с заданными интенсивностями, полученный на основе сбора статистических данных либо определенный путем анализа требований к тестируемой системе
8. **Нагрузочной точкой** называется рассчитанное (либо заданное Заказчиком) количество виртуальных пользователей в группах, выполняющих операции с определенными интенсивностями

Теперь рассмотрим как эти сущности связаны между собой. Выразив интенсивность через интервал времени между итерациями, видим что рост интенсивности выполняемых операций это сокращение интервала времени. Рост нагрузки пропорционален росту интенсивности. Естественно также, что при увеличении интенсивности растет производительность. При этом увеличивается степень использования (загруженности) ресурсов. С какого-то момента рост производительности прекращается (а нагрузка может продолжать расти), происходит насыщение и затем деградация системы. В дополнение можно заметить что при тестировании изменение интенсивности операций может подчиняться какому либо закону (например, Пуассона) либо быть равномерным в течении всего теста.

**4 Цели нагрузочного тестирования**

Основными **целями нагрузочного тестирования** являются:

1. Оценка производительности и работоспособности приложения на этапе разработки и передачи в эксплуатацию
2. Оценка производительности и работоспособности приложения на этапе выпуска новых релизов, патч-сетов
3. Оптимизация производительности приложения, включая настройки серверов и оптимизацию кода
4. Подбор соответствующей для данного приложения аппаратной (программной платформы) и конфигурации сервера

Заметим, что в рамках одной цели могут использоваться разные [**виды тестов производительности и нагрузки**](http://www.protesting.ru/testing/types/loadtesttypes.html), например, для первой, второй и третьей цели нужно производить как тестирование производительности так и тестирование стабильности. Но при планировании нагрузочного тестирования логичнее все же отталкиваться от **технических целей** (а не коммерческих, перечисленных выше), которые достигаются в результате тестирования и классифицировать тесты по ним:

1. Если интересует **исследование производительности приложения**, а именно времена отклика для операций на разных нагрузках в довольно широких диапазонах, включая стрессовые нагрузки то это все так и **тестирование производительности** (Performance Testing)
2. Если целью является понимание насколько приложение устойчиво в режиме длительного использования (исключение утечек памяти, некорректных конфигурационных настроек и т.д.) то проводится долгий нагрузочный тест - это **тестирование стабильности** (Stability Testing). При этом анализ времен отклика может иметь место, но не быть первым приоритетом, главное чтобы система "не упала".
3. **Стресс тестирование** (Stress Testing) имеет своей целью проверить возвращается ли система после запредельной нагрузки (и как скоро) к нормальному режиму, также целями стрессового тестирования могут быть проверки поведения системы в случаях когда, один из серверов приложения в пуле перестаёт работать, аварийно изменилась аппаратная конфигурации сервера базы данных и т.д. Отметим также, что при стрессовом тестировании проверяется не производительность системы, а её способность к регенерации после сверх нагрузки

**5 Этапы проведения нагрузочного тестирования**

Рассматривая **этапы** проведения **нагрузочного тестирования**, хотелось бы отметить следующие, на наш взгляд обязательные:

1. [**Анализ требований и сбор информации о тестируемой системе**](http://www.protesting.ru/automation/practice/work_load_model.html#req0)
2. [**Конфигурация тестового стенда для нагрузочного тестирования**](http://www.protesting.ru/automation/practice/test_stand_configuration.html)
3. [**Разработка модели нагрузки**](http://www.protesting.ru/automation/load/loadmodel.html)
4. [**Выбор инструмента для нагрузочного тестирования**](http://www.protesting.ru/automation/load/loadtesttools.html)
5. [**Создание и отладка тестовых скриптов**](http://www.protesting.ru/automation/practice/load_scripts_writing.html)
6. Проведение тестирования
7. Анализ результатов
8. Подготовка, отправка и публикация отчета по проведенному нагрузочному тестированию

## 5.1 Анализ требований

При анализе требований основной упор необходимо сделать на определение основных критериев успешности проведенных тестов. Для этого Вам необходимо будет выделить следующие характеристики:

* **Время отклика** (время необходимое для открытия страницы или получения ожидаемого результата)
* **Интенсивность** (число запросов в секунду – (Qps)
* **Используемые ресурсы** (загрузка процессора, кол-во используемой памяти, дисковое и сетевой I/O)
* **Максимальное количество пользователей** (определяет число пользователей, способных работать с системой в условиях заданной конфигурации)

А также некоторые метрики связанные с работой бизнес сценариев (например, количество бизнес операций в единицу времени, время выполнения бизнес операции и т.д.)

Заданные в требованиях характеристики, будут являться **базовыми нагрузочными точками** тестируемого приложения. Все получаемые результаты будут сравниваться с ними для принятия решения о завершении тестирования либо дальнейшем профилировании производительности.

**Примечание**:   
Основной проблемой при анализе требований будет являться их отсутствие! В связи с тем, что не всегда наши бизнес аналитики или люди ответственные за написание требований по производительности реально представляют как система должна работать под нагрузкой, какие именно требования должны быть предоставлены, очень часто цифры берутся просто «с потолка», поэтому нам приходится не только выделять имеющиеся требования, но и проводить их глубокий анализ на предмет их корректности.

**Примечание**:   
В случае, когда требования заданы не полностью, базовая нагрузочная точка может быть рассчитана по имеющимся данным (см. [**расчет нагрузочных точек**](http://www.protesting.ru/automation/load/loadpoint.html)), при условии наличия необходимых характеристик.

**Примечание**:   
Характеристика “Максимальное количество пользователей” на самом деле является малоинформативной, в случае если для тестирования нам необходимо будет работать с несколькими группами пользователей. Поэтому крайне важно будет знать более менее точное количество пользователей в каждой группе.

### Анализ требований в зависимости от типа проекта

При анализе требований необходимо учесть разрабатывается ли новое (**startup project**) или же проект направлен на профилирование нагрузки для уже находящегося в эксплуатации приложения (**profiling project**).

Рассмотрим на основании чего разрабатываются требования по производительности в зависимости от типа проекта.

Для startup проектов:

* анализ общепринятых критериев производительности
* анализ производительности конкурирующих приложений
* анализ экспертного мнения разработчиков, системных и сетевых администраторов, администраторов баз данных и инженеров по нагрузочному тестированию.
* анализ ожиданий целевых пользователей (групп пользователей)

Для profiling проектов:

* анализ общепринятых критериев производительности
* анализ производительности конкурирующих приложений
* анализ производительности эксплуатируемой версии приложения, с целью определения требующих профилирования функций, процессов, операций и т.д.
* анализ экспертного мнения разработчиков, системных администраторов и администраторов баз данных, инженеров по нагрузочному тестированию.
* анализ мнения целевых пользователей (групп пользователей)

И уже после получения всех данных из всех источников можно получить более менее точные требования по производительности для тестируемого приложения.

**Примечание**:   
В случае, если подобный анализ “имел место быть”, и в документации по проекту четко описаны все требования, то можно считать, что вам повезло. Так как в противном случае часть работы по разработке требований может лечь на Ваши плечи и будет включено во время отведенное на тестирование, которого и без того обычно не хватает.

## 

## 5.1.1 Анализ целевой аудитории

Общение непосредственно с пользователями, наблюдение за их действиями, помогает нам более точно понять как будет происходить работа с ПО, какие части приложения наиболее критичны и требуют особого внимания.

**Примечание**:   
В случае, если тестируемая система уже находится в эксплуатации, пользовательские сценарии и распределенее запросов в течении дня, можно получить проанализировав логи приложения и сервера.

В результате мы получим:

* Четко выделенные группы пользователей (например, администраторы, операторы, зарегистрированные клиенты, новые пользователи и т.д.)
* Описание сценариев тестирования, на основе реального поведения пользователей из каждой группы.
* График распределение выполнения бизнес операций в течении дня.

**Например**:   
Утром операторы проверяют журналы посещения, обрабатывают сообщения клиентов, после обеда вводят новые товары в базу, и в конце рабочего дня снова анализируют журналы и сообщения. Значит основная нагрузка на журналы посещения и базы сообщений будет в начале и в конце рабочего дня.

**5.1.2 Определение базового профиля нагрузки**

Выделив, на предыдущем шаге группы пользователей, бизнес сценарии, требования к системе можем приступить к созданию профиля нагрузки – набор операций и интенсивность их выполнения для каждой пользовательской группы.

Зная количество пользователей в каждой группе, мы должны будем распределить их для выполнения требуемых операций, сохранив необходимую интенсивность.

**Например**:

В группе “Администраторы” работает 10 админов. Они выполняют 2 основные операции - “добавить пользователя” и “забанить пользователя”.

Распределим выполнение операций следующим образом: 5 админов выполняют операцию добавления и 5 забанивают пользователей. Проведем перерасчет интенсивности выполнения операций:

В течении дня 10 администраторов добавляют 240 пользователей. Значит, интенсивность выполнения операции добавления для 1 администратора = 1 операция в час. Но в нашем случае только 5 админов будут выполнять операцию добавления, значит нам необходимо увеличить интенсивность так, чтобы количество добавленных пользователей осталось прежним. Итого, мы получим, что выполняя по 2 операции в час, 5 админов будут добавлять те же 240 пользователей. Проведя аналогичный перерасчет интенсивности для операции “забанить пользователя”, мы получим профиль нагрузки для группы Администраторы:

|  |  |  |  |
| --- | --- | --- | --- |
| **Группа** | **кол-во юзеров** | **операция** | **интенсивность опер./час** |
| Админы | 5 | добавить пользователя | 2 |
| Админы | 5 | забанить пользователя | 5 |

Проведя аналогичный перерасчет для каждой группы пользователей, мы получим небольшую сводную таблицу со списком операций и интенсивностью их выполнения:

|  |  |  |  |
| --- | --- | --- | --- |
| **группа** | **кол-во юзеров** | **операция** | **интенсивность опер./сек** |
| Админы | Z | добавить пользователя | N |
| Админы | Z | забанить пользователя | N |
| ... |  |  |  |
| покупатели | Z | вход/выход в систему | N |
| покупатели | Z | поиск товара | N |
| покупатели | Z | покупка товара | N |
| ... |  |  |  |
| Продавцы | Z | просмотр посещения | N |
| Продавцы | Z | добавление/удаление товара | N |

Именно эта таблица и будет являться полным **базовым профилем нагрузки** (**performance baseline profile**) для тестируемого приложения.

## 

## 5.1.3 Разработка моделей нагрузки

В зависимости от вида проводимого тестирования и целей преследуемых им, нам придется разрабатывать разные модели нагрузки.

Так разным может быть:

* количество пользователей и интенсивность запросов
* старт самих пользователей: последовательно по одному, ступенчато группами или же запуск всех сразу.
* длительность сценария от 10 минут до нескольких часов или даже дней.

Проще всего изменения проводить, варьируя параметрами в базовом профиле нагрузки и инструменте для тестирования производительности (например, в HP LoadRunner Controller).

### Модель тестирования производительности

Постепенное увеличение нагрузки, добавляя новых пользователей с некоторым интервалом времени, позволяет нам определить:

* измерение времени выполнения выбранных операций при определенных интенсивностях выполнения этих операций
* количество пользователей, способных одновременно работать с приложением
* границы приемлемой производительности при увеличении нагрузки
* производительность при разных нагрузках

### Модель стрессового тестирования

Увеличивая интенсивность операций выше пиковых (максимально разрешенных) значений либо увеличивая количество пользователей, до тех пор пока нагрузка не станет выше максимально допустимых значений, проверяем, что система работоспособна в условиях стресса. Далее, опустив нагрузку до средних значений, проверяем (способность системы к регенерации), что система вернулась к нормальному состоянию (основные нагрузочные характеристики не превышают базовые).

### Модель объемного тестирования

Можно использовать ту же модель что и для тестирования производительности однако целью будет проверка работы системы с прогнозом на будущий рост объема данных. Следовательно одним и самым важным предусловием теста будет увеличение объемов базы данных приложения до требуемых размеров. Таким образом мы сможем проверить и оценить производительность, прогнозируя рост системы на год, два или три вперед.

### Модель тестирования стабильности или надежности

Используя базовый нагрузочный профиль, запускаем тест длительностью от нескольких часов до нескольких дней, с целью выявления утечек памяти, перезапуска серверов и других аспектов влияющих на нагрузку.

## 5.1.4 Выводы

Четкое следование всем вышеописанным инструкциям по разработке моделей нагрузки, позволит нам:

* провести дополнительный анализ и тестирование требований по производительности
* уточнить параметры и характеристики производительности
* получить более четкого представления о работе системы в целом
* получить на выходе план предстоящих работ связанных с нагрузочным тестированием
* определить предельный объем данных системы (с сохранением приемлемой производительности)
* определить предельное количество пользователей (групп) системы (с сохранением приемлемой производительности)
* определить ресурсоёмкие операции или сценарии (для дальнейшего профилирования системы)
* отслеживать эффект от вводимых оптимизаций системы при регулярных измерениях производительности, используя разработанные и проверенные модели нагрузки,

В итоге, имея на руках готовые разработанные модели нагрузки, можно смело переходить к следующему этапу подготовки к нагрузочному тестированию - конфигурации тестового стенда. Но это уже совершенно другая статья.

## 5.2 Конфигурация тестового стенда

Отметим те части конфигурации, которые требуют особого внимания:

* **Hardware**
  + процессор (тип, частота, количество ядер и т.д)
  + оперативная память (тип, объем, тайминг, эффективная частота и т.д.)
  + жесткие диски (тип, скорость и т.д.)
* **Software**
  + Операционная система
  + Драйвера
* **Network**
  + топология сети
  + пропускная способность
  + протокол передачи данных
* **Application**
  + Архитектура
  + База данных (структура + данные)
  + программное обеспечение, необходимое для работы приложения (например, для Java приложений - JVM)

В самом идеальном случае **тестовый стенд один к одному дублирует конфигурацию реального сервера**, на котором работает или же будет работать приложение. Однако, как мы с вами знаем, идеальных случаев практически не бывает (то памяти мало, то процессора такой частоты нет в наличии, то операционная система не той версии, то стоимость некоторого серверного ПО не укладывается в бюджете). Перечислим основные причины, по которым не всегда получается продублировать конфигурацию системы на тестовом стенде:

1. Сложность дублирования дорогого серверного железа для тестовых нужд
2. Ограничения на использование лицензий требуемого программного обеспечения
3. Закрытость архитектуры приложения со стороны заказчика по соображениям безопасности
4. Трудность воссоздания или транспортировки базы данных приложения
5. Сложность воссоздания требуемой архитектуры сети
6. и многое другое (всё перечислить крайне сложно из-за большого количества нюансов, влияющих на конфигурацию системы)

Целесообразность же воссоздания инфраструктуры необходимо оценить с учетом выделенных ресурсов, времени и усилий, так как не всегда результат будет оправдывать средства.

**Приведем пример** (придуманный из головы и вполне возможно нереальный) для того, чтобы показать сложность сравнения результатов на двух разных системах:

Требуется конфигурация: Proc Intel Clarkdale Core I5, 16 Gb памяти DDR3-800, OS SLES11

Конфигурация тестового стенда: Proc AMD Deneb Phenom II X4, 16 Gb DDR2-800, OS SLES10

Казалось бы отличия незначительны, однако есть нюансы:

1. Разные типы процессоров
2. Разный тип памяти
3. Разные версии операционной системы

Итак:

**Cравним процессоры** [**[3]**](http://www.protesting.ru/automation/practice/test_stand_configuration.html#proc):

|  |  |  |
| --- | --- | --- |
| Название ядра | Intel Clarkdale Core I5 | AMD Deneb Phenom II X4 |
| Тактовые частоты, Мгц | 2800-3600 | 2800-3000 |
| Частоты системной шины, Мгц | 2500 | 2000-1800 |
| Пропускная способность шины процессор-чипсет | 2 ГБ/сек (1 ГБ/сек в одном направлении) | 14.4-16 ГБ/сек (7.2-8 ГБ/сек в одном направлении) |
| Размер кэша L1, Кб | 64 х2 | 128 х4 |
| Размер внутреннего кэша L2, Кб | 256 x2 | 512 х4 |
| Ширина шины L2 кеша, бит | 256 (двунаправленная) | 256 (по 128 в каждом направлении) |
| ... | ... | ... |

**Примечание**: Не правда ли, непросто будет сравнивать без помощи специалистов? Тем более вывести коэффициент на сколько какой процессор быстрее и по каким параметрам.

**Cравним память** [**[4]**](http://www.protesting.ru/automation/practice/test_stand_configuration.html#mem):

Стандартная задержка (latency) типичная для DDR2 5-5-5-15, а для DDR3 памяти 7-7-7-15. Однако даже с большей задержкой DDR3 память (в отличие от старого стандарта DDR2) обладает большей пропускной способностью (bandwidth) из-за более высокой тактовой частоты (clock speed). Что становится причиной того, что DDR3 память работает немного медленнее, чем DDR2 с такой же частотой 800MHz.

Таким образом, память на тестовом стенде будет немного быстрее чем в требуемой конфигурации.

**Cравним операционные системы**

По мнению экспертов, какой-либо существенной разницы в производительности из-за версии ОС не будет, однако небольшой прирост может быть только из-за того, что в более свежих ядрах чуть оптимальнее код.

Таким образом предположим, что SLES 11 будет незначительно быстрее SLES 10.

Удивлены таким большим количеством различий? (Причем это только для трех элементов конфигурации: процессора, памяти и операционной системы. В реальности их намного больше) Легко ли будет сделать выводы о том, как будет работать приложение на требуемой конфигурации? (Я думаю, что практически невозможно) Однако далее мы попробуем все таки найти пути для решения этой проблемы, рассмотрев некоторые практические приемы.

## 5.2.1 Анализ результатов при разных конфигурациях

Как же быть с тем фактом, что приложение будет протестировано на одной платформе, а работать ему придется на другой? Как провести анализ результатов? Как сделать окончательные выводы и спрогнозировать работу приложения под нагрузкой в будущем?

### Использование переходного коэффициента

В своей статье [**[1]**](http://www.protesting.ru/automation/practice/test_stand_configuration.html#berzin) Вячеслав Берзин, предложил при анализе результатов использовать “переходный коэффициент”, показывающий численно отличие производительности тестовой платформы от промышленной:

*"Если существует отличие в конфигурациях тестового стенда и тестируемой системы, необходима оценка переходного коэффициента для производительности тестовой и промышленной систем. На основании этого коэффициента, проводится масштабирование полученных результатов. Для получения погрешности оценки, не превышающей 5-10%, рекомендуется проведение оценки переходного коэффициента несколькими независимыми способами, например, на основании анализа независимых данных о производительности используемых аппаратных платформ."*

Использование данного подхода, значительно облегчает создание и конфигурацию тестового стенда, но прибавляет работы на этапе анализа результатов. Так что вам придется решать самим: либо пытаться один к одному (100%) скопировать промышленную систему, либо сделать её максимально приближенной и с помощью переходного коэффициента анализировать и делать прогнозы на будущий рост производительности.

Однако, не всегда возможно точно получить значение переходного коэффициента из-за нелинейных зависимостей между компонентами системы, а примерный расчет может дать слишком большую погрешность. Поэтому использование данного подхода видится реальным, когда тестовая и реальная конфигурации системы практически идентичны.

### Использование экспертной оценки

Рассмотрим мнение Андрея Широбокова [**[2]**](http://www.protesting.ru/automation/practice/test_stand_configuration.html#shirobokov) (сертифицированный специалист в области нагрузочного тестирования, имеющий более 15 лет рабочего опыта в IT):

*"Идеально ключевые моменты должны совпадать, пишу по убыванию важности:*

1. *конфигурация (кол-во и тип процессоров х память)*
2. *версия ОС и статс паки*
3. *версия и тип СУБД*
4. *версия серверов приложений и патчей*

*Естественно, что версия тестируемого приложения должна быть актуальна.*

*Если совпадает платформа (ОС и тип процессоров), то допускается "пропустить" конфигурацию, например, в случае если проведены эксперименты на конфигурациях 8х16 (8 прц х 16 Гб) и 32х64 (32 прц х 64 Гб), то можно "аппроксимировать" недостающую конфигурацию 24х48 (24 прц х 48 Гб). Обратим внимание, что памяти для такой методики должно пропорционально быть в 2 раза больше во всех случаях.*

*Если не совпадает платформа, ОС, типы процессоров, объемы памяти, то можно делать только очень общие выводы, наподобие: "В реальности будет не хуже". Но времена откликов и другие детали могут быть малозначимы."*

Как видим, как при использовании переходного коэффициента, так и при использовании экспертной оценки, анализ результатов и прогнозирование роста производительности при тестировании на разных конфигурациях стендов (тестового и реального) становится затруднительным, и либо требует дополнительных перерасчетов по сложным нелинейным формулам с большим количеством параметров, либо не обеспечивает должной точности результатов.

## 5.2.2 Выводы

В данной статье мы попытались описать основные моменты связанные с конфигурацией тестового стенда для нагрузочного тестирования. В результате мы имеем следующее:

* в идеальном случае тестовый стенд должен один к одному дублирует конфигурацию реального промышленного сервера;
* проводя тестирование на тестовом стенде, имеющем отличную от реальной инфраструктуру, мы не всегда можем дать точную оценку производительности реальной системы;
* для анализа результатов при небольших отклонениях в конфигурациях возможно использовать переходных коэффициент.

**5.3 Разработка модели нагрузки**

Определившись с [**видами нагрузочного тестирования**](http://www.protesting.ru/testing/types/loadtesttypes.html), [**целями**](http://www.protesting.ru/automation/load/loadtestgoals.html) и [**терминологией**](http://www.protesting.ru/automation/load/terminology.html), перейдем к основной задаче нагрузочного тестировании - **разработке модели нагрузки**.

Для этого необходимо определить следующее:

* список тестируемых операций
* интенсивность выполнения операций
* зависимость изменения интенсивности выполнения операций от времени

В список тестируемых задач должны войти операции, критичные с точки зрения бизнеса, а также с технической точки зрения:

* Критичными с точки зрения бизнеса являются операции, скорость выполнения которых, реально влияет на производительность бизнес процесса. *Например, увеличение длительности обслуживания клиентов в банке, невозможность выполнения необходимого количества операций в течение дня и так далее*.
* Критичными с технической точки зрения являются ресурсоемкие операции, требующие большое количество памятьи, серьезно задействующие процессор, создающие значительный сетевой трафик. *Как правило, это операции выполняемые одновременно большим количеством бизнес пользователей или создание сложных отчетов, в которые входят так называемые "тяжелые" запросы к базе данных*.

Хотим еще раз подчеркнуть, что *под степенью критичности операции мы подразумеваем её влияние на бизнес процесс и работоспособность системы*. Например, создание какого-нибудь отчета, полностью загружающего сервер базы данных в ночное время, не будет носить высокий приоритет для оптимизации, а в рабочие часы будет иметь максимальный приоритет.

**5.3.1 Изучение Приложения**

Будем называть тестируемое прикладное программное обеспечение "приложением". Чтобы выделить части приложения, а именно операции, которые будут тестироваться, необходимо провести работу связанную с изучением приложения. Очень большую пользу при этом должны оказать разработчики приложения, если речь идет о тестировании в процессе разработки, либо бизнес пользователи и системные администраторы, если приложение находится в процессе эксплуатации. В ходе этой работы разумно сделать такие шаги:

* Описать компоненты приложения и составить схемы взаимодействия между ними
* Выделить критические с точки зрения предполагаемого тестирования операции. В качестве таковых могут быть выбраны:
  1. Операции с "тяжелыми" запросами к базе данных, процессы генерации отчетов
  2. Операции, выполняемые большим количеством пользователей или с высокой интенсивностью
  3. Операции критичные с точки зрения бизнеса, и к тому же удовлетворяющие условиям двух верхних пунктов

Еще раз хочется заметить, что опрос бизнес пользователей или совместное исследование с разработчиками и администраторами системы может значительно облегчить задачу. Если приложение находится в эксплуатации, то можно провести мониторинг загрузки компонентов аппаратных серверов (процессора, память, диски) и проанализировать системные журналы веб серверов (снять stats pack, если в качестве сервера базы данных, например, используется Oracle). Системные журналы могут показать пики высокой активности пользователей в течение дня и дать количественное оценки того сколько транзакций (хитов) выполняется в единицу времени. Согласно [**закону Паретто или принципу 20/80**](http://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BA%D0%BE%D0%BD_%D0%9F%D0%B0%D1%80%D0%B5%D1%82%D0%BE), 20% операций приложения генерируют 80% нагрузки в системе, поэтому нужно стараться выбрать для моделирования именно эти 20% операций.

**5.3.2 Определение профиля нагрузки**

Ключевым моментом в модели нагрузки являются выбранные для тестирования операции или [**профиль нагрузки**](http://www.protesting.ru/automation/load/terminology.html). Естественно выполняться эти операции в тесте должны одновременно. **Профилей нагрузки для приложения может быть несколько** и это оправдано. Ведь бизнес пользователи могут выполнять разные наборы операций в разное время. Например, начало операционного дня и конец дня, начало месяца (квартала) и соответственно завершение могут отличаться. Таким образом получаем различные наборы операций приложения, выполняющиеся одновременно и соответственно создающие различную нагрузку. Кстати, **меняться могут не только сами операции но и их интенсивности**. В первом приближении моделью нагрузки является набор профилей нагрузки, где каждый профиль отличается от другого или набором операций или интенсивностями выполнения этих операций.

**Пример профиля нагрузки**, в который входит 5 операций, значение n может быть различным для каждой операции:

<Профиль нагрузки>

1. Операция\_1 - интенсивность выполнения n раз / ед. времени
2. Операция\_2 - интенсивность выполнения n раз / ед. времени
3. Операция\_3 - интенсивность выполнения n раз / ед. времени
4. Операция\_4 - интенсивность выполнения n раз / ед. времени
5. Операция\_5 - интенсивность выполнения n раз / ед. времени

# 5.3.3 Расчет нагрузочных точек

Поскольку **в профиле нагрузки как правило присутствует несколько операций** - это означает, что у нас будет несколько групп пользователей. Желательно моделировать каждую операцию отдельной группой виртуальных пользователей (хотя в жизни часто бывает наоборот, один бизнес пользователь может отвечать за выполнение нескольких операций). Тем не менее, если назначить одному виртуальному пользователю выполнение одной операции, то так легче выдержать определенную интенсивность (и соответственно производительность) для этой операции в тесте, чем в случае, когда виртуальному пользователю назначается последовательная цепочка операций. Зная интенсивность выполнения операции нужно определить количество виртуальных пользователей в группе, выполняющих эту операцию. **Идеальный случай** когда работа с приложением аналогична работе заводского конвейера и **есть точные оценки сколько операций в день делает один пользователь**. Чаще всего бывает не так и известно только общее количество операций выполняемое в течение дня. Так же может оказаться, что интенсивность выполнения операции каждым пользователем очень низкая, например, один пользователь выполняет операцию раз в день или раз в два дня.

Моделирование работы с пересчетом интенсивностей в этом случае можно проиллюстрировать следующим примером

**ПРИМЕР**: (для одной операции)

Количество операций = 200 за 4 часа  
Количество бизнес пользователей = 20

1. Определяем количество операций для каждого пользователя  
200/20 = 10

2. В час каждый пользователь выполняет 2.5 операции  
10/4 = 2.5

3. Определяем период повторения операции (интенсивность) для каждого пользователя  
60/2.5 = 24 минуты

4. Чтобы уменьшить время теста до часа, и при этом получить хоть какую то статистику по выполнению операций, а так же улучшить "перемешивание" операций во время теста, можно увеличить интенсивность в 4 раза, при этом уменьшив количество пользователей так же в 4 раза.  
24/4 = 6 мин. = 360 секунд

Таким образом, 5 виртуальных пользователей, выполняя операцию с периодом 6 минут, обеспечат за 4 часа заданную производительность равную 200 операциям.

**Что дает такой пересчет**:

**Во-первых**, есть возможность снизить время теста, не теряя статистики выполнения операций, а следовательно и достоверности результатов теста.   
**Во-вторых**, можно снизить количество моделируемых пользователей там где их число уходит за несколько тысяч и таким образом понизить требования к количеству ресурсов нагружающих компьютеров (1VU может требовать до 10Мб оперативной памятинагружающего компьютера ), а в некоторых случаях и выполнить условия лицензионного соглашения (стоимость лицензии для тестового инструментария зависит от количества виртуальных пользователей)

**Какие есть ограничения**:

Для некоторых приложений, может быть критично количество одновременно открываемых соединений между клиентом и сервером. Встречались ситуации когда запросы к базе данных начинали работать значительно медленнее если при работе с приложением увеличивается количество именно разных пользователей (разные логины) а не интенсивность запросов. И наконец, увеличение интенсивности выполнения операций, не должно приводить к ситуации когда период повторения становится меньше чем время выполнения самой операции.

В любом случае, несмотря на некоторые ограничения, подобный расчет может допускать варьирование количеством виртуальных пользователей и интенсивностью выполнения ими операций. При этом производительность и соответственно нагрузка не должна изменяться и отличаться от заданной. Итак, нагрузочная точка включает в себя расчеты виртуальных пользователей и интенсивностей по всем операциям профиля нагрузки.

# 5.3.4 Baseline нагрузочная точка

Хочется заметить что расчет нагрузочной точки, описанный в разделе [**Расчет точек нагрузки**](http://www.protesting.ru/automation/load/loadpoint.html), основанный на собранной для работающего приложения статистике (или на ожидаемом объеме работы для вновь разработанного приложения), является исходным для дальнейшего роста нагрузки, а сама **рассчитанная нагрузочная точка может считаться базовой или baseline точкой**. Теперь можно увеличивать нагрузку, двигаясь с некоторым шагом, увеличивая при этом только количество виртуальных пользователей в группах, не изменяя интенсивности выполнения операций для одного виртуального пользователя.

**Полная модель нагрузки - это набор профилей нагрузки со всеми нагрузочными точками для каждого профиля**. При разработке тестовых сценариев должны быть корректно реализованы все нагрузочные точки. Еще хотелось бы добавить, что нагрузочных точек для каждого профиля должно быть не меньше трех, чтобы можно было оценить зависимость времен отклика выполняемых операций от роста нагрузки. Очевидно, что чем линейнее такая зависимость тем лучше масштабируемость приложения и выше предсказуемость его поведения под нагрузкой.

**6 Инструменты для нагрузочного тестирования**

**Коммерческие инструменты** для автоматизированного нагрузочного тестирования:

|  |  |
| --- | --- |
| **Hewlett-Packard (Mercury Interactive)** | [**HP Performance Center**](https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-126-17_4000_100__) (включает [**HP LoadRunner**](http://www8.hp.com/us/en/software-solutions/software.html?compURI=1175451)) |
| **IBM Rational** | [**Rational Performance Tester**](http://www-01.ibm.com/software/awdtools/tester/performance/) |
| **Borland (Segue)** | [**Silk Performer**](http://www.borland.com/us/products/silk/silkperformer/) |
| **SmartBear** | [**LoadComplete Web Load Testing**](http://smartbear.com/products/qa-tools/load-testing-tool) |
| **Neotys** | [**NeoLoad**](http://www.neotys.com/product/overview-neoload.html) |

Отдельным пунктом выделим **бесплатные инструменты** для автоматизированного нагрузочного тестирования:

* [**Jmeter**](http://jakarta.apache.org/jmeter/) - an Apache Jakarta project that can be used as a load testing tool for analyzing and measuring the performance of a variety of services
* [**Grinder**](http://grinder.sourceforge.net/) - a load testing framework that makes it easy to run a distributed test using many load injector machines. Test scripts are written in Jython, and HTTP scripts can be recorded easily from a browser session.