**Міністерство освіти і науки України**

**Харківський національний університет радіоелектроніки**

Факультет комп'ютерних наук

Кафедра Програмної інженерії

Контрольна робота

з дисципліни «Якість програмного забезпечення та тестування»

Студент Нікулін Антон

гр. ПІз-11-1 Володимирович

Викладач Голян Наталія

Вікторівна

Харків, 2015 р.

**Тестирование производительности**

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

**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.

В общем случае методология стресс-тестирования основана на снятии и анализе показателей производительности приложения при нагрузках, значительно превышающих ожидаемые на стадии сопровождения и несёт в себе цель определить выносливость или устойчивость приложения на случай всплеска активности по его использованию.

Необходимость стресс-тестирования диктуется следующими факторами:

* Большая часть всех систем разрабатываются с допущением о функционировании в нормальном режиме и даже в случае, когда допускается возможность увеличения нагрузки, реальные объёмы её увеличения не принимаются во внимание.
* В случае SLA-контракта ([соглашения об уровне услуг](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%B3%D0%BB%D0%B0%D1%88%D0%B5%D0%BD%D0%B8%D0%B5_%D0%BE%D0%B1_%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5_%D1%83%D1%81%D0%BB%D1%83%D0%B3)) стоимость отказа системы в экстремальных условиях может быть очень велика.
* Обнаружение некоторых ошибок или дефектов в функционировании системы не всегда возможно с использованием других типов тестирования.
* Тестирования, проведенного разработчиком, может быть недостаточно для эмуляции условий при которых происходит отказ системы.
* Предпочтительнее быть готовым к обработке экстремальных условий системы, чем ожидать её отказа.

Основные направления применения стресс-тестирования:

1. Общее исследование поведения системы при пиковых нагрузках.
2. Исследование обработки ошибок и исключительных ситуаций системой при пиковых нагрузках.
3. Исследование узких мест системы или отдельных компонент при диспропорциональных нагрузках.
4. Тестирование ёмкости системы.

Стресс-тестирование, как и [нагрузочное тестирование](https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%B3%D1%80%D1%83%D0%B7%D0%BE%D1%87%D0%BD%D0%BE%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) также может быть использовано для регулярной оценки изменений производительности с целью получения для дальнейшего анализа динамики изменения поведения системы за длительный период.

**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**)

**Тестирование стабильности или надежности** (Stability / Reliability Testing) — один из видов автоматизированного [тестирования ПО](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D1%8F), целью которого является проверка работоспособности приложения при длительном тестировании с ожидаемым уровнем нагрузки.

Перед тем как подвергать ПО экстремальным нагрузкам стоит провести проверку стабильности в предполагаемых условиях работы, то есть погрузить продукт в полную рабочую атмосферу. При тестировании, длительность его проведения не имеет первостепенного значения, основная задача - наблюдая за потреблением ресурсов, выявить утечки памяти и проследить чтобы скорость обработки данных и/или время отклика приложения в начале теста и с течением времени не уменьшалась. В противном случае вероятны сбои в работе продукта и перезагрузки системы.

Часто в "домашних" условиях тестирование стабильности совмещают со [стресс-тестированием](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B5%D1%81%D1%81-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), то есть проверяют не только стабильность, но и способность приложения переносить жесткие условия и сильные нагрузки длительное время.

### Нагрузочное тестирование

**Нагрузочное тестирование** ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *load testing*) — подвид тестирования производительности, сбор показателей и определение производительности и времени отклика программно-технической системы или устройства в ответ на внешний запрос с целью установления соответствия требованиям, предъявляемым к данной системе (устройству).

Для исследования времени отклика системы на высоких или пиковых нагрузках производится [стресс-тестирование](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%B5%D1%81%D1%81-%D1%82%D0%B5%D1%81%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), при котором создаваемая на систему нагрузка превышает нормальные сценарии её использования. Не существует чёткой границы между нагрузочным и стресс-тестированием, однако эти понятия не стоит смешивать, так как эти виды тестирования отвечают на разные бизнес-вопросы и используют различную методологию.

Термин *нагрузочное тестирование* может быть использован в различных значениях в профессиональной среде тестирования ПО. В общем случае он означает практику моделирования ожидаемого использования приложения с помощью эмуляции работы нескольких пользователей одновременно. Таким образом, подобное тестирование больше всего подходит для многопользовательских систем, чаще — использующих клиент-серверную архитектуру (например, веб-серверов). Однако и другие типы систем ПО 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.

Основная цель нагрузочного тестирования заключается в том, чтобы, создав определённую ожидаемую в системе нагрузку (например, посредством виртуальных пользователей) и, обычно, использовав идентичное программное и аппаратное обеспечение, наблюдать за показателями производительности системы.

В идеальном случае в качестве критериев успешности нагрузочного тестирования выступают требования к производительности системы, которые формулируются и документируются на стадии разработки функциональных требований к системе до начала программирования основных архитектурных решений. Однако часто бывает так, что такие требования не были четко сформулированы или не были сформулированы вовсе. В этом случае первое нагрузочное тестирование будет являться *пробным* ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *exploratory load testing*) и основываться на разумных предположениях об ожидаемой нагрузке и потреблении аппаратной части ресурсов.

Одним из оптимальных подходов в использовании нагрузочного тестирования для измерений производительности системы является тестирование на стадии ранней разработки. Нагрузочное тестирование на первых стадиях готовности архитектурного решения с целью определить его состоятельность называется 'proof-of-concept' тестированием.

## Определение целей тестирования производительности

В общих случаях **тестирование производительности** может служить разным целям.

* С целью демонстрации того, что система удовлетворяет критериям производительности.
* С целью определения производительность какой из двух или нескольких систем лучше.
* С целью определения, какой элемент нагрузки или часть системы приводит к снижению производительности.

Многие тесты на производительность делаются без попытки осмыслить их реальные цели. Перед началом тестирования всегда должен быть задан бизнес-вопрос: «Какую цель мы преследуем, тестируя производительность?». Ответы на этот вопрос являются частью [технико-экономического обоснования](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D1%85%D0%BD%D0%B8%D0%BA%D0%BE-%D1%8D%D0%BA%D0%BE%D0%BD%D0%BE%D0%BC%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%BE%D0%B5_%D0%BE%D0%B1%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) (или [business case](https://en.wikipedia.org/wiki/en:business_case)) тестирования. Цели могут различаться в зависимости от технологий, используемых приложением, или его назначения, однако, они всегда включают что-то из нижеследующего:

### Параллелизм / Пропускная способность

Если конечными пользователями приложения считаются пользователи, выполняющие логин в систему в любой форме, то в этом случае крайне желательно достижение параллелизма. По определению это максимальное число параллельных работающих пользователей приложения, поддержка которого ожидается от приложения в любой момент времени. Модель поведения пользователя может значительно влиять на способность приложения к параллельной обработке запросов, особенно если он включает в себя периодически вход и выход из системы.

Если концепция приложения не заключается в работе с конкретными конечными пользователями, то преследуемая цель для производительности будет основана на максимальной пропускной способности или числе транзакций в единицу времени. Хорошим примером в данном случае будет являться просмотр веб-страниц

### Время ответа сервера

Эта концепция строится вокруг времени ответа одного узла приложения на запрос, посланный другим. Простым примером является HTTP 'GET' запрос из браузера рабочей станции на веб-сервер. Практически все приложения, разработанные для нагрузочного тестирования работают именно по этой схеме измерений. Иногда целесообразно ставить задачи по достижению производительности времени ответа сервера среди всех узлов приложения.

### Время отображения

Время отображения — одно из самых сложных для приложения для нагрузочного тестирования понятий, так как в общем случае они не используют концепцию работы с тем, что происходит на отдельных узлах системы, ограничиваясь только распознаванием периода времени в течение которого нет сетевой активности. Для того, чтобы замерить время отображения, в общем случае требуется включать функциональные тестовые сценарии в тесты производительности, но большинство приложений для тестирования производительности не включают в себя такую возможность.

### Требования к производительности

Очень важно детализировать требования к производительности и документировать их в каком-либо плане тестирования производительности. В идеальном случае это делается на стадии разработки требований при разработке системы, до проработки деталей её дизайна.

Однако **тестирование производительности** часто не проводится согласно спецификации, так как нет зафиксированного понимания о максимальном времени ответа для заданного числа пользователей. Тестирование производительности часто используется как часть процесса профайлинга производительности. Его идея заключается в том, чтобы найти «слабое звено» — такую часть системы, соптимизировав время реакции которой, можно улучшить общую производительность системы. Определение конкретной части системы, стоящей на этом критическом пути, иногда очень непростая задача, поэтому некоторые приложения для тестирования включают в себя (или могут быть добавлены с помощью add-on’ов) инструменты, запущенные на сервере (агенты) и наблюдающие за временем выполнения транзакций, временем доступа к базе данных, высокой загрузки сети и другими показателями серверной части системы которые могут быть проанализированы вместе с остальной статистикой по производительности.

**Тестирование производительности** может проводиться с использованием глобальной сети и даже в географически удаленных местах, если учитывать тот факт, что скорость работы сети Интернет зависит от местоположения. Оно также может проводиться и локально, но в этом случае необходимо настроить сетевые маршрутизаторы таким образом, чтобы появилась задержка, присутствующая во всех публичных сетях. Нагрузка, прилагаемая к системе, должна совпадать с реальным положением дел. Так например, если 50 % пользователей системы для доступа к системе используют сетевой канал шириной 56К, а другая половина использует оптический канал, то компьютеры, создающие тестовую нагрузку на систему должны использовать те же соединения (идеальный вариант) или эмулировать задержки вышеуказанных сетевых соединений, следуя заданным профайлам пользователей.

### Типичные вопросы тестирования производительности

Требования к производительности должны адресовать следующие, как минимум, вопросы:

* Что охватывается тестом производительности? Какие подсистемы, компоненты, интерфейсы и т. д. должны быть протестированы?
* Если в тест включаются пользовательские интерфейсы, то сколько одновременно работающих в системе пользователей ожидается для каждого интерфейса (необходимо определить пиковые и нормальные значения)
* Как выглядит аппаратная составляющая тестируемой системы? (Необходимо описать все сервера и сетевое оборудование)
* Каков сценарий использования каждого компонента системы? (например, 20 % запросов составляет вход в систему, 40 % — поиск, 30 % — выбор элемента, 10 % — выход из системы)
* Каков сценарий использования системы? [в одном тесте на производительность могут быть задействованы разные сценарии использования каждого компонента]
* Каковы требования ко времени выполнения серии операций серверной части приложения?

## Конфигурация vs Производительность

На **результаты нагрузочного тестирования** могут влиять разные факторы, такие как **конфигурация тестового стенда**, загруженность сети, заполненность базы данных и многие другие. Причем влияние их на производительность приложения может быть значительным и иметь нелинейную зависимость, поэтому выразить её формулой будет практически невозможно. Следовательно, чем меньше будут разниться параметры тестовой и реальной инфраструктуры, тем меньше будет погрешность в полученных результатах.

**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** 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 процессор**:

|  |  |  |
| --- | --- | --- |
| 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) |

**Cравним память**:

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.

## Анализ результатов при разных конфигурациях

### Использование переходного коэффициента

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.