САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-Механический факультет

Кафедра Системного Программирования

Организация фреймворка автоматического выполнения тестовых скриптов

Курсовая работа студента 344 группы
Твёрдого Евгения Андреевича

Заведующий кафедрой:

д.ф. – м.н., профессор Терехов А. Н.

Научный руководитель:

Васильев Игорь Борисович

Санкт-Петербург

2015 г.

Введение

В разработке программного обеспечения немалую роль играет тестирование. Есть несколько подходов к нему:

- Ручное
- Автоматическое

Ручное тестирование удовлетворяет потребности до некоторой стадии развития продукта. Весь функционал может быть протестирован самим разработчиком или несколькими тестировщиками. Также ручное тестирование имеет смысл при наличии графического интерфейса, так как оно учитывает человеческий фактор и позволяет получить оценку удобства использования интерфейса.

Со временем архитектура программного продукта усложняется, и проверить максимальное количество различных сценариев становится все труднее. К тому же, существует проблема регрессионного тестирования: при модификации программного кода возникает необходимость перетестировать протестированный ранее код.

Если в разработке используется дорогостоящее оборудование, то его немного и возникают трудности в организации одновременного использования такого оборудования разработчиками и тестировщиками. Ночью оборудование простаивает. При автоматическом тестировании можно задействовать эти ресурсы, так как не требуется присутствие человека во время проведения тестов.

При ручном тестировании может возникнуть такая ситуация, что не удается воспроизвести ошибку по вине человеческого фактора. Плюс автоматического тестирования в том, что от запуска к запуску последовательность шагов четко сохраняется. Это повышает шанс перевоспроизвести ошибку с целью её исправления.

В данной работе будут разработаны механизмы автоматического запуска тестовых скриптов на основе конфигурационной информации об оборудовании (в виде plaintext файла) и параметризации (в виде XML файла) тестовых скриптов, которые являются основой тестовой среды для тестирования систем хранения данных (СХД) высокой производительности. В данный момент СХД поддерживает

одноконтроллерный (задействован один узел) и двухконтроллерный режимы работы системы (Рис. 1), при котором оба узла активны, работают одновременно и имеют доступ к единому набору дисков. В нашем случае под узлами понимаются аппаратно-независимые компоненты системы хранения данных, имеющие собственные процессоры, кэш-память, материнскую плату и могут быть объединены в кластер.

Каждый из узлов соединен с инициатором по высокоскоростному каналу, между собой узлы также соединены по высокоскоростному каналу для увеличения скорости и отказоустойчивости. Система может работать в двух режимах Active-Active - инициатор распределяет нагрузку между двумя узлами и Active-Passive - инициатор обращается к основному (Active) узлу, пассивный (Passive) узел синхронизируется с активным и хранить точную копию Кэша. В случае выхода из строя активного узла вся нагрузка переходит на пассивный.

Современные системы хранения могут масштабироваться до очень большого количества узлов и с увеличением количества узлов ручное тестирование становится неэффективным из-за того что нужно делать очень много действий на каждом узле, возникает много ошибок при конфигурировании, которые в последствии могут приводить к иным результатам тестирования.

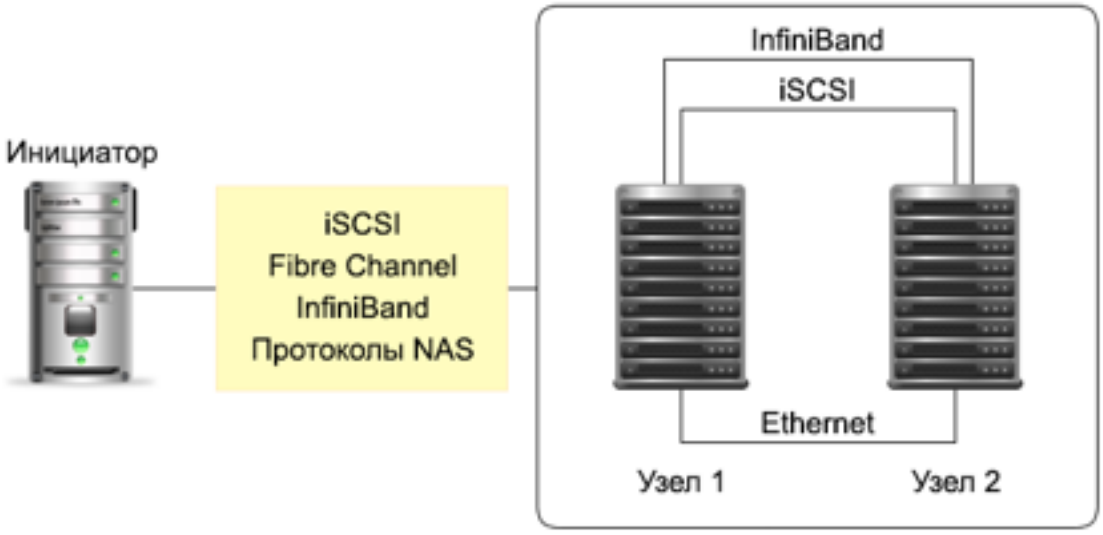

Рисунок 1

Тестовая среда подразумевает использование фреймворка низкого уровня, осуществляющего взаимодействие между сущностями, вовлеченными в тестирование, и фреймфорка высокого уровня, который обрабатывает конфигурационную информацию и осуществляет автоматический запуск. В процессе исследований был выбран фреймворк STAF т.к. он мощный, универсальный и до сих пор развивается.

Конечную автоматизацию тестирования выполняет исполнительная машина, которая на основании конфигурационной информации выполняет параметризацию и запуск скриптов. Поскольку было принято решение использовать фреймворк нижнего уровня STAF, было также решено исследовать возможность переиспользования его расширения executive engine STAX.

1. Постановка задачи

Цель данной курсовой работы - исследовать возможности переиспользования XML based executive engine STAX таким образом, чтобы в XML файле находилось описание, параметризация и ожидаемый результат выполнения тестового скрипта, а тестовый скрипт (реализующий тестовый сценарий) был написан на Python и исполнялся внешним интерпретатором. Рассмотреть возможность расширения XML файл новыми тегами для пометки тестовых скриптов и тестовых наборов таким образом, чтобы была возможность манипулировать включением или исключением их из запуска по определенным признакам

2. Общие сведения о STAF

2.1 Общая архитектура

STAF - агент, который обеспечивает связь с другими агентами установленными на других сущностях. Является отдельно запускаемым процессом, у которого есть внешние и внутренние сервисы.

Каждый STAFProc имеет уникальную идентификацию и позволяет отношения многие ко многим, таким образом позволяя очень гибко организовывать процесс тестирования.

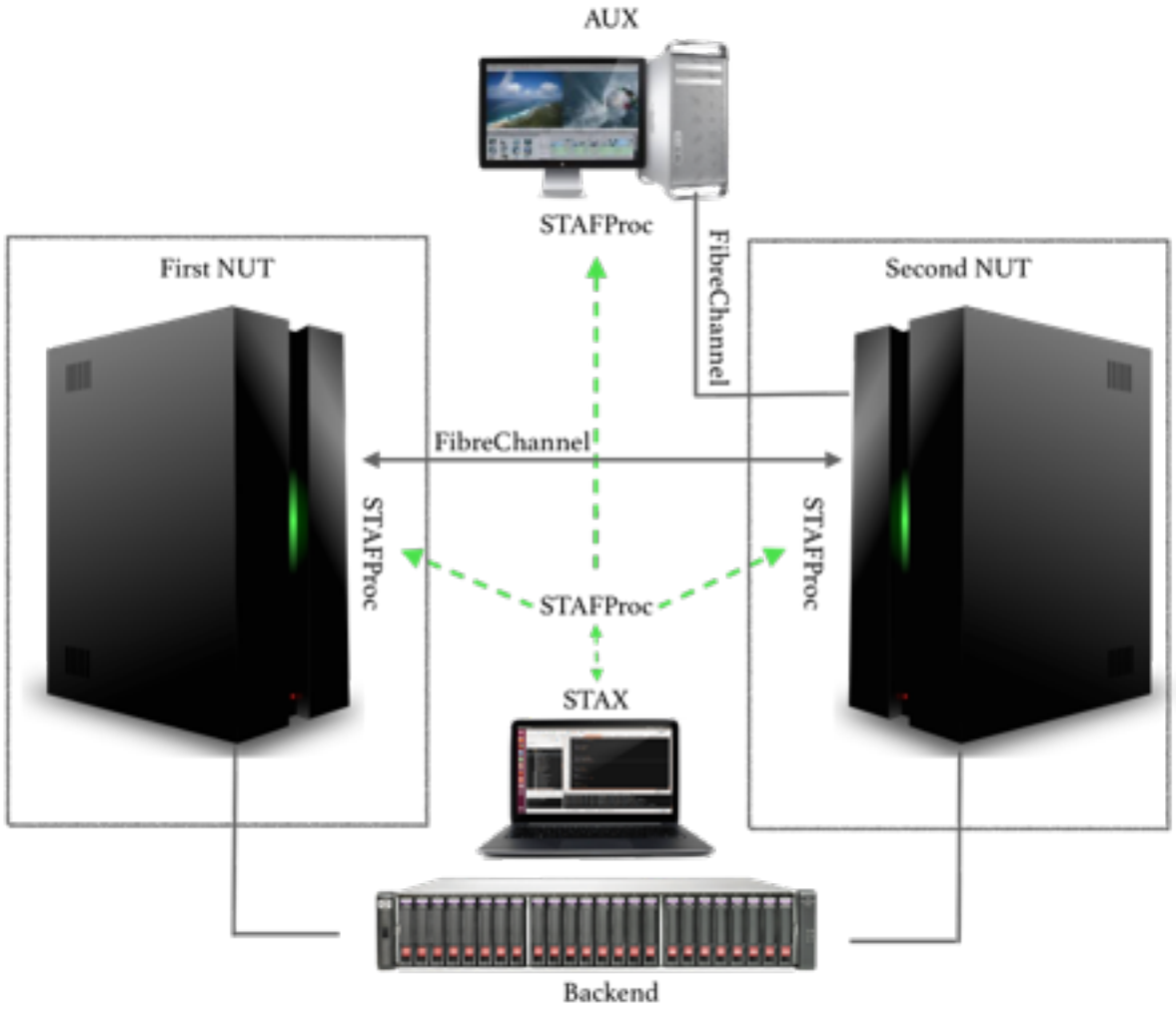

Рисунок 2. Пример архитектуры тестируемой системы.

Объект тестирования - система хранения данных, состоящая из одного или нескольких «Узлов», каждый из которых соединен с общим массивом дисков (Backend'ом). К «узлам» по высокоскоростному каналу данных подключается клиентская машина.

Фреймворк STAF имеет внешние и внутренние сервисы. STAX является внешним сервисом. Тестирование проводится с отдельной машины (на рисунке обозначена надписью STAX), на которой запускается STAF у которого зарегистрирован STAX как внешний сервис. Для того чтобы тестовая машина могла взаимодействовать с объектами тестирования, на каждом из них должен быть установлен STAF и запущен STAFProc. На рисунке выше приведена общая архитектура системы. NUT (Node Under Test) - узлы системы, AUX (Auxiliary) - клиент.

2.2 Тестовый движок STAX

STAX (STAF eXecution engine) - Тестовый движок на основе XML, выполненный в виде внешнего сервиса для фреймворка STAF (Software Testing Automation Framework). Движок разработан для упрощения подготовки тестового сценария. Написание тестового плана не требует навыков программирования.

+не нужен программист

- не очень гибкий подход

Передача управляющей информации осуществляется через представление в XML файле. STAX Написан на Java. У STAX есть собственная реализация питона на Java. Это позволяет воспользоваться мощными и простыми в использовании особенностями Python.

Также STAX предоставляет мощное GUI приложение для мониторинга выполнения и управления тестовыми заданиями.

Некоторые из основных особенностей STAX:

- Поддержка параллельного выполнения
- Определение детализации контроля исполнения
- Поддержка вложенных тестов
- Возможность контролировать длительности исполнения
- Возможность импортировать модули во время исполнения
- Поддержка существующих Python и Java модулей и пакетов

Используя эти возможности можно создавать сложные скрипты для автоматизации тестовой среды, обеспечивая максимальную эффективность и контроль.

2.3 Архитектура XML файла для STAX

STAX Elements

Основополагающий элемент в XML файле для STAX.

Может содержать:

- Данные, которые будут использованы во время выполнения задания
- Команды или процессы, которые должны быть исполнены
- Определение логики обработки функций исключений и оберток других описаны ниже элементов STAX.

Processes and Commands

Определяет какой процесс или команду следует запустить и на какой машине это следует сделать. Например, команда может запускать приложения на Java или Python, в котором описан тестовый сценарий. Элемент «Process» может содержать множество дополнительных опций.

Expression Evaluation via Python

Позволяет делать вставки на Python.

Groups

STAX может исполнять набор команд последовательно или параллельно. Когда команды исполняются параллельно, STAX запускает каждую из них в отдельном потоке

Loops (циклы)

Помогают избежать многократного повторения одинакового кода.

STAX-Threads

При параллельном исполнении STAX использует пул потоков, что позволяет сократить ресурсозатраты по сравнению с использованием настоящих системных потоков.

Functions

Функции помогают структурировать код. Выделятся две главные роли: повторное использование кода и разложение на процедуры.

Monitoring STAX Jobs

Это приложение отображает графическое представление запущенных элементов тестового сценария. STAX Monitor помогает легко увидеть, какие процессы и команды STAF выполняются в данный момент, а также какие блоки в них содержатся. С помощью STAX Monitor можно выбирать блок и контролировать выполнение тестового задания, приостанавливать, возобновлять и снимать задачи. Также STAX Monitor отображает пройденные тесты и их результаты.

3. Методика тестирования

Тестирование проводится со специально выделенной тестирующей машины, на которой должен быть установлен STAX и запущен STAFProc. На всех тестовых машинах (AUX'ы и NUT'ы) должен быть установлен STAF и запущен STAFProc.

Тестирующая машина локально запускает тестовые скрипты, которые выполняют необходимые действия для проверки поведения системы в тех или иных условиях.

Test Case (тестовый случай) - артефакт, описывающий совокупность шагов, конкретных условий и параметров, необходимых для проверки реализации тестируемой функциональности или её части.

Под тест кейсом понимается структура вида:

Test Actions > Expected Result > Test Result

Пример:

Проверка на обнаружение SDC[1]

Test Actions	**Expected Result**	**Test Result (Passed/Failed)**
Discover SDC	SDC Discovered	Passed

Test Suite (тестовый набор) - комбинация Test Case'ов для проверки определенной части программного обеспечения, объединенной общей функциональностью или целями, преследуемыми запуском данного набора.

[1] SDC - Silent Data Corruption (Скрытое повреждение данных)

3.1 Параметры тестов

Для запуска теста ему нужно передать параметры: конфигурацию оборудования и параметры запуска. Через переменные окружения предаётся конфигурация оборудования, так как она общая для всех тестов. Параметры запуска для каждого теста различны, поэтому передаются как аргументы командной строки при запуске тестового скрипта, написанного на Python.

Файл конфигруации оборудования содержит:

- Список NUT
- Список AUX
- IP адреса каждого NUT'а и AUX'а
- Описание транспорта с клиентом
- Описание транспорта синхронизации
- Используемый транспорт каждого узла

3.3 Задачи, которые должна решать тестовая машина

- Согласно конфигурационному файлу оборудования проверить присутствие и работоспособность необходимых компонент распределённой системы тестирования. Если необходимо, установить компоненты и активировать.
- Согласно конфигурационному файлу оборудования (если необходимо) проверить и обновить тестируемое программное обеспечение.
- Запустить тестовые наборы согласно требованиям, указанным в конфигурационном файле.
- В случае зависания или экстренного завершения любого участника тестирования, по возможности продолжить процесс тестирования, восстановив работоспособность.

4. Реализация

STAX подразумевает использование XML файлов для описания сценария тестирования. Как было описано в предыдущем разделе, также могут быть использованы скриптовые вставки на языке Python. Интерпретатор Python реализованный в STAX использует API java для взаимодействия со STAF. Поэтому возникают сложности с использованием библиотек python'a, отвечающих за взаимодействие со STAF.

Чтобы избежать ограничений на описание тестовых сценариев в XML, основная нагрузка теста вынесена в отдельные библиотеки на Python (Внешний интерпретатор Python вызывается через функцию subprocess встроенного интерпретатора jython'a). В XML файле происходит выбор тестов удовлетворяющих конфигурации, последовательности их вызова и их параметров. После запуска теста можно отслеживать его состояние с помощью STAX Monitor.

4.1 run.sh

За запуск тестового сценария отвечает bash файл run.sh, которому необходимо передать путь к конфигурационному файлу и файлу с описанием хода тестового сценария. По умолчанию это файлы config_default.conf и stax_test.xml. Конфигурационные файлы хранятся в папке hw_configs. Файл с тестовыми сценариями хранится в корне тестовой системы.

4.2 stax_init.xml.

Это файл, в который вынесена подготовка к запуску тестового сценария. В нем осуществляется получение конфигурации оборудования и передача параметров общих для всех тестов в переменные окружения.

4.3 stax_test.xml

В этом файле описывается ход тестового сценария. Файл обязательно должен содержать вызов функции инициализации из файла stax_init.xml.

4.4 Запуск тестового сценария

После настройки конфигурации происходит выбор подходящих под условия тестовых сценариев и их запуск.

4.5 Анализ результатов

Тестовый сценарий, написанный на Python, может вернуть 2 типа результатов.

- Passed (Завершился корректно)
- Failed (Завершился с ошибкой)

После получения результата он сравнивается с результатом, который ожидался при запуске теста. Это могли быть такие же 2 результата. В итоге, можно получить следующие конфигурации:

Ожидаемый результат	Полученный результат	Результат тестирования
Passed	Failded	Failed
Passed	Passed	Passed
Failed	Failed	Passed
Failed	Passed	Failed

4.6 Логирование

У фреймворка STAF есть внешний сервис, отвечающий за логирование.

Бывает несколько уровней логов, некоторые из них:

- error
- warning
- info

Параметры запуска и результат тестирования записываются в лог STAF'а.

5. Апробация

4.1 Тестирование обнаружения SDC

Проверка работоспособности реализованной системы, осуществлена на примере обнаружения SDC.

За запуск теста отвечает скрипт run.sh. Ход теста описан в файле stax_test.xml, а конфигурация в файле config_default.conf, поэтому скрипт запускается без параметров.

Тестовому скрипту через переменные окружения передаются параметры конфигурации оборудования. После чего тестовый скрипт возвращает данные о конфигурации оборудования, которая может меняться динамически. На основании этих данных подбираются параметры тестового скрипта и происходит запуск.

После прохода каждого тестового сценария его результат сравнивается с ожидаемым результатом, и выносится вердикт о результате тестирования. Затем формируется полный отчет о результате тестирования, по которому можно легко отследить, какие тесты завершены успешно, а какие с ошибкой.

6. Заключение

В рамках данной курсовой работы были достигнуты следующие результаты:

- Исследованы возможности переиспользования XML based executive engine STAX.
- Реализована система автоматического запуска тестовых скриптов.
- Проведена апробация получившейся системы на примере конкретного теста.

7. Литература

[1] STAX Service User's Guid, URL: http://staf.sourceforge.net/current/STAX/staxug.html

[2] Python 2.7 ducumentation, URL: https://docs.python.org/2.7/

[3] STAF V3 User's Guide, URL: http://staf.sourceforge.net/current/STAFUG.htm