

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Уральский федеральный университет
имени первого Президента России Б.Н.Ельцина»**

**Уральский региональный центр
образования и разработок**

УДК
Код ГРНТИ

УТВЕРЖДАЮ

Директор Уральского регионального
центра образования и разработок
кандидат физико-математических
наук, доцент

_____ Асанов М.О.
«___» ноября 2011 г.

ОТЧЁТ
О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

по проекту № ООК 7 «Создание средств для упрощения реализации
трудоемких алгоритмов на языке Java и мониторинга их состояния»
в рамках Открытого окружного конкурса студенческих инициативных научных
исследований в области информатики и информационных технологий

вид отчета: итоговый

Руководитель НИР,
аспирант УрФУ

_____ Бурмистров И.С.

г. Екатеринбург, 2011

Список исполнителей

Научный руководитель,
аспирант УрФУ

_____ *Бурмистров И.С.*
подпись, дата (раздел(ы)_____)

Исполнители

студентка КН-302, института математики

_____ *Техажева С.А.*

и компьютерных наук УрФУ

_____ подпись, дата (раздел(ы)_____)

студент КН-302, института математики

_____ *Мухаметьянов Д.И.*

и компьютерных наук УрФУ

_____ подпись, дата (раздел(ы)_____)

студент КН-401, института математики

_____ *Плинер Ю.А.*

и компьютерных наук УрФУ

_____ подпись, дата (раздел(ы)_____)

Нормоконтролер

подпись, дата

Реферат

Объектом исследования является автоматизированная система по анализу работы трудоемких алгоритмов.

Цель работы — разработка системы, которая позволила бы удобным способом собирать статистику работы исследуемых алгоритмов и в визуальном виде представлять эту статистику.

В рамках научно-исследовательской работы была спроектирована и реализована система по управлению исследованиями, включающая в себя базу данных текстов — возможных входных данных для алгоритмов, сервис по запуску алгоритмов, веб-сервис для сбора статистики и визуализации результатов. Также была разработана концепция автоматизированного DI-контейнера, внедрение которого позволит существенным образом сократить и упростить конфигурирование любых приложений, написанных на языке Java, в том числе исследуемые алгоритмы, а также саму систему по управлению исследованиями.

Система по управлению исследованиями была реализована на языке Java SE.

Внедрение реализованной системы позволило одной исследовательской группе собрать в одном месте базу входных данных для исследуемых алгоритмов, результаты работы этих алгоритмов, а также автоматизировать процесс сбора статистики и визуализации результатов.

Степень внедрения системы по управлению исследованиями — процесс сбора статистики работы алгоритмов полностью переведен на реализованную систему. Веб-сервис для демонстрации текущего состояния исследований доступен по адресу <http://overclocking.usu.edu.ru>.

Степень реализации автоматизированного DI-контейнера — реализован прототип контейнера, полностью работоспособный, но ограниченный по функциональности.

Степень внедрения автоматизированного DI-контейнера — в код системы по управлению исследованиями внедрено использование DI-контейнера, который впоследствии практически незаметным образом будет заменен на автоматизированный DI-контейнер, когда тот будет полностью реализован.

Весь исходный код доступен по адресу <http://overclocking.googlecode.com/svn/webService/trunk>.

Содержание

Введение

Исследования трудоемких алгоритмов, работающих длительное время – достаточно тяжелая задача. Если алгоритм выполняется в течение нескольких часов или дней, то необходимо учитывать многие внешние факторы, например: неожиданное выключение света, отказ жесткого диска, параллельно выполняющиеся программы. Кроме того, необходима качественная система логгирования, чтобы в случае ошибки ее причину можно было бы понять по логам. Это означает, что к исследовательским задачам необходимо подходить как к задачам промышленной разработки ПО и использовать современные подходы этой области, например:

- покрывать код большим количеством автоматизированных тестов, чтобы минимизировать вероятность ошибки
- разрабатывать качественную архитектуру исследовательского проекта, чтобы максимально быстро вносить в него необходимые изменения
- использовать системы контроля версий, чтобы иметь возможность следить за историей изменения той или иной части проекта

Когда исследования достаточно обширные и требуют запуска алгоритма в различных условиях с разными параметрами, то появляются дополнительные трудности:

- алгоритм может запускаться на разных компьютерах, но должен использовать все время одни и те же входные данные
- все результаты работы необходимо собирать в одно место для анализа
- большое количество настроек вынуждает использовать конфигурационные файлы, за корректным заполнением которых необходимо следить

Если подходить к решению всех этих трудностей с должным уровнем внимания и качества, то может оказаться, что основное время при исследовании тратится на вещи, напрямую к исследованию не относящиеся: запуск алгоритмов с различными настройками, сбор результатов с большого количества компьютеров, отслеживание состояния всех запущенных программ.

Результаты НИР помогли одной исследовательской группе сконцентрироваться на непосредственной реализации алгоритмов и минимизировали накладные расходы на сбор и анализ результатов. В частности, в рамках НИР была проделана следующая работа:

а) реализована система для управления исследованиями, состоящая из нескольких частей:

- веб-сервис для отображения текущего состояния исследований и построения графиков по их результатам

- единое хранилище входных данных алгоритмов и их результатов

- сервисы для запуска алгоритмов

б) исследована возможность применения IDE-контейнера при реализации трудоемких алгоритмов и требования к его интерфейсу

в) исследованы существующие реализации IDE-контейнеров

г) разработан интерфейс автоматизированного IDE-контейнера, удовлетворяющего всем требованиям

В первой части отчета будут описаны результаты разработки системы для управления исследованиями. Вторая часть отчета посвящена обзору существующих IDE-контейнеров и обоснованию выбора автоматизированного IDE-контейнера для внедрения в код веб-сервиса и исследуемых алгоритмов.

Весь исходный код системы по управлению исследованиями, а также DI-контейнера, доступен по адресу <http://overclocking.googlecode.com/svn/webService/trunk>.

1 Система для управления исследованиями

При разработке архитектуры системы необходимо было учитывать следующие требования:

а) система должна обладать свойством масштабируемости, то есть увеличивать производительность при добавлении новых мощностей. Это необходимо в силу трудоемкости исследуемых алгоритмов. Важно уметь проводить исследования параллельно на нескольких компьютерах.

б) алгоритм должен иметь возможность сохранить любой промежуточный результат, чтобы впоследствии возобновить работу с некоторого состояния без перезапуска всего алгоритма. Это позволит каждый алгоритм декомпозировать на небольшие составные части, тем самым защищаясь от внешних факторов, таких как выключение света или крах оборудования.

в) входными данными для алгоритмов являются тексты достаточно большого размера (до нескольких гигабайт).

Система состоит из 7 основных частей:

а) NoSQL-база данных для хранения входных данных алгоритмов, а также результатов их работы

б) сервис импорта входных данных для алгоритмов

в) сервис запуска алгоритмов

г) SQL-база данных для хранения статистической информации по результатам запуска алгоритмов

д) сервис импорта статистической информации в SQL-базу

е) сервис экспорта статистической информации из NoSQL-базы данных в SQL-базу

ж) веб-сервис для отображения текущего состояния системы, а также построения графиков по результатам работы

Остановимся на каждой из частей подробнее.

1.1 NoSQL-база данных Cassandra

В качестве NoSQL-базы данных была выбрана широко известная база данных Cassandra, поскольку она обладает следующими характеристиками:

а) **распределенность**. Базу данных можно развернуть на нескольких физических компьютерах. При этом все данные могут храниться в нескольких копиях для защиты от их непредвиденной потери.

б) **масштабируемость**. Количество компьютеров, на которых развернута база данных, можно легко увеличивать.

в) **устойчивость к сбоям**. С базой данных по-прежнему можно работать, даже если некоторые ее узлы стали по какой-то причине недоступными.

г) **простота использования**. Cassandra проста как для администрирования, так и для использования в реальных приложениях, поскольку имеет уже написанных клиентов для большинства языков программирования.

Перед объяснением того, каким образом было организовано хранение данных в Cassandra, кратко объясним основные принципы работы этой базы данных.

1.1.1 Введение в модель данных Cassandra

Минимальным элементом данных в Cassandra является **столбец (column)**. Столбец – это тройка следующих элементов: имя, значение, метка времени. Имя и значение являются произвольными массивами байт. Пример столбца: (**имя**: «email», **значение**: «someEmail@example.com», **метка времени**: 1342323). Метки времени играют роль ревизий. Если в столбец с одним именем два раза записали значение с разными метками времени, то останется то, у которого метка времени больше. Например, если сначала была записана тройка («x», «y», 1), а затем – тройка («x», «z», 2), то значением столбца с именем «x» будет «z». При дальнейшем описании метки времени будут для нас не важны, поэтому мы часто будем под столбцом понимать пару (имя, значение).

Следующим элементом данных является **строка (row)**. **Строка** – это некоторый набор столбцов, имеющий ключ. То есть, по некоторому ключу мож-

но хранить произвольный набор колонок. Количество колонок в строке ограничено двумя миллиардами. Особенностью организации колонок в строке является то, что колонки хранятся в отсортированном виде по имени. Способ сравнения ключей в сортировке можно выбирать. Например, можно считать имена колонок строками, и сравнивать их лексикографически.

Следующий элемент данных – это **семейство колонок (column family)**. Это неограниченный набор строк, имеющий имя.

Несколько семейств колонок объединяются в **пространство ключей (keyspace)**. И, наконец, несколько пространств ключей объединяются в **кластер (cluster)**.

Таким образом, в Cassandra имеет место следующая иерархия данных: cluster -> keyspace -> column family -> row -> column.

1.1.2 Структура базы данных Cassandra в проекте по управлению исследованиями

Для хранения всех данных в проекте используется один кластер с названием «OverclockingCluster» с одним пространством ключей с названием «Overclocking».

Были спроектированы способы хранения данных для двух случаев:

- а) хранение больших объектов (например: файлы, картинки), представленных в виде массива байт (так называемые blob-ы)
- б) хранение небольших объектов, которые можно описать в виде некоторых сущностей (например, объект «пользователь»)

Рассмотрим способ хранения больших объектов. Для этого используется одно семейство колонок. Каждый такой объект имеет уникальный id и хранится в строке с ключом, равным этому id. Для того, чтобы записать объект в базу данных, выполняется следующая последовательность действий:

- а) Объект разбивается на небольшие части (например, по 1 мегабайту). Первой части соответствуют первые байты объекта, второй – следующие, и т.д.
- б) Каждая часть записывается в колонку с именем, равным ее номеру.
- в) В колонку с именем «length» записывается количество частей. Запись в эту колонку происходит строго после записи всех остальных колонок.

Данный способ хранения позволяет удобным образом впоследствии работать с объектом, а именно:

а) Объект можно читать частично.

б) По наличию или отсутствию колонки «length» можно понять, объект записан полностью или находится в процессе записи.

в) Можно следить за прогрессом в записи/чтении объекта.

Рассмотрим способ хранения небольших объектов-сущностей. Каждый такой объект имеет некоторый тип (тип «пользователь», тип «метаинформация» и т.д.). Также каждый объект имеет уникальный id. Объект записывается в семейство колонок с именем, равным типу объекта в строку с ключом, равным id объекта. Для записи объекта в базу данных выполняется следующая последовательность действий:

а) объект сериализуется в массив байт с помощью xml-сериалайзера

б) полученный массив байт записывается в качестве значения в колонку с именем «XmlData»

1.2 Сервис импорта входных данных

Мы рассмотрим сервис импорта на примере импорта файлов, поскольку для наших целей было необходимо импортировать именно файлы. Сервис импорта представляет из себя консольное приложение, которое необходимо запускать при необходимости. Приложению указывается путь к файлу, который необходимо импортировать. Сервис выполняет следующую последовательность действий:

а) проверяет, не был ли файл с таким именем уже импортирован. Для этого поддерживается соглашение, что id файла в базе данных совпадает с именем файла в файловой системе. При наличии такого соглашения достаточно проверить существование объекта с заданным id.

б) если файл еще не импортирован, то запускается процедура импорта. Она аналогична процедуре записи больших объектов, описанной в предыдущем пункте.

в) в случае успешной процедуры импорта сервис записывает также метainформацию о файле в виде отдельного объекта с тем же идентификатором. Метаинформация включает в себя:

- имя файла
- размер файла
- тип файла (что по смыслу представляет из себя файл. Например: простой текст, строка ДНК)
- тип содержимого файла (как файл хранится. на данный момент поддерживаются два типа: простой текст и архив GZip)

1.3 Сервис запуска алгоритмов

Данный сервис является центральным сервисом системы. Он спроектирован в виде консольного приложения, который автоматически диагностирует появление новых входных данных и запускает нужный алгоритм на этих данных. Более подробно, сервис выполняет следующие действия:

- а) Смотрит в базе данных актуальный список возможных входных данных.
- б) Смотрит в базе данных актуальный список результатов алгоритма.
- в) Если находятся входные данные, для которых еще не построен результат, то на этих данных запускается алгоритм.
- г) Результат алгоритма записывается в базу данных. Тем самым одновременно происходит фиксация того, на каких файлах алгоритм уже отработал.

Архитектура сервиса имеет следующие особенности:

- а) Сервис однопоточный. Это позволяет избежать большинства трудностей в реализации и избавляет от труднодиагностируемых ошибок в многопоточных сервисах.
- б) Для каждого типа алгоритмов может работать отдельный экземпляр сервиса. Это значит, что сервис обладает свойством масштабируемости.
- в) В текущей реализации не рекомендуется запускать более одного экземпляра сервиса для одного типа алгоритмов. Поскольку в этом случае велика вероятность, что на одних и тех же данных алгоритм отработает несколько раз, тем самым проделав лишнюю работу. Но при необходимости реализацию сервиса можно доработать таким образом, чтобы параллельная обработка задач одного типа была возможна.
- г) Сервис зависит только от базы данных. Это очень сильно упрощает его разворачивание и администрирование, поскольку необходимо задать только настройку доступа к базе данных.

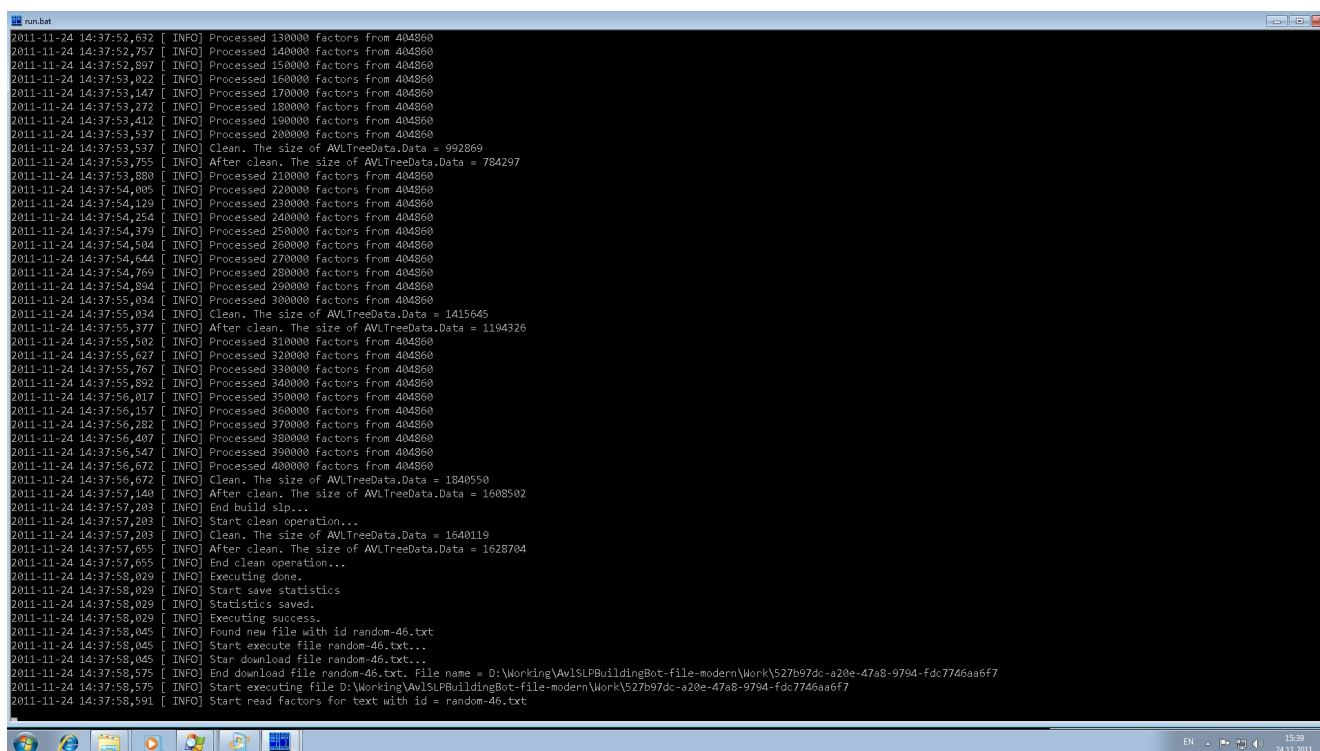
В ходе эксплуатации сервиса появились некоторые потребности, которые планируется реализовать в следующей версии:

а) Сервис должен уметь «забывать» результаты некоторых алгоритмов. Это полезно в ходе исследований, когда периодически обнаруживаются ошибки/неточности в алгоритме и возникает потребность пересчитать результаты. Сейчас это реализуется посредством удаления из базы данных соответствующих результатов.

б) Сервис должен обладать свойством отказоустойчивости. В текущей реализации на один тип алгоритма необходимо запустить один сервис. В случае, если он по какой-то причине перестанет работать, исследования будут приостановлены до тех пор, пока кто-то не перезапустит сервис.

На рисунке ниже представлен скриншот сервиса в процессе его работы.

Рис. 1.1. Пример консоли сервиса в процессе работы



```
run.bat
2011-11-24 14:37:52,632 [ INFO] Processed 130000 Factors from 404860
2011-11-24 14:37:52,757 [ INFO] Processed 140000 Factors from 404860
2011-11-24 14:37:53,097 [ INFO] Processed 150000 Factors from 404860
2011-11-24 14:37:53,022 [ INFO] Processed 160000 Factors from 404860
2011-11-24 14:37:53,147 [ INFO] Processed 170000 Factors from 404860
2011-11-24 14:37:53,272 [ INFO] Processed 180000 Factors from 404860
2011-11-24 14:37:53,412 [ INFO] Processed 190000 Factors from 404860
2011-11-24 14:37:53,537 [ INFO] Processed 200000 Factors from 404860
2011-11-24 14:37:53,537 [ INFO] Clean. The size of AVLTreeData.Data = 992869
2011-11-24 14:37:53,795 [ INFO] After clean. The size of AVLTreeData.Data = 784297
2011-11-24 14:37:53,880 [ INFO] Processed 210000 Factors from 404860
2011-11-24 14:37:54,005 [ INFO] Processed 220000 Factors from 404860
2011-11-24 14:37:54,129 [ INFO] Processed 230000 Factors from 404860
2011-11-24 14:37:54,254 [ INFO] Processed 240000 Factors from 404860
2011-11-24 14:37:54,379 [ INFO] Processed 250000 Factors from 404860
2011-11-24 14:37:54,504 [ INFO] Processed 260000 Factors from 404860
2011-11-24 14:37:54,644 [ INFO] Processed 270000 Factors from 404860
2011-11-24 14:37:54,769 [ INFO] Processed 280000 Factors from 404860
2011-11-24 14:37:54,894 [ INFO] Processed 290000 Factors from 404860
2011-11-24 14:37:55,034 [ INFO] Processed 300000 Factors from 404860
2011-11-24 14:37:55,034 [ INFO] Clean. The size of AVLTreeData.Data = 1415645
2011-11-24 14:37:55,377 [ INFO] After clean. The size of AVLTreeData.Data = 1194326
2011-11-24 14:37:55,502 [ INFO] Processed 310000 Factors from 404860
2011-11-24 14:37:55,627 [ INFO] Processed 320000 Factors from 404860
2011-11-24 14:37:55,767 [ INFO] Processed 330000 Factors from 404860
2011-11-24 14:37:55,892 [ INFO] Processed 340000 Factors from 404860
2011-11-24 14:37:56,017 [ INFO] Processed 350000 Factors from 404860
2011-11-24 14:37:56,157 [ INFO] Processed 360000 Factors from 404860
2011-11-24 14:37:56,282 [ INFO] Processed 370000 Factors from 404860
2011-11-24 14:37:56,407 [ INFO] Processed 380000 Factors from 404860
2011-11-24 14:37:56,547 [ INFO] Processed 390000 Factors from 404860
2011-11-24 14:37:56,672 [ INFO] Processed 400000 Factors from 404860
2011-11-24 14:37:56,672 [ INFO] Clean. The size of AVLTreeData.Data = 1840550
2011-11-24 14:37:57,140 [ INFO] After clean. The size of AVLTreeData.Data = 1608502
2011-11-24 14:37:57,203 [ INFO] End build slp...
2011-11-24 14:37:57,203 [ INFO] Start clean operation...
2011-11-24 14:37:57,203 [ INFO] Clean. The size of AVLTreeData.Data = 1640119
2011-11-24 14:37:57,655 [ INFO] After clean. The size of AVLTreeData.Data = 1628704
2011-11-24 14:37:57,655 [ INFO] End clean operation...
2011-11-24 14:37:58,029 [ INFO] Executing done.
2011-11-24 14:37:58,029 [ INFO] Start save statistics
2011-11-24 14:37:58,029 [ INFO] Statistics saved.
2011-11-24 14:37:58,029 [ INFO] Executing success.
2011-11-24 14:37:58,045 [ INFO] Found new file with id random-46.txt
2011-11-24 14:37:58,045 [ INFO] Start execute file random-46.txt...
2011-11-24 14:37:58,045 [ INFO] Start download file random-46.txt...
2011-11-24 14:37:58,575 [ INFO] End download file random-46.txt. File name = D:\Working\AvlSLPBuildingBot-file-modern\Work\527b97dc-a20e-47a8-9794-fdc7746aa6f7
2011-11-24 14:37:58,575 [ INFO] Start executing file D:\Working\AvlSLPBuildingBot-file-modern\Work\527b97dc-a20e-47a8-9794-fdc7746aa6f7
2011-11-24 14:37:58,591 [ INFO] Start read factors for text with id = random-46.txt
```

1.4 SQL-база данных для хранения статистической информации по результатам запуска алгоритмов

Для анализа результатов алгоритмов, построения каких-то статистических выборок и другой аналитической работы база данных Cassandra является неподходящим решением, поскольку не предоставляет удобного инструмента сложных запросов. Для этих целей хорошо подходят традиционные SQL-базы данных. Поэтому было принято решение статистические результаты работы алгоритмов (а именно: время работы при заданном входе, численный размер результата и другие) хранить в SQL-базе.

В качестве такой базы данных была выбрана широко известная база MySQL по следующим причинам:

а) MySQL – бесплатная база данных. Поэтому практически любой хостинг ее поддерживает.

б) Язык запросов MySQL достаточно функциональный и полностью удовлетворял нашим требованиям.

1.5 Сервис импорта статистической информации в SQL-базу

Сервис импорта представляет из себя веб-сервис с публичным API, имеющий несколько методов для загрузки новой информации по результатам алгоритма. Реализация сервиса как веб-сервиса с публичным API была выбрана по следующим причинам. Бесплатные хостинги имеют очень ограниченные ресурсы. Это значит, что единственное, чем может пользоваться веб-сервис – это легковесная база данных. Таким образом, все остальные сервисы системы управления исследованиями (прежде всего – база данных Cassandra и сервис запуска алгоритмов) будут расположены физически очень сильно удаленно от веб-сервиса. Значит, необходим интерфейс удаленного вызова. Веб-сервис с публичным API для этого очень хорошо подходит.

Сервис реализован на фреймворке **Spring MVC** поверх контейнера сервлетов **Apache Tomcat**. Остановимся на этих инструментах подробнее.

1.5.1 Контейнер сервлетов

Сервлет – это Java-программа, выполняющаяся на стороне сервера и расширяющая функциональные возможности сервера. Сервлет взаимодействует с клиентами посредством принципа «запрос»-«ответ». Все сервлеты реализуют один интерфейс, определяющий методы жизненного цикла.

Жизненный цикл сервлета

Управлением сервлетами осуществляет специальная программа, которая называется **контейнер сервлетов**. Жизненный цикл сервлета состоит из следующих шагов:

- а) Если сервлет отсутствует в контейнере, то контейнер создает сервлет и вызывает специальный метод инициализации.
- б) Если сервлет есть в контейнере, то он обрабатывает запрос в отдельном потоке.
- в) Если контейнеру необходимо удалить сервлет, он вызывает специальный метод для разрушения сервлета.

В качестве контейнера сервлетов был выбран **Apache Tomcat**.

1.5.2 Паттерн Model-View-Controller

Концепция **Model-View-Controller (MVC)** позволяет разделить данные, представление и обработку действий пользователя на три отдельных компонента:

- **Модель (Model)**. Модель предоставляет знания: данные и методы работы с этими данными. Не содержит информации, как эти знания можно визуализировать.

- **Представление (View)**. Отвечает за представление данных. Или, другими словами – за визуализацию модели. Например, в качестве View может выступать верстка веб-страницы.

- **Контроллер (Controller)**. Обеспечивает связь между моделью и представлением. Например, отвечает за обработку введенной пользователем информации и использует Model и View для реализации необходимой реакции.

Важно отметить, что как представление, так и контроллер зависят от модели. Однако модель не зависит ни от представления, ни от контроллера. Тем самым достигается назначение такого разделения: оно позволяет строить модель независимо от визуального представления, а также создавать несколько различных представлений для одной модели.

Вся мощь паттерна MVC используется при реализации веб-сервиса. Сервис импорта от данного паттерна использует только контроллер. Контроллер принимает извне информацию и сохраняет ее в базу данных. Для реализации паттерна MVC был использован фреймворк **Spring MVC**.

1.6 Сервис экспорта статистической информации из NoSQL-базы данных в SQL-базу

Сервис экспорта представляет из себя консольное приложение. Работа сервиса организована следующим образом:

а) По расписанию проверяют, не появилось ли в базе данных Cassandra новых результатов алгоритмов.

б) Если появились новые, но еще не экспортированные данные, сервис посредством корректного запроса к API сервиса импорта в SQL-базу закатывает в нее эти данные.

Архитектура сервиса обладает следующими особенностями:

а) Параллельно может работать несколько независимых экземпляров сервиса. При этом может случиться так, что одна и та же информация будет экспортирована несколько раз. Но экспорт отдельной сущности является быстрым действием, поэтому возможное выполнение такой работы несколько раз не критично.

б) Сервис зависит только от базы данных Cassandra и от адреса для вызовов методов API сервиса импорта.

1.7 Веб-сервис для отображения текущего состояния системы, а также построения графиков по результатам работы

Веб-сервис предоставляет пользователю следующую функциональность:

- а) Аутентификация. Для работы в сервисе необходимо пройти процедуру аутентификации по логину и паролю.
- б) Просмотр информации про возможные входные данные алгоритмов (например, тип, размер).
- в) Просмотр результаты работы алгоритмов в виде таблицы результатов.
- г) Построение графиков по результатам работы алгоритмов.

1.7.1 Процедура аутентификации

Опишем подробно, каким образом веб-сервис аутентифицирует пользователя и работает с механизмом сессий.

а) Первая страница, которую видит неаутентифицированный пользователь – **страница логина**. Данная страница содержит два поля ввода – по полю на логин и пароль, соответственно.

б) После вводом пользователем логина и пароля данные посылаются на сервер. Используя логин в качестве ключа, сервер находит в базе данных по данному логину информацию про пользователя.

в) Если пользователя с таким логином не найдено, то сервер возвращает страницу логина с ошибкой аутентификации.

г) Если пользователь с таким логином найден, то происходит сравнение паролей – хранимого в базе данных и пришедшего от клиента. Стоит заметить, что хранить в базе данных пароль в открытом виде небезопасно. В этом случае системный администратор (или любой другой человек, имеющий доступ к базе данных) имеет возможность узнать пароль любого пользователя. Поэтому в базе данных хранится **хеш от пароля**. При этом алгоритм хеширования выбирается таким образом, чтобы восстановление данных по их хеш-коду было невозможной или очень сложной задачей. Более точно, следует выбирать **криптографические хеш-функции**. Одним из таких алгоритмов является алгоритм **MD5**. Таким образом, при сравнении паролей сначала вычисляется значение хеш-функции па-

роля, введенного пользователем, после чего полученное значение сравнивается с записью в базе данных.

д) Если сравнение паролей дало отрицательный результат, то сервер возвращает страницу логина с ошибкой аутентификации.

е) В случае положительного результата сравнения паролей сервер считает, что пользователь успешно прошел процедуру аутентификации. Сервер выдаёт пользователю **сессию**. Физически это означает, что в памяти веб-сервиса по случайному уникальному идентификатору сохраняется объект, в котором описаны параметры сессии – идентификатор пользователя и другая сопутствующая информация. Также в cookies пользователя с именем «SessionId» сохраняется идентификатор сессии. После этого сервер перенаправляет пользователя на одну из страниц веб-сервиса.

ж) При каждом последующем запросе пользователя к серверу сервер проводит следующие действия:

1) Если у пользователя выставлена кука «SessionId», и в хранилище сессий есть сессия с соответствующим идентификатором, то система считает, что пользователь аутентифицирован, и возвращает ему ответ на его запрос.

2) Если кука не выставлена, или сессия не найдена в хранилище сессий, то пользователь считается неаутентифицированным, и его перенаправляют на страницу логина, на которой он проходит процедуру аутентификации.

2 Автоматизированный DI-контейнер

При разработке больших сложных систем очень важную роль играет тестирование. Очень удобно, если имеется достаточное количество автоматизированных тестов, поскольку их можно запускать в автоматическом режиме, и почти мгновенно получать результат. Чем лучше код покрыт тестами, тем проще находить ошибку: при правильном подходе ошибка воспроизведется в небольшом простом тесте.

Однако далеко не каждую систему можно покрыть автоматизированными тестами. Поэтому одна из основных тенденций в области программирования в последнее время – это проектирование архитектуры программных систем таким образом, чтобы их было легко тестировать. Чтобы дизайн кода удовлетворял этому требованию, необходимо придерживаться нескольких базовых принципов:

а) бить код на множество мелких классов, каждый из которых отвечает за небольшую часть функциональности: чем меньше класс, тем проще тестировать его поведение

б) тестировать класс сразу после его реализации или даже до реализации: чем раньше начать тестирование, тем проще понять, удобно ли построена архитектура

в) строить зависимости классов только от интерфейсов, а не от других классов: в этом случае в целях тестирования есть возможность подменять все модули, от которых зависит тестируемый класс, обеспечивая таким образом изолированное тестирование

Последний из перечисленных принципов носит название *dependency inversion principle*, что дословно означает «принцип инверсии зависимостей»

Чтобы код удовлетворял этому принципу, следует придерживаться следующих правил:

а) у каждого класса должен быть интерфейс, который он реализует

б) все классы должны принимать все интерфейсы, от которых они зависят, в конструкторе

Если следовать этим двум правилам, то возникает вопрос: в каком месте приложения будут создаваться необходимые классы? Обычно создание всей иерархии классов выделяют в специальный класс-конфигуратор. При этом воз-

никает проблема, что в достаточно большой системе этот класс будет очень большим, а для его создания будет нужно провести ряд однотипных действий. Кроме того, в процессе разработки этот класс будет постоянно меняться, а для его поддержки будет требоваться прилагать все больше и больше усилий.

Для автоматизации процесса конфигурирования были придуманы **dependency injection-контейнеры (DI-контейнеры)**. Существует большое количество DI-контейнеров, и каждый из них обладает своими особенностями. Мы рассмотрим две реализации DI-контейнеров для языка Java, после чего предложим альтернативный подход.

2.1 BeanFactory – контейнер фреймворка Spring

Рассмотрим один из самых распространенных контейнеров – BeanFactory.

Конфигурации с помощью конструкторов. Пример конфигурирования: `<bean id=«beanId» class=«beanClass»>
<constructorArg [аттрибуты]>
</bean>`

BeanId – с его помощью можно получить готовый экземпляр из BeanFactory, вызвав метод `factory.getBean(beanId)`. **BeanClass** – класс, экземпляр которого мы хотим получить. Атрибуты в `constructorArg` (под аргументами подразумеваются аргументы конструктора):

- а) `value` – значение, которое хотим передать в конструктор
- б) `type` – тип аргумента, которому хотим установить требуемое значение
- в) `index` – номер аргумента, которому хотим установить требуемое значение
- г) `name` – имя аргумента, которому хотим установить требуемое значение
- д) `ref` – `id bean'a`, который хотим подставить в качестве требуемого аргумента

Таким образом, правила для разрешения всех зависимостей задаются с помощью xml-файла.

Большое количество конфигурирования в виде xml неудобно тем, что это очень слабо согласуется с кодом, в результате чего вероятно большое количество ошибок, вызванных неправильной версией конфигурационного файла.

2.2 Google Guice

Google Guice, контейнер от компании Google, предлагает другой подход. Для его конфигурирования необходимо проделать следующие вещи:

- а) Пометить конструктор класса, который планируется создавать через контейнер, атрибутом `@Inject`
- б) Для каждого интерфейса, который планируется получать через контейнер, указать класс, его реализующий.

Таким образом, в контейнере guice ушли от конфигурирования посредством xml. Это более удобный способ, поскольку всё конфигурирование описано прямо в коде, что позволяет следить за изменениями/обновлениями. Между тем, данный способ обладает рядом недостатков:

- а) Количество кода, необходимого для конфигурирования, мало отличается от способа без контейнера: всё равно необходимы большие классы – конфигураторы.
- б) Использование атрибутов «заражает» весь код конкретной библиоткой контейнера. Это значит, что впоследствии перейти на использование какого-то другого контейнера будет гораздо сложнее.

Проанализировав недостатки описанных выше контейнеров, мы решили предложить другой подход

2.3 Концепция автоматизированного DI-контейнера

Заметим, что в программировании есть подход, который говорит о том, что иногда разумно использовать в коде набор некоторых соглашений, вместо того чтобы сложным образом конфигурировать систему. Вот небольшой список соглашений/тенденций, использование которых могло бы существенным образом упростить конфигурирование системы:

- а) у большинства интерфейсов системы ровно одна реализация
- б) в большинстве случаев у классов ровно один публичный конструктор

Если отталкиваться от этих двух простых правил, то можно реализовать **автоматизированный DI-контейнер**. В основе концепции автоматизированного DI-контейнера лежат следующие принципы:

а) В большинстве случаев контейнер без всякой конфигурации может понять, какой класс необходимо использовать в качестве реализации интерфейса, поскольку в большинстве случаев реализация ровно одна.

б) В большинстве случаев контейнер без всякой конфигурации может понять, каким образом необходимо создавать объект, поскольку у него ровно один публичный конструктор.

Таким образом, объем кода/файлов для конфигурации может существенно сократиться, если следовать этим принципам. На текущий момент полнофункциональный автоматизированный DI-контейнер еще не реализован, однако в код системы по управлению исследованиями внедрено использование контейнера, который необходимо полностью конфигурировать. Вся конфигурация выделена в отдельный модуль. После реализации автоматизированного DI-контейнера эту конфигурацию можно будет либо существенно сократить, либо полностью удалить.

Заключение

В результате НИР была проделана следующая работа:

а) полностью реализована и внедрена система по управлению исследованиями, позволившая автоматизировать процесс анализа работы трудоемких алгоритмов

б) разработана концепция автоматизированного DI-контейнера

Результаты НИР являются успешными, поскольку уже сейчас видно существенное упрощение процесса сбора статистики о работе исследуемых алгоритмов. Однако уже сейчас видно, каким образом текущие результаты можно улучшить и вывести разработанную систему на другой уровень.