**Федеральное государственное автономное образовательное учреждение   
высшего образования**

**КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ**

**ВЫСШАЯ ШКОЛА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И   
ИНФОРМАЦИОННЫХ СИСТЕМ**

Направление подготовки: 09.03.03 Прикладная информатика

Профиль: Прикладная информатика в экономике

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**Разработка клиент-серверного фреймворка для высоконагруженных вычислений на базе Linux и MPI**

**Работа завершена:**

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г.

Студент группы 11-308 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_И.А. Габидуллин

**Работа допущена к защите:**

Научный руководитель

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_А.С. Тощев

Директор Высшей школы ИТИС

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Ф. Хасьянов

Казань – 2017 г.

Содержание

[Введение 4](#_Toc482550116)

[1 Обзорно-аналитическая часть 6](#_Toc482550117)

[1.1 Принципы параллельных вычислений 6](#_Toc482550118)

[1.2 Технологии параллельных вычислений 7](#_Toc482550119)

[1.3 Обзор средств управления параллельными вычислениями 8](#_Toc482550120)

[1.4 Описание фреймворка 9](#_Toc482550121)

[2 Технологическая часть 11](#_Toc482550122)

[2.1 Язык программирования 11](#_Toc482550123)

[2.2 Средства разработки 12](#_Toc482550124)

[2.3 Средства проектирования 13](#_Toc482550125)

[2.4 Разработка серверной части 14](#_Toc482550126)

[2.5 Разработка клиентской части 15](#_Toc482550127)

[3 Проектирование и разработка фреймворка 16](#_Toc482550128)

[3.1 Проектирование фреймворка 16](#_Toc482550129)

[3.1.1 Структура проекта 16](#_Toc482550130)

[3.2 Разработка серверной части 19](#_Toc482550131)

[3.2.1 Разработка сервиса параллельных задач 19](#_Toc482550132)

[3.2.2 Разработка сервера управления 19](#_Toc482550133)

[3.3 Разработка клиентской части 19](#_Toc482550134)

[3.3.1 Разработка консольного приложения 19](#_Toc482550135)

[3.4 Взаимодействие клиента с сервером 19](#_Toc482550136)

[Заключение 20](#_Toc482550137)

[Список использованных источников 21](#_Toc482550138)

[Приложение 22](#_Toc482550139)

# Введение

В мире существует множество задач, которые можно разделить на мелкие подзадачи и решать их параллельно. Особенно часто это встречается в науке. Разнообразные математические преобразования и вычисления. И в мире, в котором даже на мобильных телефонах по несколько процессоров, необходимо использовать возможности параллельных вычислений.

Параллельные вычисления уже давно стали важной частью Computer Science и активно используются на практике. Во многих научных организациях, например, в Казанском Федеральном Университете существует кластер электронно-вычислительных машин для выполнения параллельных вычислений на базе технологии MPI. Но не существует простого и удобного способа управлять вычислениями: разворачивать программный код, запускать, отслеживать, получать результаты или данные о скорости работы. Сейчас всё это выполняется прямо на кластере, пользователь подключается к кластеру, загружает нужные ему файлы и запускает процесс.

Исходя из вышеизложенного, было принято решение создать клиент-серверный фреймворк для работы с параллельными вычислениями. Данный фреймворк должен позволять гибко настраивать взаимодействия с сервером параллельных вычислений. Единожды развернув и настроив фреймворк пользователь получает удобный и гибкий способ управления своими вычислениями. Пользователь получает возможность организовать работу извне, не подключаясь напрямую к высокопроизводительному кластеру. Таким образом, данная работа несет в себе практическую пользу в научной сфере.

Фреймворк базово должен поддерживать такие вещи как:

1. загрузка программы на сервер;
2. выполнение программы;
3. получение статистики;
4. возможность управлять ходом работы программы;
5. возможность легко дополняться и изменяться.

Данная дипломная работа посвящается разработке клиент-серверного фреймворка для управления параллельными MPI задачами.

Для достижения цели дипломной работы необходимо было выполнить следующие задачи:

1. проектирование архитектуры фреймворка, клиентской и серверной части;
2. проектирование архитектуры взаимодействия клиентской и серверной части;
3. изучение и применение найденных способов управления процессами в операционной системе;
4. разработка серверной части фреймворка;
5. разработка клиенсткой части фреймворка.

# Обзорно-аналитическая часть

## Принципы параллельных вычислений

В 1966 году профессором Стэндфордского университета Майклом Финном была предложена, так называемая «Таксономия Флинна» (Flynn's taxonomy). В ней электронно-вычислительные машины были разделены по принципу параллелизма в потоках команд и потоках данных. Рассмотрим её:

* ОКОД — система, которая работает с одиночным потоком команд и одиночным потоком данных (Single Instruction stream over a Single Data stream);
* ОКМД — система, которая работает с одиночным потоком команд и множественным потоком данных (Single Instruction, Multiple Data);
* МКОД — система, которая работает со множественным потоком команд и одиночным потоком данных (Multiple Instruction Single Data);
* МКМД — система, которая работает со множественным потоком команд и множественным потоком данных (Multiple Instruction Multiple Data).

[ссылка <https://en.wikipedia.org/wiki/Flynn%27s_taxonomy>]

Архитектура ОКОД — это традиционный компьютер [фон-Неймановской архитектуры](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D1%84%D0%BE%D0%BD_%D0%9D%D0%B5%D0%B9%D0%BC%D0%B0%D0%BD%D0%B0) с одним процессором, который выполняет последовательно одну инструкцию за другой, работая с одним потоком данных. ОКОД не является параллельной архитекутрой. МКОД же является очень редко используемой архитектурой. На практике чаще всего используются такие архитектуры как ОКМД и МКМД.

Также ОКМД и МКМД разделяются на подклассы:

* системы с общей памятью (shared memory, SM);
* системы с распределенной памятью (distributed memory, DM).

Таким образом, фреймворк должен поддерживать архитектуры ОКМД и МКМД. Или же иметь возможность добавить поддержку новой архитектуры, не изменяя общей структуры проекта.

## Технологии параллельных вычислений

Существуют несколько крупных и используемых технологий для параллельных вычислений. Рассмотрим некоторые из них:

* MPI (Message Passing Interface) — программный интерфейс ([API](https://ru.wikipedia.org/wiki/API)) для передачи [информации](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8F), который позволяет обмениваться сообщениями между процессами, выполняющими одну задачу. Существуют реализации для языков Fortran, Java, C и C++;
* OpenMP (Open Multi-Processing) — открытый стандарт для распараллеливания программ на языках C, C++ и Fortran. Дает описание совокупности директив компилятора, библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточных приложений на многопроцессорных системах с общей памятью;
* CUDA (Compute Unified Device Architecture) — программно-аппаратная архитектура параллельных вычислений, которая позволяет существенно увеличить вычислительную производительность благодаря использованию графических процессоров фирмы Nvidia. Для разработки используется диалект языка программирования С.

Каждая из данных технологий направлена на решение своих специфических задач.

MPI ориентирован на системы с распределенной памятью, то есть когда затраты на передачу данных велики. Основным средством коммуникации между процессами в MPI является передача сообщений друг другу.

OpenMP используется же для систем, которые имеют общую память. Реализация данной технологии имеет несколько потоков (thread), один из них главный (master), остальные подчиненные (slave). Главный поток создает задачи, которые подчиненные выполняют. Программирование осуществляется при помощи специальных директив.

CUDA использует мощь графических процессоров для произведения разнообразных вычислений. Данная технология является самой молодой. Архитектура CUDA даёт разработчику возможность по своему усмотрению организовывать доступ к набору инструкций графического ускорителя и управлять его памятью.

Самой распространенной технологий из этих трёх является MPI, так как она позволяет наиболее гибко распределять задачи между процессорами и потоками, также при помощи неё можно использовать систему с распределённой памятью, что очень важно в кластерах. Но оставшиеся две технологии также являются сильными технологиями для своих задач и не стоит забывать о них.

## Обзор средств управления параллельными вычислениями

Есть несколько проектов, которые позволяют управлять параллельными вычислениями. Большинство из них являются коммерческими проектами. Так как разрабатываемый проект не является коммерческим, рассмотрим некоторые некоммерческие программы.

PBS (Portable Batch System) — система управления [распределенными вычислениями](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B5_%D0%B2%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F). Основная функция PBS — запуск вычислительных задач в вычислительной среде по [расписанию](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D0%B8%D1%81%D0%B0%D0%BD%D0%B8%D0%B5). Наиболее часто используется для управления вычислительным процессом в [кластерах](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80_(%D0%B3%D1%80%D1%83%D0%BF%D0%BF%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BE%D0%B2)). Является одной из самых старых систем. На основе неё создано более современные системы управления, имеющие большее количество возможностей.

TORQUE (Terascale Open-Source Resource and QUEue Manager) — менеджер [распределенных ресурсов](https://ru.wikipedia.org/wiki/%D0%A0%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D1%91%D0%BD%D0%BD%D1%8B%D0%B5_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D1%8B) для вычислительных [кластеров](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%82%D0%B5%D1%80_(%D0%B3%D1%80%D1%83%D0%BF%D0%BF%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%BE%D0%B2)) из [машин](https://ru.wikipedia.org/wiki/%D0%A5%D0%BE%D1%81%D1%82) под управлением [Linux](https://ru.wikipedia.org/wiki/Linux) и других [Unix-подобных](https://ru.wikipedia.org/wiki/UNIX-%D0%BF%D0%BE%D0%B4%D0%BE%D0%B1%D0%BD%D0%B0%D1%8F_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0) [операционных систем](https://ru.wikipedia.org/wiki/%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0). Является современной версией PBS. Одна из самых развитых систем управления распределенными вычислениями, поддерживается большим организаций и научных лабораторий.

Данные проекты развивались долгое время и имеют большое количество возможностей. Но также они являются сложными в установке и настройке системами. Фреймворк же должен быть лёгок как в настройке, так и установке, иметь возможность легко подвергаться модификациям. А также не исключать возможности совместного использования вместе с указанными системами.

## Описание фреймворка

Рассмотрим функции, которые должен способен выполнять фреймворк в базовой версии:

1. создать задачу для выполнения параллельных вычислений, при этом может загружаться программа и данные из клиента на сервер или же программа и данные будут сразу находиться на сервере, если они будут создавать слишком большую нагрузку протоколу HTTP;
2. начать выполнение задачи с указанным количеством потоков;
3. остановить запущенную ранее задачу;
4. получить статус задачи, выполняется она или нет;
5. удаление задачи с сервера, удаление параллельной программы, а также всех данных.

Возможные дополнительные функции:

1. получить результаты работы параллельной программы;
2. получить метрики (загруженность, время выполнения) выполнения параллельной программы.

Данный функционал будет покрывать большинство потребностей в управлении задачами параллельных вычислений. Также фреймворк не обязательно использовать лишь с параллельными задачами, можно запускать обычно трудоёмкие задачи на отдельном сервере, произведя лишь небольшую настройку.

# Технологическая часть

Важной частью в разработке какого-либо проекта является выбор инструментов и технологий, которые будут использоваться.

Выбирать следует исходя из целей и задач, так как неверно выбранная технологическая база в дальнейшем может помешать разработке. Или же в худшем случае придется начать работу сначала, используя уже подходящие технологии.

В сфере разработки программного обеспечения следует выделять следующие аспекты, в которых нужно выбирать те или иные инструменты:

1. Язык программирования и программная платформа (фреймворк);
2. Средства разработки;
3. Средства проектирования.

В данной главе будет рассмотрен выбор тех или иных инструментов и технологий для каждого аспекта разработки и проектирования, которые будут использоваться на данном проекте.

## Язык программирования

В качестве языка разработки был выбран C# 7-ой версии. Данный язык программирования подходит для разработки разнообразных видов приложений: приложений для мобильной платформы, веб приложений, бизнес приложений, а также высоконагруженных систем. По этой причине данный язык как нельзя подходит для данного проекта, так как в нём сочетается разные виды приложений. Используя данный язык можно написать, как клиентскую часть, так и серверную.

Также данный язык выбран из-за того, что он достаточно популярный и распространенный. Разработчиков, которые знакомы с ним и работают используя его очень большое количество и найти таких специалистов не составляет труда. По индексу TIOBE (индекс, оценивающий популярность языков программирования) язык программирования C# на май 2017 года занимает 5 место по популярности (ссылка наhttps://www.tiobe.com/tiobe-index/).

В каждой новой версии C# добавляются новые возможности для разработчика и становится всё больше способов сделать программный код более структурированным и читабельным. Например, в последних версиях языка добавились такие возможности как интерполяция строк и локальные функции.

Язык C# поддерживается несколькими фреймворками, в дальнейшем будет описано какие именно фреймворки были выбраны для клиентской и серверной части проекта.

## Средства разработки

Для того, чтобы воспользоваться самыми новыми возможностями языка программирования, следует воспользоваться самой новой средой разработки. В данном случае выбор пал на Visual Studio 2017 Community.

Данная среда позволяет быстро и качественно писать код, так как в ней существует большое количество подсказок, которые могут помочь разработчику быстрее и качественнее создавать программный код. В ней удобно следить за исполнением кода. Visual Studio предоставляет множество данных о производительности создаваемой программы, что очень важно в разработке. Также данная среда поддерживает большое количество языков программирования, а значит программисту не надо осваивать большое количество сред разработки, а достаточно лишь изучить одну.

При помощи Visual Studio можно легко собирать проекты, а также разворачивать, либо настраивать развёртывание при необходимости.

Для тестирования веб-сервера использовалось расширение для веб-браузера Opera – RestMan. Плагин легко устанавливается из магазина расширений веб-браузера. А благодаря интуитивному и удобному интерфейсу, расширением достаточно легко научиться пользоваться.

Данное расширение позволяет отправлять REST веб запросы любого вида к указанному веб серверу и получать ответы. Ответы же отображаются в удобном для анализа виде.

## Средства проектирования

Проектирование – это очень важная часть разработки. Надо иметь способ легко создавать нужные для проекта диаграммы. Средой проектирования была выбрана компьютерную программу Visual Paradigm Community Edition.

Данная программа является бесплатной для некоммерческого использования. Программа поддерживает все виды UML диаграмм. Программа позволяет сохранять диаграммы в одном проекте, в следствии чего очень удобно создавать большое количество диаграмм для одного проекта.

К недостаткам можно отнести не совсем очевидный пользовательский интерфейс.

## Разработка серверной части

Так как языком разработки был выбран C#, то выбор фреймворка ограничивается разными видами .NET Framework. В свою очередь .NET Framework имеет множество версий, между которыми можно выбирать:

1. .NET Framework – классический Windows фреймворк. Является старейшей платформой среди .NET платформ. Поддерживает большое количество типов приложений (Windows приложение, веб-приложение, мобильное приложение). Также платформа имеет большие возможности взаимодействия с системой. Но данная платформа поддерживает лишь семейство операционных систем Microsoft Windows. Поэтому данная программная платформа не подходит для данного проекта;
2. Mono – некоммерческий кроссплатформенный (Linux, MacOS, Windows) фреймворк. Является перенесенной на другие операционные системы .NET Framework. Но с некоторыми недоработками;
3. .NET Core – является разработанной компанией Microsoft кроссплатформенной программной платформой. Является самой молодой из данных проектов. В данной платформе используются наработки Mono. Данная платформа активно развивается и добавляет всё большее количество функций из классической .NET Framework, но для большего количества платформ.

Таким образом, стоит выбор между .NET Core и Mono. Был выбран .NET Core. Данная платформа поддерживает все нужные для проекта возможности, позволяет создать кроссплатформенный код. Также с данной платформой легче начать работу, так как Visual Studio 2017 сразу после установки поддерживает разработку для этой платформы.

## Разработка клиентской части

Предполагается то, что клиентская часть можно создать, используя любую платформу, так как взаимодействие с серверной частью происходит через протокол HTTP (Web API).

В данной работе будет создано консольное приложение, которое взаимодействует с сервером по протоколу HTTP. Разработка будет происходить на платформе .NET Core для кроссплатформенности.

Предполагается, что, пользователь данного проекта может с лёгкостью добавить свои новые функции или создать абсолютно новый клиент. Например, сайт, который будет выполнять роль клиента. Благодаря выбранному протоколу взаимодействия и открытому исходному коду серверной части, можно такой клиент легко создать.

# Проектирование и разработка фреймворка

## Проектирование фреймворка

### Структура проекта

В основу архитектуры проекта положена модель слоёв, описанная Мартином Фаулером в книге «Архитектура корпоративных программных приложений» [ссылка]. Архитектура заключается в том, что проект разбивается на несколько независимых слоёв (layer). Таким образом достигается гибкая для модернизации и изменений структура, что в случае с фреймворком очень важно.

В проекте существует несколько слоёв:

* engine layer (базовый слой – движок, в котором проводятся все минимальные, простые манипуляции);
* manager layer (слой который управляет engine layer, а также предоставляет доступ к его вычислениям и результатам);
* presentation layer (слой который отвечает за отображение данных в любом виде, будь то консольное или веб приложение).

Рассмотрим Presentation layer. Presentation layer представляет из себя консольное кроссплатформенное приложения которое общается с сервером посредством HTTP запросов. Консольное приложение генерирует специфический HTTP запросы по указанному адресу сервера.

Сервер представляя из себя Manager layer. Его можно назвать Task Manger, то есть он отвечает за создание и управление задачами. Сам Task Manager общается с другими существующими менеджерами:

* Process manager – взаимодействует с запущенными задачами, может их останавливать и получать статус задачи;
* Parallel manager – запускает параллельные задачи (параллельную программу) с нужным количеством потоков.

По аналогии с менеджерами существует два движка, Process engine и Parallel engine, они выполняют те же функции что и менеджеры, но на более низком уровне, менеджеры управляют ими.

Presentation layer взаимодействует с Task manager по протоколу HTTP, в то время как Task manager управляет двумя другими менеджерами напрямую через их интерфейсы.

Разбивку на слои можно отобразить при помощи диаграммы. На рисунке 1 можно наглядно увидеть каким именно образом и на какие слои разбивается проект.

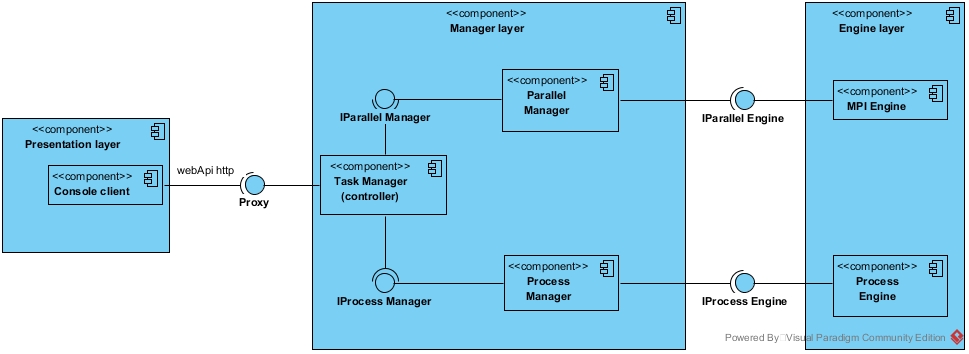


Рисунок 1. Структура слоёв фреймворка

На рисунке 2 можно изучить структуру решения фреймворка, на какие именно под проекты решение разбивается и как происходит взаимодействие между ними.

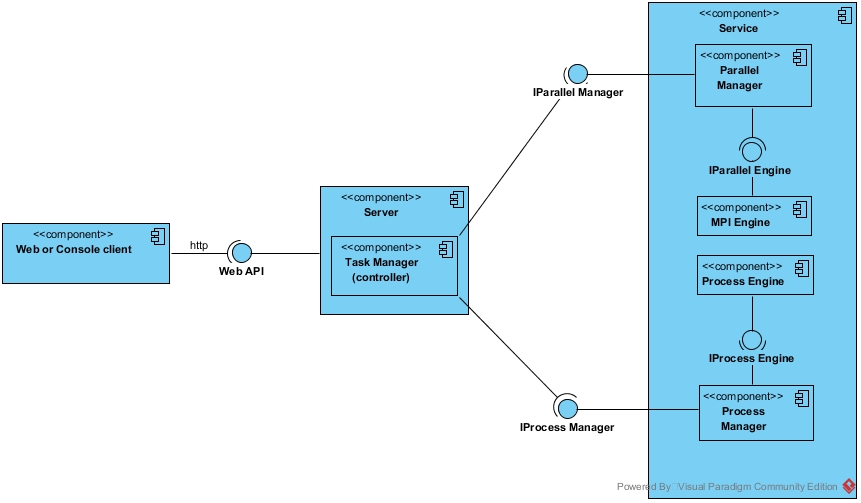


Рисунок 2. Структура решения фреймворка

Данная диаграмма будет полезна разработчикам, которым нужно внести какие-либо изменения в фреймворк, так как на ней наглядно отображено где и какие модули проекта располагаются.

Таким образом, получается структура, в которой легко можно разобраться и добавить нужный функционал. Например, движки создаются через программный интерфейс и для того, чтобы подменить вызов MPI задач на вызов OpenMP задач, надо всего лишь реализовать интерфейс IParallel Engine для OpenMP задач и вызывать данную реализацию в реализации Parallel manager.

Из-за универсальности Web API легко можно создать новые клиенты для сервера. Например, веб сайт, который облегчит управление сервером, а также позволит детально и наглядно изучать данные, которые будут возвращаться сервером. Причем клиент можно создать на любой технологии, главное, чтобы была возможность генерировать HTTP запросы.

## Разработка серверной части

### Разработка сервиса параллельных задач

### Разработка сервера управления

## Разработка клиентской части

### Разработка консольного приложения

## Взаимодействие клиента с сервером

# Заключение

# Список использованных источников

# Приложение