электронный научно-технический журнал
ИНЖЕНЕРНЫЙ ВЕСТНИК
Издатель: Общероссийская общественная организация "Академия инженерных наук им. А.М. Прохорова". Эл No. ФС77-51036. ISSN 2307-0595

Принципы построения масштабируемой Web-ориентированной системы для оценки качества Парето-аппроксимации при решении задачи многокритериальной оптимизации

09, сентябрь 2017
Грошев С. В.[1,*], Пивоварова Н. В.[1,**]
УДК: 519.6

[1]Россия, МГТУ им. Н.Э. Баумана
[*]sgro@newmail.ru
[**]pivovarova@bmstu.ru

Введение

Достаточно часто в реальных ситуациях качество эксплуатации исследуемого объекта или системы оценивается не единственным критерием или показателем качества, а совокупностью таких критериев, являющихся одинаково значимыми. Задачи такого типа относятся к задачам многокритериальной оптимизации (МКО), и являются важным этапом при проектировании и оценке проектного объекта.

Существует большое число методов, позволяющих решить эту задачу. Относительно новым и прогрессирующим классом МКО-методов являются методы, предполагающие предварительное построение некоторой конечномерной аппроксимации множества решений, получивших название Парето-аппроксимация [1].

В настоящее время существует довольно много методов Парето-аппроксимации, что ставит вопрос сравнительной оценки эффективности этих методов [3]. Как показано в [2], многие программные комплексы, реализующие эти алгоритмы, не предоставляют быстрый доступ и не обладают удобным для выполнения расчетов и анализа интерфейсом.

В данной работе рассматривается разработка сервисно-ориентированной системы для выполнения и оценки качества Парето-аппроксимации. Основной целью этой системы является обеспечение быстрого и удобного доступа к алгоритмам решения и оценки Парето-аппроксимации, без привязанности к конкретной операционной системе или приложению.

Вопросы постановки МКО-задачи, а также алгоритмы построения П-аппроксимации и оценка качества аппроксимации подробно рассмотрены в работах [1-3]. В представленной работе рассматриваются ключевые проблемы, возникающие при разработке удобного клиента, легко масштабируемого сервера и интеграции различных функциональных модулей в систему, а также способы их преодоления. В качестве приложения в составе сервисной системы представлена реализация браузерного приложения.

1. Основные функциональные требования к системе

В предыдущих работах авторов показано [2,3], что существуют программные комплексы, позволяющие решать задачи многокритериальной оптимизации различными методами. Однако бывает необходимо оценить результаты решений разными методами одной и той же задачи. Сделать это быстро и удобно, существующие на данный момент системы не позволяют.

В связи с указанными причинами, были выделены следующие основные функциональные требования к разрабатываемой системе:

- возможность построения Парето-аппроксимации с выбором алгоритма расчета и тестовой задачи;
- возможность настройки параметров алгоритма расчета и тестовой задачи;
- возможность проведения сравнительного статистического анализа стохастических алгоритмов с использованием методов оценки, основанных на индикаторах качества, эмпирических функциях достижимости и доминантном ранжировании;
- возможность расчета унарных или бинарных индикаторов для одной или двух Парето-аппроксимаций соответственно;
- предоставление удобного интерфейса взаимодействия с расчетами модулями;
- предоставление справочных данных об алгоритмах и их параметрах для повышения удобства работы с системой;
- возможность систематизации и хранения данных о расчётах в единой системе.

Была поставлена задача разработки масштабируемого межплатформенного приложения для выполнения Парето-аппроксимации и оценки ее качества. Приложение должно обеспечивать одновременную работу нескольких пользователей: одновременное проведение нескольких расчетов, одновременный доступ множества пользователей к базе данных системы, одновременная обработка множества веб-запросов. Кроме того, приложение должно включать в себя систему регистрации, аутентификации. Так же необходимо организовать систему авторизации для обеспечения возможности работы нескольких пользователей в рамках одного проекта.

Пользователь должен иметь возможность:

- создавать расчет Парето-аппроксимации;
- настраивать параметры алгоритма и задачи;
- получать результаты расчета;
- получать результаты оценки качества расчета;
- получать результаты статического анализа расчета.

Приложение должно предоставлять удобный пользовательский интерфейс. Необходимо обеспечить возможность хранения и управления собственными проектами, а именно, изменение/удаление текущих параметров, включение других пользователей в группу допущенных к работе над данным проектом. Приложение должно обеспечивать работу на

всех доступных пользователю платформах. Одной из главных задач, решаемых в рамках данной работы, является организация работы приложения не зависимо от устройства пользователя. Система должна быть максимально проста и доступна.

В качестве основы построения системы выбран фреймворк для построения распределенных приложений и межпроцессного взаимодействия WCF (Windows Communication Foundation), который представляет платформу для построения сервисноориентированных приложений [4]. С помощью WCF можно отправлять данные в виде асинхронных сообщений от одной конечной точки службы к другой (рис. 1).

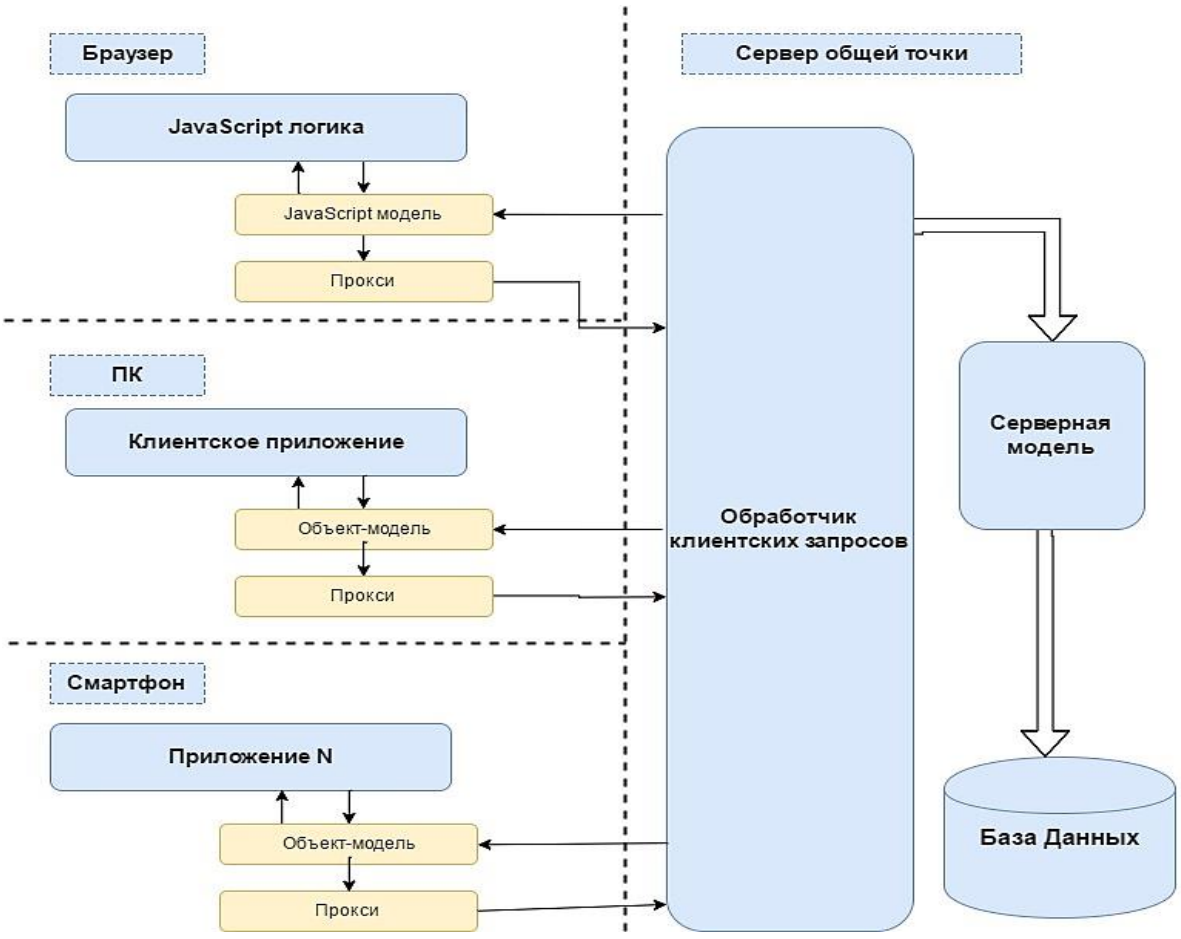

Рис. 1. Пример организации взаимодействия WCF и клиента [5]

Главной особенностью WCF является то, что это, прежде всего технология для построения сервис-ориентированной архитектуры приложений (SOA – Service-Oriented Architecture), что позволяет абстрагироваться от конкретной технологии [5]. Эта архитектура подразумевает применение веб-служб для отправки и получения данных. Общим преимуществом служб является слабая связанность вместо жесткой запрограммированности для различных приложений. Слабая связь означает, что любой клиент, созданный на любой платформе, может подключаться к любой службе при условии, что выполняются необходимые контракты.

В качестве главного преимущества, рассматриваемого фреймворка можно выделить высокую гибкость, которую получает разработчик после настройки сервиса. WCF упро-

щает изменение и использование программных компонентов. Разработчик получает возможность транслировать однажды написанные алгоритмы на различные платформы различными существующими путями, или переопределять свои. Пользователь, в то же время, имеет возможность использовать этот компонент в любом из сконфигурированных сценариев. Это преимущество позволяет сократить время на интеграцию изменений в программы, используемые на разных устройствах, повысить мобильность, и скорость внедрения приложения в новую область. Так же необходимо обратить внимание на возможность сослаться на проекты или методы, не являющиеся частью решения WCF. Таким образом, в состав сервиса можно включать уже существовавший код, содержащий классы данных, которые используются для обращения и взаимодействия с ресурсами службы данных как с объектами.

Как основной недостаток WCF, часто отмечают процесс обучения. Порой данный фреймворк кажется довольно сложным в освоении для многих разработчиков, когда они впервые начинают его изучение.

2. Принципы построения масштабируемых веб-систем

При проектировании крупномасштабных веб-систем учитывают следующие ключевые принципы [6].

- **Доступность.** Время в течение, которого, веб-ресурс способен поддерживать работоспособное состояние особенно важно по отношению к репутации утилитарности многих компаний.
- **Производительность.** Веб-производительность стала одним из наиболее значимых показателей для огромного количества веб-ресурсов. В связи с этим, ключевым моментом при проектировании веб-системы является оптимизация длительности ответов и задержек.
- **Надежность.** Система должна быть надежной, таким образом, чтобы определенный запрос на получение данных единообразно возвращал определенные данные и только правообладателю. В случае изменения данных или обновления, то тот же запрос должен возвращать данные с учетом новых изменений.
- **Масштабируемость.** Размер любой крупной распределенной системы – еще один значительный фактор, оказывающий влияние на проектные решения. Масштабируемость может относится к различным параметрам системы, и, как следствие, к различным функциональным узлам: максимальный траффик, который система способна выдержать, насколько легко проходит наращивание объемов памяти, максимальное число обрабатываемых транзакций.
- **Управляемость.** Проектирование системы, которая проста в эксплуатации, еще один аспект, требующий тщательного рассмотрения. Управляемость системы приравнивается к масштабируемости операций «обслуживание» и «обновления».

Каждый из вышеописанных аспектов является базовым принципом для принятия решений в проектировании распределенной веб-архитектуры. Однако они зачастую проти-

воречат друг другу, вследствие чего зачастую приходится пренебрегать одним принципом для достижений целей другого.

Как правило, разбиение системы на ряд атомарных сервисов позволяют изолировать работу одних частей системы от других. Подобная абстракция обеспечивает строгую иерархию отношений между службой, базовой средой службы и ее потребителями. Создание структурированной схемы помогает локализировать проблемы и позволяет производить масштабирование каждой отдельной части системы независимо друг от друга. Этот вид сервисно-ориентированного устройства систем для обслуживания широкого круга запросов аналогичен объектно-ориентированному подходу в программировании.

Главное преимущество такого подхода заключается в возможности решать проблемы и вносить изменения независимо друг от друга. При этом производительность каждой из служб может быть оптимизирована в рамках своего функционала. Как с точки зрения обслуживания, так и стоимости каждая служба может быть масштабирована независимо по мере необходимости. Это является положительным фактором, поскольку их объединение и смешивание могло бы непреднамеренно влиять на их производительность.

Существуют две основные сложности, возникающие с ростом приложения: масштабирование доступа к базе данных и к серверу приложений. У систем с хорошей масштабируемостью сервер приложений или веб-сервер, как правило, минимизируется и зачастую реализуют архитектуру, которая не поддерживает совместного разделения ресурсов. Такой подход предусматривает горизонтальную масштабируемость уровня сервера приложений. В итоге, при использовании такого дизайна, тяжёлый труд будет связан непосредственно с сервером базы данных и вспомогательными службами. Именно на этом слое появляются и преобладают настоящие проблемы масштабирования и производительности.

3. Выбор архитектуры клиент-сервер

Клиент-серверные приложения чаще всего построены на основе двухзвенной архитектуры [7]. Двухзвенной она называется из-за необходимости распределения *трех* базовых *компонентов* между *двумя узлами* (клиентом и сервером).

Двухзвенная архитектура используется в клиент-серверных системах, где сервер отвечает на клиентские запросы напрямую и в полном объеме, при этом используя только собственные ресурсы. То есть сервер не вызывает сторонние сетевые приложения и не обращается к сторонним ресурсам для выполнения какой-либо части запроса.

Со временем развитие распределенной архитектуры информационных систем с базами данных привело к появлению трехзвенной модели, особенностью которой стал третий узел – сервер приложений [7]. Это -самостоятельный узел, находящийся между клиентскими подсистемами и сервером базы данных. Такая архитектура связана с использованием распределенных вычислений, которые реализуются путем разделения сетевого приложения на две или более частей, каждая из которых может выполняться на отдельной машине. Выделенные части приложения взаимодействуют друг с другом по заранее установленному протоколу (рис. 2).

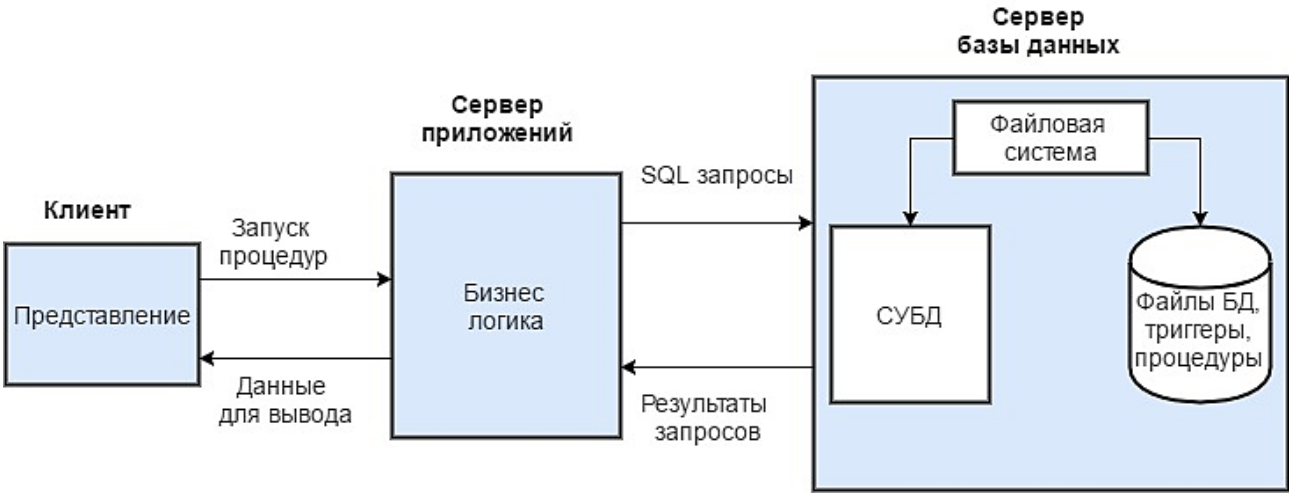

Рис. 2. Пример организации трехзвенной архитектуры [7]

Для трёхзвенной архитектуры типично разделение на узлы следующим образом:

- представление данных на стороне клиента;
- прикладной компонент на сервере приложений, выполняющий функции промежуточного ПО;
- управляющий ресурсами узел на сервере базы данных.

Подобная архитектура не ограничена наличием лишь трех узлов, она может быть расширена до многозвенной путем добавления дополнительных серверов, каждый из которых обеспечивает собственные сервисы и использует, предоставляемые услуги серверов разного уровня.

Таким образом, основная идея архитектуры «клиент-сервер» заключается в разделении сетевого приложения на несколько компонентов, каждый из которых выполняет специально отведенный набор сервисов [10]. Компоненты такого приложения могут выполнятся на разных серверах, выполняя серверные или клиентские функции. Использование клиент-серверной архитектуры позволяет повысить надежность, безопасность и производительность сетевых приложений [11].

4. Проектирование сервера базы данных

Проектирование базы данных (БД) предполагает интеграцию данных, предназначенных для решения нескольких прикладных задач разных пользователей [8]. Соответственно, при интеграции данных должны учитываться требования к данным каждого пользователя, основанные на его представлении о данных и связях между ними.

Обобщение представлений всех пользователей о данных называется концептуальной моделью (схемой) БД. Концептуальная модель представляет информационное описание предметной области с учетом логических взаимосвязей, поэтому её еще называют инфологической (информационно-логической) моделью. В модели отсутствуют какие-либо понятия, связанные с компьютером, компьютерной памятью, способами размещения данных в памяти, и, по сути, это - модель только предметной области.

Следующий этап разработки базы данных предполагает выбор представления концептуальной модели с помощью модели данных конкретной системы управления базами данных (СУБД). Полученное таким образом представление концептуальной модели называется логической моделью БД. Или, другими словами, логическая модель – это концептуальная схема, специфицированная в языке конкретной СУБД. Логическая модель представляет данные и элементы данных вне зависимости от их содержания и среды хранения. Далее разработчик системы средствами СУБД отображает полученную логическую модель БД в память компьютера и определяет методы доступа.

В проектируемой системе необходимо сохранять множество данных: информацию о пользователях, результаты расчетов и анализов. Для этого на нашем сервере понадобиться СУБД. В архитектуре «клиент-сервер» работают так называемые «промышленные» СУБД. Свое название они получили в виду того, что именно СУБД этого класса могут обеспечить работу информационных систем масштаба среднего и крупного предприятия. К разряду промышленных СУБД относятся такие системы как MS SQL Server, Oracle, MongoDB, Hwaci, MariaDB, Percona, SAP AG, и другие. В представляемой реализации для хранения и управления клиентскими и серверными данными использовалась СУБД Microsoft SQL Server.

5. Сервер обработки расчетов

Сервер обработки расчетов выделен как отдельный функциональный элемент архитектуры приложения. Он предназначен для обработки задач по расчету и анализу Парето-аппроксимаций. Сами расчеты производятся программами, которые могут быть написаны на различных языках программирования. Сервер обработки расчетов должен осуществлять взаимодействие с исполняемыми файлами расчетных модулей и файлами с данными.

Расчеты могут выполняться длительное время и создавать серьёзную нагрузку на сервер, а это значит, что производить расчеты на том же сервере, где работает MVC-приложение, нельзя. Взаимодействие сервера обработки расчетов и MVC- приложения будет происходить при помощи API.

Обработка расчетов происходит следующим образом: MVC-приложение посылает запрос, в котором передается тип расчета и данные. Затем сервер обработки проверяет необходимые параметры, и, если проверка проходит успешно, сервер расчетов отдает ответ HTTP 200 и отложено выполняет расчет. Когда расчет завешен, сервер отдает результаты в MVC-приложение. При такой схеме работы клиент не ждет окончания расчетов, так как они могут занимать много времени.

Сервер обработки расчетов должен состоять из веб-сервера, модуля работы с API, модуля работы с файлами. Такая структура позволит обеспечить гибкость взаимодействия с программами расчета и анализа. Расчетный сервер является независимым элементом. Это важное условие позволит легко решить задачу горизонтального масштабирования. При высокой загрузки одного сервера расчетов необходимо установить дополнительные копии приложения.

6. Клиентская часть приложения

Традиционным клиентом веб-приложения является браузер. Основное предназначение браузера – отображать веб-ресурсы. Для этого на сервер отправляется запрос, а результат выводится в окне браузера. Под ресурсами в основном подразумеваются *HTML*-документы, однако это также может быть *PDF*-файл, картинка или иное содержание. Расположение ресурса определяется с помощью *URI* (унифицированного идентификатора ресурсов).

Многие годы браузеры отвечали лишь части спецификаций, и для них создавались отдельные расширения. Для веб-разработчиков это означало серьезные проблемы с совместимостью. Сегодня большинство браузеров в большей или меньшей степени отвечает всем спецификациям.

За внешний вид страницы отвечают каскадные таблицы стилей (CSS) – формальный язык описания внешнего вида документа, написанного с использованием языка разметки [9]. CSS используется создателями веб-страниц для задания цветов, шрифтов, расположения отдельных блоков и других аспектов представления внешнего вида этих веб-страниц. Основной целью разработки CSS являлось разделение описания логической структуры веб-страницы, которое производится с помощью HTML или других языков разметки, от описания внешнего вида этой веб-страницы, которое теперь производится с помощью формального языка CSS. Такое разделение может увеличить доступность документа, предоставить большую гибкость и возможность управления его представлением, а также уменьшить сложность и повторяемость в структурном содержимом. Кроме того, CSS позволяет представлять один и тот же документ в различных стилях или методах вывода, таких как экранное представление, печатное представление.

Вычисления, которые необходимо выполнить на стороне клиента, выполняются при помощи JavaScript. Этот язык обычно используется как встраиваемый для программного доступа к объектам приложений. Наиболее широкое применение этот язык находит в браузерах, как язык сценариев для придания интерактивности веб-страницам [12].

Клиентский JavaScript расширяет ядро языка, добавляя объекты управления браузером и Document Object Model (DOM). Например, клиентские расширения позволяют приложению размещать элементы на HTML-форме и реагировать на пользовательские события, таких как щелчок мышью, ввод данных в форму или навигация по страницам.

JavaScript позволяет делать AJAX запросы. AJAX-подход к построению интерактивных пользовательских интерфейсов веб-приложений заключается в фоновом обмене данными браузера с веб-сервером. В результате при обновлении данных веб-страница не перезагружается полностью. Следует отметить, что AJAX — не самостоятельная технология, а концепция использования нескольких смежных технологий, использующих динамическое обращение к серверу «на лету», без перезагрузки всей страницы полностью.

7. Реализация клиентской части веб-приложения

Клиентом реализуемого приложения является браузер. На сегодняшний день различными компаниями создано большое число браузеров. В зависимости от браузера должны применятся различные приемы для корректного отображения информации, поэтому для того что бы упростить разработку интерфейса и исключить вероятность неправильного отображения информации были использованы различные фреймворки.

Одним из таких фреймворков является Bootstrap. Bootstrap – это свободный набор инструментов для создания сайтов и веб-приложений. Он включает в себя HTML и CSS – шаблоны оформления типографики и различных элементов веб-верстки. Файлы этого фреймворка по умолчанию включены в проект. Использование Bootstrap позволяет легко и быстро реализовать адаптивную верстку - подход, предполагающий изменение дизайна в зависимости от поведения пользователя, размера экрана, платформы и ориентации девайса. Это увеличит гибкость и удобство приложения.

Другим инструментом для удобной реализации визуальной составляющей проекта является JavaScript-библиотека jQuery. Использование этой библиотеки значительно сокращает время создания различных визуальных эффектов и упрощает способ взаимодействия с элементами DOM. Также данная библиотека использовалась для организации работы с AJAX.

Значительное удобство использования jQuery становится заметно на стадии выполнения расчетов. Расчеты могут выполняться длительное время, и получить результаты сразу после создания невозможно. Благодаря фоновым запросам, пользователю не придется ждать, каждый раз перезагружая страницу. Средствами jQuery результат будет выведен, как только расчетный сервер пришлет ответ. Сразу после создания пользователь увидит сообщение о том, что расчет выполняется. После завершения выполнения запроса, выведется сообщение об успешном окончании, и автоматически загрузятся результаты расчетов. Все это будет выполнено без перезагрузки страницы, и соответственно без участия пользователя.

Помимо этого, с помощью данной библиотеки реализована обработка различных событий, таких как: добавление, изменение, удаление информации. В виду того что зачастую результаты расчетов довольно громоздкие, реализована возможность сворачивать и разворачивать часть данных выводимых на экран, таким образом, чтобы необходимая информация умещалась на видимой области экрана (рис. 3).

Рис. 3. Параметры задачи свернуты для удобства

8. Реализация серверной части веб-приложения

Серверная часть приложения представляет собой совокупность двух компонентов: MVC-приложение и сервер, выполняющий расчеты. MVC-приложение обрабатывает пользовательские запросы, обращается к СУБД, формирует и возвращает ответы пользователю.

В разрабатываемом приложении не столь важно максимизировать производительность MVC-приложения, как, например, сервера расчетов. Однако важно обеспечить надежность и большое число одновременно обрабатываемых запросов. В качестве платформы для реализации MVC-приложения была выбрана ASP.NET платформа, организации работы с которой осуществляется посредством языка программирования C#. Платформа ASP.NET расширяема и дополняема. Компоненты платформы ASP.NET MVC можно легко заменить или настроить. Разработчик может подключать собственный механизм представлений, политику маршрутизации URL-адресов, сериализацию параметров методов действий и другие компоненты.

В разрабатываемом приложении есть несколько основных модулей: Controllers, Views, Models и Web.config. В модуле Controllers определяется, какой контролер будет обрабатывать определённый запрос, идентифицируемый по URL с помощь регулярного выражения. Модуль Views содержит все шаблоны страниц, необходимые для генерации представле-

ния. Внутри контролеров реализуется логика предметной области, с которой мы работаем. Сущности, которыми оперирует приложение, представлены в модуле Models. Они описываются с помощью классов с соответствующими полями и методами (рис. 4). Настройки, необходимые приложению, хранятся в виде XML-файла в модуле Web.config.

Когда поступает запрос от пользователя, по URL вызывается нужный контролер. Контролер проверяет данные, выполняет предписанные ему манипуляции с данными и выполняет генерацию страницы с помощью полученных данных и шаблона представления.

```
ссылок: 7
public class CalculationInfoModel
{
    ссылок: 15 | 0 исключения
    public Calculation calc { get; set; }
    ссылок: 7 | 0 исключения
    public PISAcfg cfg { get; set; }
    ссылок: 5 | 0 исключения
    public Dictionary<int,string> alg_names { get; set; }
    ссылок: 5 | 0 исключения
    public Dictionary<int,string> prb_names { get; set; }
    ссылок: 14 | 0 исключения
    public Problem_Param prb_param { get; set; }
    ссылок: 8 | 0 исключения
    public Algorithm_Param alg_param { get; set; }
    ссылок: 5 | 0 исключения
    public List<string> result_strings { get; set; }
    ссылок: 5 | 0 исключения
    public List<string> front_strings { get; set; }
    ссылок: 5 | 0 исключения
    public List<Helper> helps { get; set; }

    ссылка: 1 | 0 исключения
    public CalculationInfoModel()...
    ссылка: 1 | 0 исключения
    public CalculationInfoModel(int id)...

    ссылка: 1 | 0 исключения
    public List<string> getResult()...
    ссылка: 1 | 0 исключения
    public List<string> getFront()...


}
```

Рис. 5. Пример объекта Models

Приложение не работает с чистыми SQL-запросами, так как такой подход существенно ухудшает защищенность данных. Для взаимодействия с базой данных используется Entity Framework, что позволяет абстрагироваться от самой базы данных и работать с данными независимо от типа хранилища. Схема базы данных генерируется по классам, представленным в модуле Models. Такой подход не привязывает нас к конкретной СУБД и позволяет при необходимости ее сменить.

9. Реализация сервера расчетов

Основной задачей системы является проведение и анализ различных расчетов Парето-аппроксимаций. Выполнение этой задачи на стороне MVC-приложения нецелесообразно, поэтому было решено предоставить процесс выполнения расчетов целиком отдельному серверу. Так как с ростом числа пользователей нагрузка на узел будет увеличиваться, сервер расчетов должен легко масштабироваться.

Сервер обработки расчетов представляет собой веб-сервер IIS, который должен выполнять скрипты с алгоритмами расчетов, оценок и анализа. IIS позволяет добиваться улучшенной производительности благодаря своей модульной архитектуре, настраиваемости и возможности создания функций улучшения производительности.

В MVC-приложении пользователь создает «анализ», «расчет» или «оценку», вводя необходимые данные в форму создания (рис. 5). Затем следует запрос к серверу обработки расчетов, где проверяются входные данные. Если все данные введены верно, то сервер обработки расчетов ставит задачу на расчет в очередь, и отдает ответ об успешные постановки в очередь. В случае если данные не прошли проверку, возвращается ответ с указанием какой из параметров ошибочный.

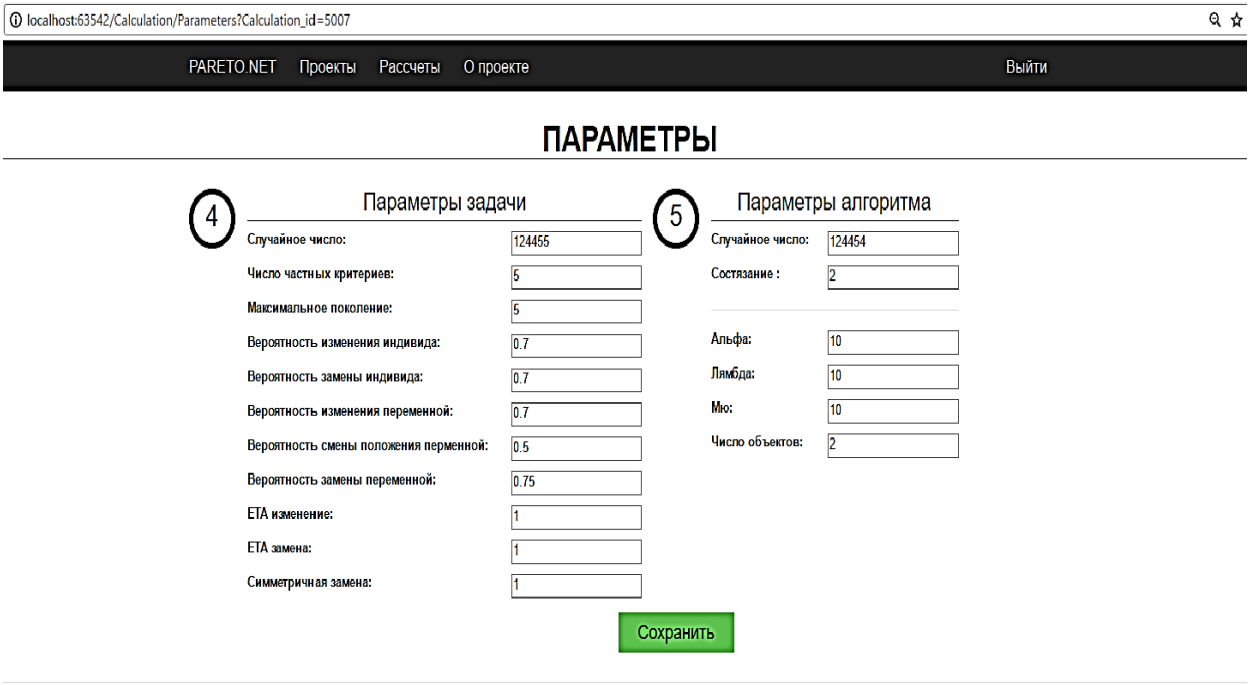

Рис. 5. Указание параметров при создании расчета

Обмен данными между MVC-приложением и сервером расчетов реализован посредством обмена JSON сообщениями по протоколу HTTP. Тело сообщения всегда содержит информацию о статусе операции, типе сообщения, идентификатора выполняемой операции. Также тело сообщения несет в себе дополнительные данные, в зависимости от типа операции: фронты Парето, индикаторы и результаты анализа.

10. Аутентификация и регистрация пользователя

Для того что бы хранить результаты расчетов и работы с ними, пользователь должен быть зарегистрирован. При попытке доступа к системе пользователю будет предложено авторизоваться либо выполнить процесс регистрации, после чего он сможет получить доступ к личному рабочему пространству.

Для регистрации необходимо ввести следующие данные:

- имя пользователя;
- e-mail пользователя;
- пароль;
- подтверждение пароля.

Система регистрации обрабатывает следующие ошибки:

- попытка введения имени, уже существующего в базе данных пользователей;
- отсутствие информации в полях формы;
- идентичность введенных паролей.

Механизм авторизации реализован с помощью фильтров. Фильтр действий обычно является атрибутом, который реализует абстрактный класс FilterAttribute. Некоторые фильтры действий реализуют класс FilterAttribute напрямую и вызываются до выполнения метода действия. Это свойство используется для проверки наличия прав доступа к определенной части приложения. Такой подход очень удобен, поскольку позволяет компактно описывать ограничения доступа указанием фильтра над методом. При попытке доступа к контроллеру расчетов, система предварительно выполнит проверку аутентификации пользователя. В случае удачного прохождения проверки, пользователь получит доступ к запрашиваемому действию в противном случае он будет перенаправлен на страницу аутентификации.

Данные, введенные пользователем, сверяются с имеющимися в базе данных. При этом используется схема взаимодействия с базой данных с доступом к ограниченному числу таблиц. При успешной аутентификации приложение установит сессионную cookie, пользователь сможет продолжить работу с сервисом, и ему становится доступно рабочее пространство, представляющее собой список поименованных проектов, содержащих расчеты, анализы или оценки. Пользователь может создавать новые проекты, изменять или удалять существующие (рис. 6).

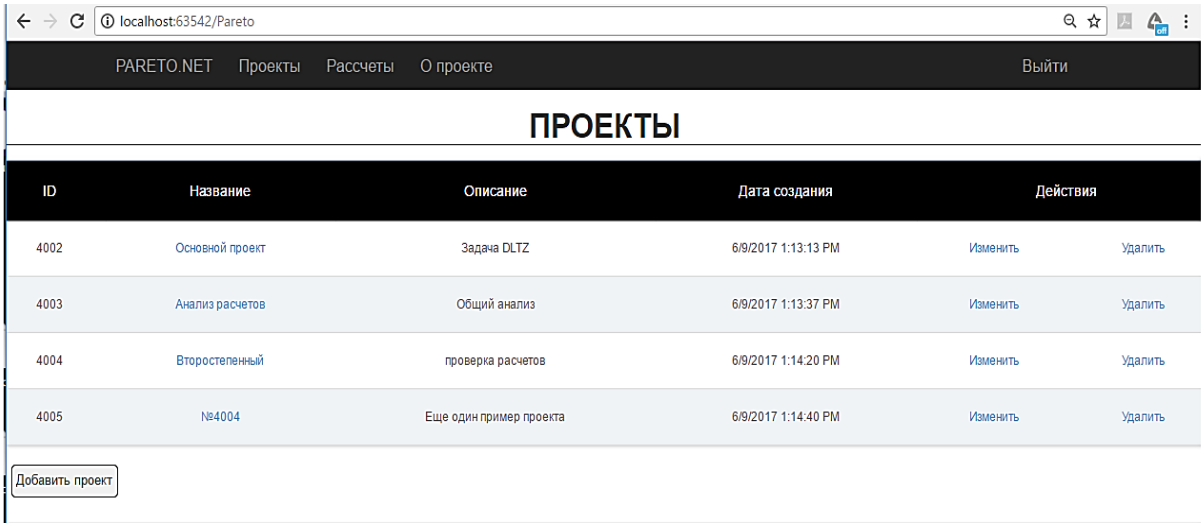

Рис. 6. Список пользовательских проектов

Как только проект был выбран или создан вновь, пользователю станет доступно создание или просмотре «расчетов», «оценок» и «анализа». Все элементы проекта разбиты по таблицам, содержащим краткую информацию, для удобного просмотра.

Чтобы создать новый «расчет», «анализ» или «оценку», необходимо кликнуть на кнопку «Создать расчет», «Создать анализ» или «Создать оценку» или в области проекта. При создании расчета, пользователь должен указать задачу и алгоритм решения, после чего заполнить все необходимые числовые поля параметров (рис. 7).

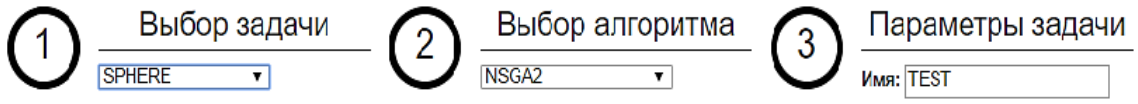

Рис. 7. Выбор алгоритма и задачи

Заключение

В представленной работе разработана архитектура масштабируемого веб-сервиса, построенного на основе WCF. Описаны основные функциональные узлы данного фреймворка и принцип их работы. Продемонстрированы основные методики, используемые при построении высоконагруженных проектов. Указаны способы, позволяющие оптимизировать работу основных частей системы. Детально разобраны способы взаимодействия клиента и сервера. Описана схема работы веб-серверов с асинхронной обработкой запросов. Выделены основные компоненты системы для дальнейшей разработки.

Представлено описание разработанной веб-системы. Приведен обзор программного обеспечения, используемого при создании сервиса. Рассматривается работа каждого из компонентов системы, а также методов их взаимодействия. Показаны основные элементы интерфейса приложения. Приведен обзор сценариев работы пользователя с проектом.

Работа поддержана РФФИ (проект 16-07-00287).

Список литературы

[1]. Карпенко А.П., Митина Е.В., Семенихин А.С. Популяционные методы аппроксимации множества Парето в задаче многокритериальной оптимизации // Наука и образование: электронное научно - техническое издание. 2012. №4. Режим доступа: http://www.technomag.edu.ru/doc/363023.html (дата обращения: 30.09.2017)

[2]. Грошев С.В., Карпенко А.П., Сабитов Д.Р., Шибитов И.А. Программная система PARETO-RATING для оценки качества Парето-аппроксимации в задаче многокритериальной оптимизации. // Наука и образование: научное издание МГТУ им. Н.Э. Баумана. 2014; (7): 193-214. DOI: 10.7463/0714.0720253. Режим доступа: http://technomag.bmstu.ru/doc/720253.html (дата обращения: 30.09.2017)

[3]. Белоус В.В., Грошев С.В., Карпенко А.П., Шибитов И.А. Оценка качества Парето-аппроксимации в задаче многокритериальной оптимизации. Обзор программных систем. Наука и образование: электронное научно- техническое издание, 2014, № 4, С. 300-320. DOI: 10.7463/0414.0709198. Режим доступа: http://technomag.bmstu.ru/doc/709198.html (дата обращения: 30.09.2017)

[4]. Сибраро П. и др. WCF 4: Windows Communication Foundation и NET 4 для профессионалов= Professional WCF 4: Windows Communication Foundation with. NET 4. М.: ООО И.Д. Вильямс. 2011. 465 с.

[5]. Биберштейн Н., Боуз С. Компас в мире сервис-ориентированной архитектуры (SOA). М.: КУДИЦ-Пресс. 2007. 256 с.

[6]. Avison D., Fitzgerald G. Information systems development: methodologies, techniques and tools. – McGraw Hill. 2003.

[7]. Коржов В.А. Многоуровневые системы клиент-сервер. // Сети/Network world. М.: Изд-во Открытые системы. 1997. С. 32-54.

[8]. Коннолли Т., Бегг К. Базы данных. Проектирование, реализация и сопровождение. Теория и практика. 3-е изд.: Пер. с англ. М.: Изд. дом "Вильямс". 2003. 1440 с. С. 520-582.

[9]. Мейер Э.А. CSS-каскадные таблицы стилей: подробное руководство. 3-е изд. Пер. с англ. СПб.: Символ-Плюс. 2008. 576 с.

[10]. Семченко П. Н. Концепция клиент-серверной среды динамических экспертных систем // Интеллектуальные системы. 2010. №. 3. С. 25.

[11]. Таненбаум Э., Стеен М. Ван. Распределенные системы. Принципы и парадигмы. СПб.: Питер. 2003. 877 с.

[12]. Флэнаган Д. JavaScript. Подробное руководство. 5-е изд. Пер. с англ. СПб.: Символ-Плюс. 2008. 992 с.