ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ №3(30), 2016, с. 3–23

УДК 004.457+004.031.2+004.382.2

А. В. Баранов, А. А. Зонов

Вариант организации облачного сервиса для высокопроизводительных вычислений

Аннотация. Статья посвящена вопросу создания облачного сервиса вида SaaS для готовых суперкомпьютерных приложений. В статье рассмотрены технологии организации подобного облачного сервиса на основе программного комплекса «Пирамида» и Системы управления прохождением параллельных заданий (СУППЗ).

Созданный облачный сервис призван выполнять роль дополнительного уровня абстракции, позволяющего объединить разные высокопроизводительные вычислительные установки и организовать для них единый интерфейс управления.

Ключевые слова и фразы: высокопроизводительные вычисления, облачные технологии, разработка облачных сервисов, Ruby on Rails, Angularjs, архитектура RESTful, SaaS.

Введение

С появлением в конце 90-х годов 20-го века кластерной архитектуры начался и продолжается непрерывный рост числа пользователей супер-ЭВМ и суперкомпьютерных приложений. При этом за прошедшие годы представление кластерной вычислительной системы с точки зрения пользователя практически не изменилось. Для работы на кластерной системе пользователю необходимо:

- зарегистрироваться в системе и соединиться с сервером доступа вычислительной установки;
- подготовить параллельную программу и данные для расчётов;
- оформить для запуска так называемое *параллельное задание* (совокупность параллельной программы, входных данных и требований к вычислительным ресурсам) и направить его в очередь системы пакетной обработки заданий (СПО);
- дождаться прохождения задания через очередь, его старта и окончания, после чего получить доступ к результатам расчётов.

© А. В. Баранов, А. А. Зонов, 2016
© Межведомственный суперкомпьютерный центр РАН, 2016
© Программные системы: теория и приложения, 2016

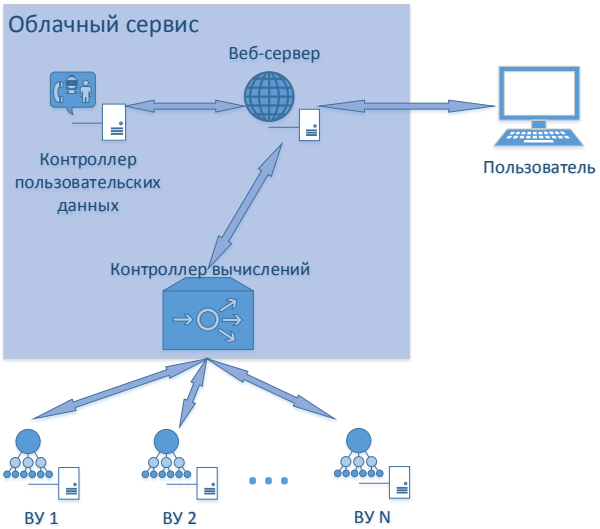

Рис. 1. Общая структура облачного сервиса

В распоряжении пользователя может быть несколько разных территориально распределённых вычислительных установок с отличающимися друг от друга архитектурами. Для каждой из них пользователь вынужден повторять обозначенный порядок работы, что далеко не всегда является удобным и эффективным. При этом очень часто пользователи не разрабатывают новые параллельные приложения, а используют для расчётов готовые продукты, такие как ANSYS Fluent, ESI CFD-Fastran и т.п. В случае использования готовых программных решений необходимость дублирования на каждой вычислительной установке подготовительных действий для производства расчётов становится крайне неудобной и нежелательной для пользователя.

1. Общая структура разрабатываемого облачного сервиса и требования к нему

Целью представляемой авторами работы является выбор средств и решений для быстрого построения и разработка облачного сервиса вида SaaS для готовых суперкомпьютерных приложений. Облачный сервис должен выполнять роль дополнительного уровня абстракции, позволяющего объединить вычислительные установки и организовать

для них единый интерфейс управления. Общая структура подобного сервиса приведена на рис. 1.

Пользователь через веб-браузер соединяется с облачным сервисом и с помощью *контроллера пользовательских данных* оформляет параллельное задание. Управление заданиями осуществляет *контроллер вычислений*, который непрозрачным образом размещает задание пользователя на одной из доступных вычислительных установок (ВУ). Каждая из вычислительных установок находится под управлением локальной системы пакетной обработки (СПО), в качестве которой могут выступать такие системы, как SLURM, PBS, Moab и т.п.

Локальная СПО обеспечивает следующие функции:

- приём входного потока заданий от разных пользователей;
- ведение локальной очереди заданий;
- выделение ресурсов вычислительной установки для прошедшего очередь задания, их конфигурирование;
- разворачивание задания на выделенных ресурсах, производство расчётов;
- освобождение выделенных ресурсов по окончании задания;
- предоставление пользователю результатов расчётов.

Облачный сервис автоматизирует для пользователя следующие процессы:

- подготовки параллельного задания;
- выбора подходящей вычислительной установки для размещения задания;
- взаимодействия с локальной СПО для организации выполнения задания.

Поток заданий, поступающих на выполнение от пользователей облачного сервиса, назовём *облачным* потоком заданий. Одновременно и совместно с облачным потоком заданий должен обрабатываться *стандартный* поток заданий, поступающих на выполнение обычным (стандартным) способом, т.е. непосредственно через СПО. *Обеспечение совместной обработки стандартного и облачного потоков является главным требованием к создаваемому облачному сервису.*

Другие требования к облачному сервису могут различаться в зависимости от того, в какой роли выступает его клиент — пользователя или администратора.

Для роли «пользователь» создаваемый облачный сервис должен предоставлять следующие возможности:

- для начала работы с облачным сервисом пользователь должен иметь возможность зарегистрироваться, а в начале каждого нового сеанса работы — авторизоваться;
- пользователь должен иметь возможность формирования своих заданий, а также формирования, редактирования и хранения в системе шаблонов типовых заданий;
- пользователю должна предоставляться информация о доступных вычислительных ресурсах;
- пользователь должен иметь возможность направлять сформированные задания на выполнение, отслеживать состояния и управлять своими заданиями;
- пользователю должен быть предоставлен интерфейс управления своей учётной записью.

Для роли «администратор» должны предоставляться следующие возможности и интерфейсы:

- управления учётными записями пользователей;
- подключения к облачному сервису и отключения от него вычислительных установок.

К требованиям системного уровня следует отнести:

- доступ к облачному сервису должен осуществляться посредством веб-интерфейса;
- должна быть обеспечена изоляция пользователей друг от друга — пользователь должен иметь доступ только к своей учётной записи и только к своим заданиям;
- облачный сервис должен контролировать соединения с СПО и производить запуск заданий по доступным соединениям;
- взаимодействие облачного сервиса с вычислительными установками должно осуществляться через защищённое (в нашем случае — через SSH) соединение.

В качестве примера приложения для организации облачного сервиса авторами был взят программный комплекс организации параллельных вычислений с распараллеливанием по данным «Пирамида» [1], хотя рассмотренный в статье подход может быть использован и для других приложений. В качестве локальной СПО вычислительных установок выступила отечественная система управления прохождением параллельных заданий (СУППЗ) [2].

2. Выбор технологий и инструментальных средств разработки облачного сервиса

2.1. Существующие комплексные решения для построения облачных сервисов

На сегодняшний день на рынке представлено несколько программных продуктов для построения инфраструктуры частного, публичного или гибридного облака, среди которых можно отметить следующие:

- VMware vCloud Director — проприетарная программная платформа для построения частных и гибридных облаков с моделью предоставления облачных вычислений IaaS;
- HPE Helion Eucalyptus — платформа для построения частных облаков[1];
- Apache Cloudstack — программная платформа автоматизации развёртывания, настройки и обеспечения функционирования частных, публичных и гибридных облаков[2];
- OpenStack — платформа для построения облаков с моделью предоставления облачных вычислений IaaS[3].

Несмотря на различия приведённых программных платформ, все они используют общие принципы предоставления вычислительных ресурсов пользователям. Пользователь при помощи клиентских инструментов инициирует запрос на создание/использование виртуальной машины. После аутентификации и авторизации пользователя запрос перенаправляется в специальный компонент, присутствующий во всех платформах, — *контроллер облака*. С помощью контроллера облака в сетевом хранилище производится поиск образа необходимой виртуальной машины (машин) и подбираются наиболее подходящие вычислительные ресурсы, на которых начинается выполнение виртуальных машин. Пользователь подключается к запущенной виртуальной машине и работает с ней, а всю деятельность по поддержанию виртуальной машины в работоспособном состоянии, выделении дискового пространства, маршрутизации сетевых пакетов и прочие действия по организации вычислительного процесса платформа берёт на себя.

---

[1] имеются как бесплатная (совместимая только с гипервизорами Xen и KVM), так и платная (дополнительно совместимая с гипервизором VMware ESXi) версии

[2] распространяется под лицензией GPL

[3] распространяется в открытых исходных текстах

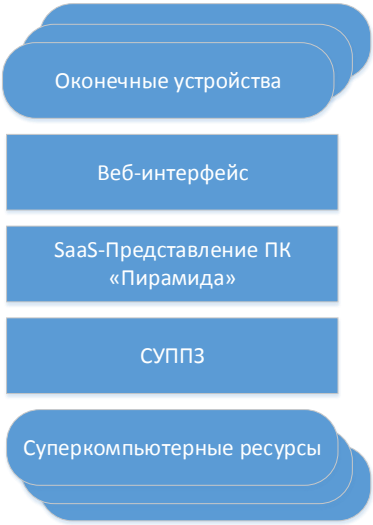

Рис. 2. Архитектура облачного сервиса

Заметим, что при подборе вычислительных ресурсов и запуске на них виртуальных машин контроллер облака взаимодействует непосредственно с гипервизорами физических вычислительных модулей (серверов), анализируя текущую загруженность решающего поля и балансируя вычислительную нагрузку. Другими словами, решающее поле должно быть передано под полное управление облачной платформы, что делает невозможным использование СПО и сочетания двух потоков заданий — облачного и стандартного. Поскольку это противоречит главному требованию к создаваемому облачному сервису, авторы приняли решение отказаться от использования готовой облачной платформы и провести собственную разработку.

2.2. Архитектура и структура облачного сервиса

Место разрабатываемого облачного сервиса в цепочке предоставления параллельного приложения (на примере ПК «Пирамида») как сервиса демонстрирует рис. 2.

Пользователь облачного сервиса через веб-интерфейс получает доступ к представлению SaaS. Представление SaaS обеспечивает хранение и обработку пользовательских данных, а также предоставляет интерфейс СУППЗ.

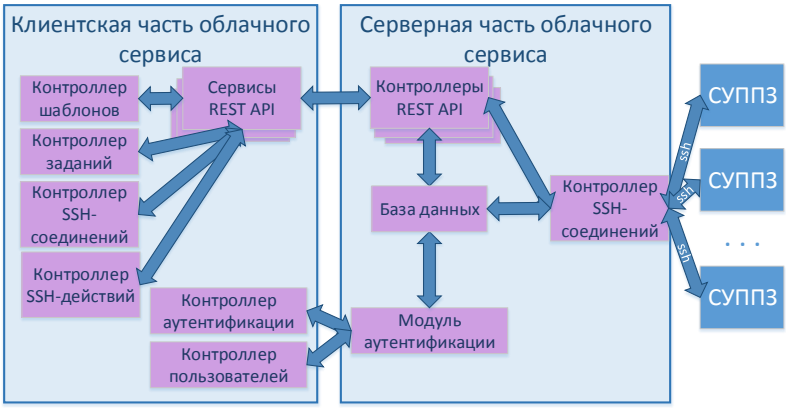

Рис. 3. Структура облачного сервиса

Разработка облачного сервиса подразумевает работу над тремя его составляющими:

- клиентская часть;
- серверная часть;
- интерфейс взаимодействия клиентской и серверной частей.

На рис. 3 обозначены основные структурные элементы облачного сервиса.

Взаимодействие клиентской и серверной частей сервиса было выстроено в соответствии с архитектурой RESTful. Преимущества такого подхода были обоснованы в [3], и в настоящее время применение архитектуры RESTful и программного интерфейса REST API де-факто является стандартом веб-программирования. REST API подразумевает унифицированный доступ к данным с некоторым ограниченным набором действий над ними. Данные, которыми обмениваются клиент и сервер, организуются в единый формат JSON.

Взаимодействие с пользователем в клиентской части облачного сервиса реализуется с помощью набора *контроллеров*, состав которого определяется, с одной стороны, требуемыми функциональными возможностями облачного сервиса, с другой стороны — техническими требованиями безопасности и обеспечения взаимодействия с СУППЗ. В набор контроллеров входят:

- контроллер аутентификации, необходимый для обеспечения авторизованного доступа к облачному сервису;

- контроллер пользователей, необходимый для регистрации и управления пользователями облачного сервиса;
- контроллер шаблонов, предназначенный для создания, хранения и изменения пользователями шаблонов своих заданий;
- контроллер заданий, необходимый для управления пользовательскими заданиями;
- контроллеры SSH-соединений и SSH-действий, необходимые для организации связи облачного сервиса с вычислительными установками под управлением СУППЗ.

Сервисы REST API клиентской части сервиса преобразуют запросы контроллеров в формат JSON и производят обмен данными с сервером по протоколу HTTP, для чего в серверной части должен функционировать веб-сервер.

В составе серверной части выделяются модуль аутентификации, осуществляющий управление пользователями, и контроллер SSH-соединений для управления соединениями с СУППЗ. Информация о текущей конфигурации облачного сервиса, о пользователях и доступных ВУ под управлением СУППЗ сохраняется в специальной базе данных (БД) облачного сервиса. Контроллеры REST API серверной части в ответ на запрос пользователя, в зависимости от характера поступившего запроса, либо производят обращение к БД, либо вызывают соответствующее действие контроллера SSH-соединений. Ответ на запрос преобразуется котроллером REST API в формат JSON и отправляется клиенту.

2.3. Выбор решений для построения облачного сервиса

Для построения облачного сервиса в соответствии со структурой, представленной на рис. 3, необходимо выбрать следующие решения:

- веб-сервер;
- каркас серверного веб-приложения (Web Application Framework, WAF);
- систему управления базой данных (СУБД);
- средство поддержки SSH-соединений в веб-приложении;
- шаблон проектирования клиентской части.

Поскольку разработка облачного сервиса ведётся в исследовательских целях, для всех выбираемых решений были выдвинуты следующие обязательные требования:

- решение должно обеспечивать высокую скорость разработки;

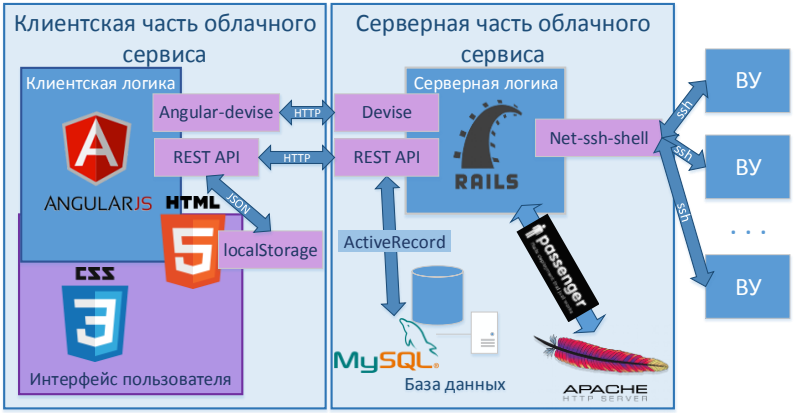

Рис. 4. Выбранные решения для построения облачного сервиса

- решение должно быть свободно распространяемым и функционировать в среде Linux;
- решение должно поддерживать схему «модель-представление-контроллер» (Model-View-Controller, MVC) [4], согласно которой модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделяются на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные;
- решение должно быть широко распространённым и иметь хорошую документальную и инструментальную поддержку;
- решение должно, с одной стороны, иметь многолетнюю апробацию большим числом пользователей и разработчиков, с другой стороны — быть передовым и соответствовать современному уровню развития технологий веб-разработки.

Кроме этого, выбираемые решения должны поддерживать совместную работу друг с другом.

Схема взаимодействия выбранных авторами решений для построения облачного сервиса представлена на рис. 4.

В качестве веб-сервера и СУБД авторами были выбраны Apache и MySQL соответственно, как полностью соответствующие приведённым требованиям.

Каркас серверного веб-приложения, помимо соответствия перечисленным общим требованиям, должен:

- соответствовать архитектуре RESTful;
- содержать как можно больше готовых решений для реализации контроллеров облачного сервиса, представленных на рис. 3.

Анализ интернет-публикаций и мнений экспертов-разработчиков показал, что выдвинутым требованиям в полной мере удовлетворяют два распространённых решения — Django [5] и Ruby on Rails (RoR) [6]. Отмечается [7], что при одинаково широкой распространённости обоих решений каркас Ruby on Rails более подходит для осуществления быстрой разработки, носящей исследовательский характер, а также содержит больше решений, готовых к использованию в новых проектах. По этой причине логика серверной части облачного сервиса была реализована с использованием каркаса RoR. Взаимодействие RoR-приложения с вебсервером Apache было обеспечено за счёт применения Ruby-библиотеки Passenger.

Модуль механизма аутентификации основан на Ruby-библиотеке Devise. Библиотека позволяет разрабатывать собственные системы аутентификации, используя современные алгоритмы авторизации и взаимодействия с пользователем.

Модуль взаимодействия с клиентской частью облачного сервиса состоит из набора RoR-контроллеров, предоставляющих REST API к ресурсам облачного сервиса. Для сущностей БД облачного сервиса такие контроллеры строятся автоматически при помощи объектного представления реляционной БД ActiveRecord.

Модуль взаимодействия с вычислительными установками по протоколу SSH основан на Ruby-библиотеке Net-SSH-shell [8].

Разработка клиентской части облачного сервиса предполагает использование стандартного стека веб-технологий: HTML5, JavaScript, CSS3. На сегодняшний день существует множество прекомпиляторов стандартного стека веб-технологий HTML, CSS, JavaScript, упрощающих разработку приложений различной специфики. С 2012 года мировыми лидерами веб-разработки активно используются шаблоны проектирования клиентской части веб-приложения. Такие шаблоны позволяют освободить разработчика от самостоятельного построения взаимодействия HTML-элементов страницы с JavaScript-логикой приложения. Код динамического веб-приложения, написанный без использования шаблонов проектирования, более чем наполовину состоит из функций взаимодействия страничных элементов с логикой приложения. Многие шаблоны проектирования позволяют структурировать изначально не имеющий стандартной структуры код JavaScript.

Авторами работы выбран шаблон проектирования Angularjs [9] от корпорации Google. На сегодняшний день среди конкурирующих шаблонов проектирования Angularjs имеет наилучшую производительность, поддержку и документацию.

Модуль Angular-devise средствами одноименной JavaScript-библиотеки организует клиентские функции механизма аутентификации Devise.

В соответствии с шаблоном проектирования Angularjs, клиентская часть веб-приложения представляет собой одно или несколько одностраничных веб-приложений. Это означает, что все страницы и их контроллеры в пределах одного одностраничного приложения загружаются в браузер клиента в связке (либо одним файлом при предварительной конкатенации). В отличие от традиционной архитектуры клиентской части, одностраничное приложение является целостным и самостоятельным. По запросу клиента такое приложение загружается с веб-сервера в браузер один раз, а дальнейшая работа в нём (переход по страницам приложения, создание/удаление новых элементов страницы) сопровождается только асинхронным обменом данных (AJAX) с веб-сервером без перезагрузки страницы.

3. Проект облачного сервиса

В соответствии с технологией MVC структура серверного приложения облачного сервиса разбивается на модели, представления и контроллеры. Использование каркаса Ruby on Rails позволило авторам автоматически сгенерировать (или использовать готовые) 80% необходимых контроллеров, 25% необходимых моделей и 20% необходимых представлений, что существенно ускорило процесс разработки.

Спроектированные модели данных облачного сервиса образовали схему базы данных. Часть БД, содержащая учётные записи пользователей (уникальный идентификатор, адрес электронной почты, шифрованный пароль, информацию о сессиях и параметры системы аутентификации), была автоматически сгенерирована системой аутентификации.

Интерфейс пользователя спроектирован на трех страницах: «Шаблоны заданий», «Задания» и «Детали задания». Страницам соответствуют представления и их контроллеры (рис. 5), которые обмениваются данными через общее хранилище. *Маршрутизатор* содержит правила перехода по веб-страницам, пути и URL к представлениям и контроллерам.

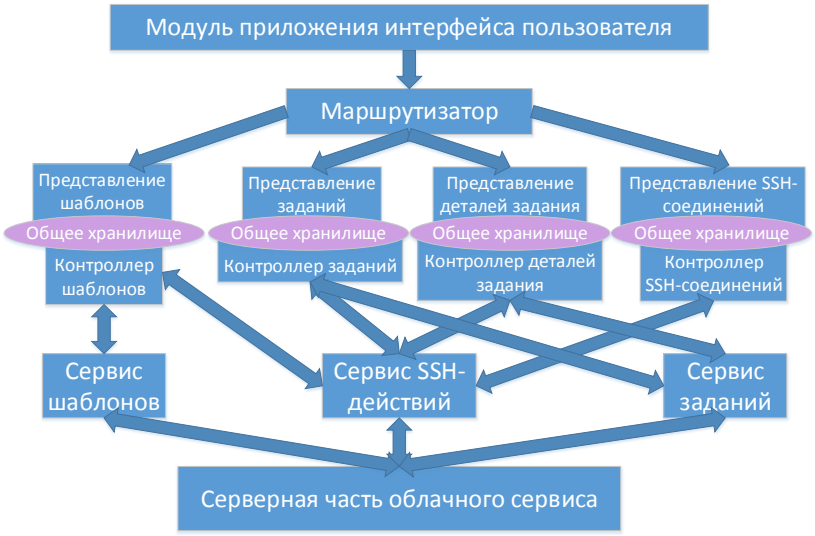

Рис. 5. Структура приложения интерфейса пользователя

Через контроллер шаблонов осуществляется работа пользователя с шаблонами заданий, в том числе:

- загрузка шаблонов пользователя, сохранённых в БД серверной части облачного сервиса;
- отправка нового шаблона для сохранения в БД серверной части облачного сервиса;
- удаление шаблона из БД серверной части облачного сервиса;
- запуск параллельного задания по выбранному шаблону.

Контроллеры заданий и деталей задания выполняют следующие функции:

- загрузка заданий пользователя, сохранённых в БД серверной части облачного сервиса;
- остановка выполнения выбранного задания;
- удаление из очереди СУППЗ выбранного задания;
- удаление завершённого задания из БД серверной части облачного сервиса;
- синхронизация состояния данных клиентской и серверной частей облачного сервиса каждые 2 секунды.

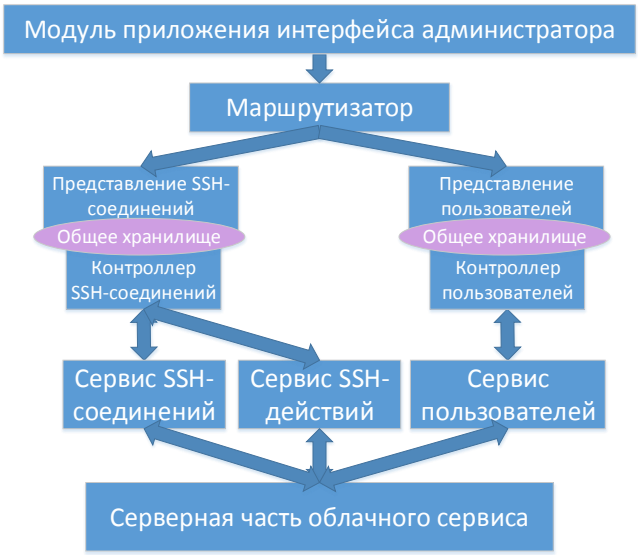

Рис. 6. Структура приложения интерфейса администратора

Принцип построения приложения администратора (рис. 6) схож с принципом построения приложения пользователя, с точностью до изменения функций контроллеров.

Контроллер пользователей выполняет следующие функции:

- загрузка зарегистрированных пользователей и доступных им соединений из БД серверной части облачного сервиса;
- загрузка соединений, сохранённых в БД серверной части облачного сервиса;
- сохранение в БД серверной части облачного сервиса обновлённой информации о доступных пользователям соединениях.

Главными модулями разработанного облачного сервиса, которые обеспечивают возможность осуществления высокопроизводительных вычислений, являются контроллеры SSH-соединений клиентской и серверной части. Именно с их помощью осуществляется связь веб-приложения и конечного пользователя с высокопроизводительными вычислительными установками под управлением СУППЗ, а также диспетчеризация пользовательских заданий.

Со стороны клиента контроллер SSH-соединений осуществляет:

- загрузку соединений, сохранённых в БД серверной части облачного сервиса;
- отправку нового соединения для сохранения в БД серверной части облачного сервиса;
- удаление соединения из БД серверной части облачного сервиса;
- тестирование работоспособности соединения.

Управление заданиями осуществляется вызовом действий контроллеров серверной части облачного сервиса, отвечающих за SSH-соединения с вычислительными установками. REST API к этим действиям организует сервис SSH-действий.

Контроллер SSH-действий серверной части выполняет следующие действия:

- проверка работоспособности SSH-соединения;
- запуск задания по заданному шаблону задания;
- остановка выполнения задания по заданному имени задания;
- удаление из очереди СУППЗ задания по заданному имени задания;
- получение информации о состоянии очереди СУППЗ.

В рассматриваемом проекте облачного сервиса для диспетчеризации заданий между доступными вычислительными установками используется следующий простейший алгоритм. Все вычислительные установки, соединение с которыми доступно пользователю, образуют циклический список, по которому перемещается специальная метка — *токен*. Перед запуском задания из доступных пользователю SSH-соединений выбирается следующее за соединением с токеном. Если соединение работоспособно, токен переходит к выбранному соединению, и задание помещается в очередь СУППЗ. Заметим, что со стороны СУППЗ постановка задание в очередь через облачный сервис ничем не отличается от постановки задания в очередь стандартным образом. Это позволяет обеспечить требуемое совмещение облачного и стандартного потоков заданий.

После размещения задания в СУППЗ облачный сервис способен отслеживать его состояние (в очереди, заблокировано, запущено, выполнено), а после запуска задания — определять и отображать процент выполненной ПК «Пирамида» вычислительной работы.

4. Пользовательский интерфейс облачного сервиса

При разработке интерфейса облачного сервиса применялся сервис NinjaMock [10]. Сервис предназначен для прототипирования веб-

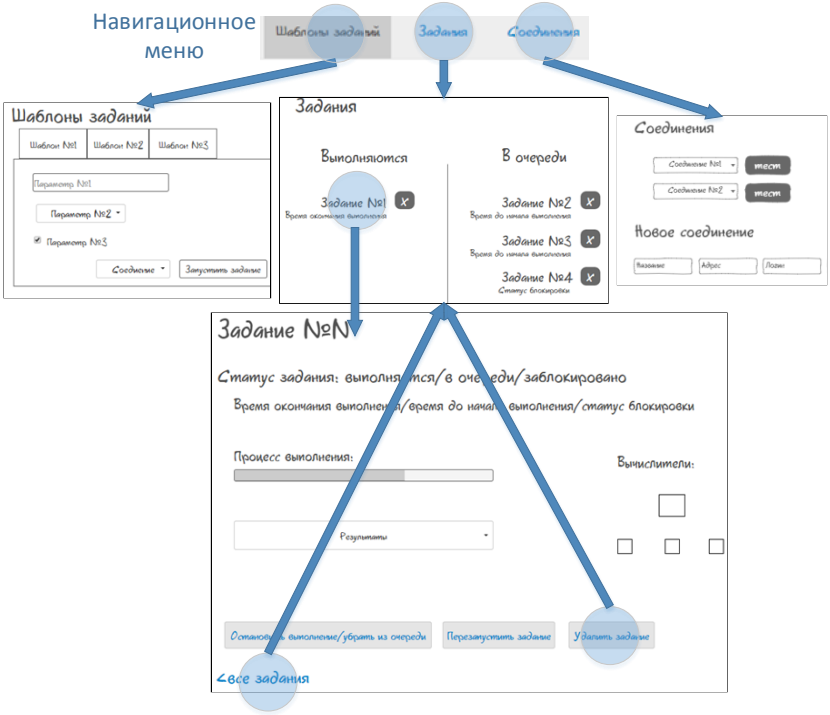

Рис. 7. Проект пользовательского интерфейса ПК «Пирамида»

интерфейсов и мобильных приложений. С его помощью созданы прототипы интерфейсов облачного сервиса.

Интерфейс параллельного SaaS-приложения предоставляет пользователю следующие элементы:

- форму создания и редактирования шаблонов заданий;
- элементы управления заданиями;
- элементы отображения информации о состоянии заданий и результатах их выполнения;
- форму создания и редактирования SSH-соединений;
- элементы управления SSH-соединениями.

Указанные элементы размещены на четырёх страницах разработанного прототипа интерфейса (рис. 7):

- шаблоны заданий;

- задания;
- соединения;
- детали задания.

В верхней части всех страниц находится навигационное меню, предназначенное для перехода на соответствующие страницы.

Под заголовком «Шаблоны заданий» находятся созданные ранее шаблоны заданий. При выборе конкретного шаблона отображаются его параметры с возможностью их редактирования. Под параметрами расположены элементы управления шаблоном: выбор соединения для запуска задания по выбранному шаблону и кнопка запуска задания. Под заголовком «Новый шаблон» находится форма создания нового шаблона задания.

На странице «Задания» отображаются все задания пользователя, сгруппированные по их статусу: на выполнении, в очереди (заблокировано), завершено. Задания отображаются в виде имени задания и дополнительной информации о его статусе под именем.

Дополнительной информацией может быть:

- прогнозируемое время окончания выполнения для заданий, находящихся на выполнении;
- прогнозируемое время до начала выполнения или статус блокировки для заданий, находящихся в очереди;
- время завершения для завершённых задач.

Кнопка «X» справа от имени задания в соответствии со статусом задания выполняет остановку выполнения задания, удаление задания из очереди или удаление информации о задании из базы заданий пользователя.

Нажатие на имя задания переводит на страницу «Детали задания», которая предоставляет подробную информацию о задании и результаты выполнения в реальном времени, а также элементы управления заданием. Прогресс-индикатор отображает процент выполненной работы ПК «Пирамида».

Под заголовком «Соединения» находятся созданные пользователем SSH-соединения. Нажав на имя соединения, в выпадающей форме можно редактировать параметры SSH-соединения и проверять работоспособность соединения.

Проект интерфейса администратора облачного сервиса представлен на рис. 8.

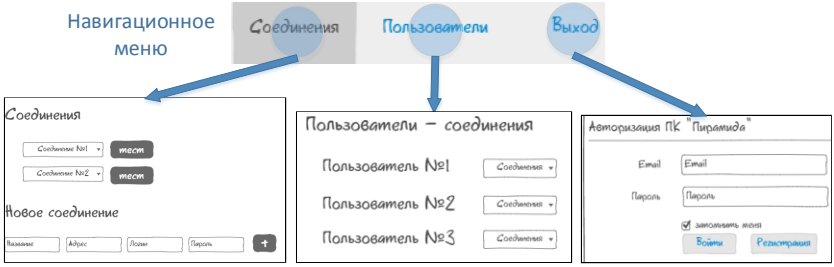

Рис. 8. Проект интерфейса администратора облачного сервиса

Страница «Пользователи» позволяет администратору выделить доступные каждому пользователю соединения.

Для неавторизованных пользователей стартовой страницей облачного сервиса является страница авторизации. Пользователю предлагается ввести логин (e-mail) и пароль своей учётной записи. Отметка «запомнить меня» позволяет сохранить введённые пользователем данные и не запрашивать их при повторном доступе к облачному сервису.

Незарегистрированный пользователь может пройти регистрацию, нажав кнопку «Регистрация».

5. Результаты опытной эксплуатации

Результатом настоящей работы стал макет облачного сервиса, установленный на испытательном стенде в МСЦ РАН. На стенде была проведена опытная эксплуатация макета, показавшая жизнеспособность предложенного авторами варианта построения облачного сервиса для высокопроизводительных приложений.

Полученный опыт разработки и опытной эксплуатации облачного сервиса позволяет определить следующие достоинства применённого авторами подхода.

(1) Выбранный авторами стек технологий и программных решений обеспечил высокую скорость разработки за счёт значительной степени автоматизации процесса создания веб-приложения.

(2) Созданный макет облачного сервиса, в отличие от существующих платформ для построения частных облаков, позволяет облачному сервису взаимодействовать с системами пакетной обработки высокопроизводительных вычислительных установок. Это обеспечивает требуемое совмещение стандартного и облачного потоков заданий.

(3) В созданный макет облачного сервиса интегрирован веб-интерфейс ПК «Пирамида», что существенно облегчает работу пользователя с этим комплексом.

Тем не менее, опытная эксплуатация выявила и отрицательные стороны применённого подхода:

(1) возникают трудности при глубокой модификации веб-приложения, построенного в соответствии с шаблоном проектирования Angularjs: модель шаблона не позволяет делать это легко и быстро;
(2) для успешного развития рассматриваемого варианта облачного сервиса необходима определённая стандартизация предлагаемого подхода и подключения большого числа разработчиков, что не совсем зависит от авторов.

Заключение

Для быстрого построения облачного сервиса вида SaaS, автоматизирующего рутинную работу пользователя по формированию параллельного задания и его размещению на суперкомпьютерных вычислительных установках, авторами были сформулированы требования к облачному сервису, рассмотрены существующие технологии и современные инструментальные средства для построения подобного сервиса вида SaaS, сделан выбор решений для построения облачного сервиса.

В результате был создан макет облачного сервиса в виде веб-приложения, которое реализует доступ к программному комплексу «Пирамида», развёрнутому на нескольких вычислительных установках под управлением отечественной системы пакетной обработки СУППЗ. Рассмотренное в статье сочетание технологий и средств построения облачного сервиса, использованных при создании макета, применено авторами впервые и является новым. Разработанный макет был установлен на испытательном стенде в МСЦ РАН, была проведена его опытная эксплуатация, по результатам которой можно говорить о жизнеспособности предложенного варианта построения облачного сервиса для высокопроизводительных приложений.

Список литературы

[1] А. В. Баранов, А. В. Киселёв, Е. А. Киселёв, В. В. Корнеев, Д. В. Семёнов. «Программный комплекс «Пирамида» организации параллельных вычислений с распараллеливанием по данным», *Научный сервис в сети интернет: суперкомпьютерные центры и задачи*, Труды

Международной суперкомпьютерной конференции, Изд-во МГУ, М., 2010, с. 299–302, URL: http://agora.guru.ru/abrau2010/pdf/299.pdf ↑6

[2] *Система Управления Прохождением Параллельных Заданий*, URL: http://suppz.jscc.ru ↑6

[3] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*, Dissertation submitted in partial satisfaction of the requirements for the degree of doctor of philosophy in Information and Computer Science. Chapter 5, University of California, Irvine, 2000, URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm ↑9

[4] M. H. Trygve. *Reenskaug/MVC. XEROX PARC 1978-79*, URL: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html ↑11

[5] Django. The web framework for perfectionists with deadlines, URL: https://www.djangoproject.com ↑12

[6] Ruby on rails, URL: http://rubyonrails.org ↑12

[7] А. Кончин. *В чём особенности и преимущества Ruby on Rails*, URL: https://anadea.info/ru/blog/why-rubyonrails-is-so-popular ↑12

[8] *A simple library to aid with stateful shell interactions*, URL: https://www.ruby-toolbox.com/projects/ssh-shell ↑12

[9] *Angularjs by Google — HTML enhanced for web apps!* URL: https://www.angularjs.org ↑13

[10] *NinjaMock — free tool for mobile app wireframes and website mockups*, URL: https://ninjamock.com/ ↑16

Рекомендовал к публикации *Программный комитет*
Четвёртого национального суперкомпьютерного форума ***НСКФ-2015***

Пример ссылки на эту публикацию:

А. В. Баранов, А. А. Зонов. «Вариант организации облачного сервиса для высокопроизводительных вычислений», *Программные системы: теория и приложения*, 2016, **7**:3(30), с. 3–23.
URL: http://psta.psiras.ru/read/psta2016_3_3-23.pdf

Об авторах:

Антон Викторович Баранов

Ведущий научный сотрудник МСЦ РАН, к.т.н., доцент. Области научных интересов: организация высокопроизводительных вычислений, планирование заданий и управление вычислительными ресурсами в суперкомпьютерах, технологии виртуализации и облачных вычислений

e-mail: antbar@mail.ru

Антон Андреевич Зонов

Стажёр-исследователь МСЦ РАН. Области научных интересов: технологии виртуализации и облачных вычислений, методы и средства разработки web-приложений

e-mail: andi4andi@gmail.com

Anton Baranov, Anton Zonov. *The variant of organization of cloud service for high-performance computing.*

Abstract. The article focuses on the creation of a cloud SaaS-service for Supercomputing Applications. The technology presented in the article show how to organize a cloud service based on software complex ≪Pyramid≫ and batch system of parallel tasks passing. The cloud service serves as an additional layer of abstraction that allows to combine various supercomputers and arrange for them a single management interface. (*In Russian*).

Key words and phrases: high-performance computing, cloud computing, Ruby on Rails, Angularjs, RESTful, Software as a Service, SaaS.

© A. V. Baranov, A. A. Zonov, 2016
© Joint Supercomputer Center of RAS, 2016
© Program systems: Theory and Applications, 2016

References

[1] A. V. Baranov, A. V. Kiselëv, Ye. A. Kiselëv, V. V. Korneyev, D. V. Semënov. "Programmnyy kompleks "Piramida" organizatsii parallel'nykh vychisleniy s rasparallelivaniyem po dannym", *Nauchnyy servis v seti internet: superkomp'yuternyye tsentry i zadachi*, Trudy Mezhdunarodnoy superkomp'yuternoy konferentsii, Izd-vo MGU, M., 2010, pp. 299–302 (in Russian), URL: http://agora.guru.ru/abrau2010/pdf/299.pdf

[2] *Sistema Upravleniya Prokhozhdeniyem Parallel'nykh Zadaniy* (in Russian), URL: http://suppz.jscc.ru

[3] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*, Dissertation submitted in partial satisfaction of the requirements for the degree of doctor of philosophy in Information and Computer Science. Chapter 5, University of California, Irvine, 2000, URL: http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[4] M. H. Trygve. *Reenskaug/MVC. XEROX PARC 1978-79*, URL: http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html

[5] Django. The web framework for perfectionists with deadlines, URL: https://www.djangoproject.com

[6] Ruby on rails, URL: http://rubyonrails.org

[7] A. Konchin. *V chëm osobennosti i preimushchestva Ruby on Rails* (in Russian), URL: https://anadea.info/ru/blog/why-rubyonrails-is-so-popular

[8] *A simple library to aid with stateful shell interactions*, URL: https://www.ruby-toolbox.com/projects/ssh-shell

[9] *Angularjs by Google — HTML enhanced for web apps!* URL: https://www.angularjs.org

[10] *NinjaMock — free tool for mobile app wireframes and website mockups*, URL: https://ninjamock.com/

Sample citation of this publication:

Anton Baranov, Anton Zonov. "The variant of organization of cloud service for high-performance computing", *Program systems: theory and applications*, 2016, **7**:3(30), pp. 3–23. (*In Russian*).

URL: http://psta.psiras.ru/read/psta2016_3_3-23.pdf