

Реферат

Отчёт 51с., 1 табл., 28 изображений, 2 листингов, 17 источников.

Распределённые системы обработки информации, распределённые вычисления, Python, Flask, PostgreSQL.

Предмет работы составляет разработка системы распределения вычислений.

Цель работы - создание программного обеспечения, осуществляющего распределение вычислительных задач по произвольным вычислительным узлам, причём полученная система должна быть распределённой.

В процессе работы была спроектирована структура системы и отдельные её компоненты. Были реализованы все сервисы системы и проведено их тестирование. Разработанная система предусматривает наличие механизма деградации функциональности и масштабирование некоторых узлов.

В результате была разработана система распределения вычислений и приведены инструкции по её пользованию.

Оглавление

1	Введение	8
2	Аналитический раздел	9
2.1	Существующие аналоги	9
2.2	Определение общей структуры комплекса	10
2.3	Возможные прецеденты	12
2.4	Осуществляемая деятельность	14
2.5	Выводы	16
3	Конструкторский раздел	17
3.1	Общая структура системы	17
3.2	Система мониторинга	20
3.3	Фронтэнд пользователей	21
3.4	Фронтэнд вычислительных узлов	23
3.5	Система управления сессией	26
3.6	Система управления	27
3.7	Система хранения данных	29
3.8	Система хранения файлов	32
3.9	Система балансировки нагрузки	32
3.10	Система вычисления	34
3.11	Выводы	35
4	Технологический раздел	36
4.1	Выбор языка программирования	36
4.2	Выбор программных средств	36
4.3	Система мониторинга	37
4.4	Фронтэнд пользователей	37
4.5	Фронтэнд вычислительных узлов	38
4.6	Система управления сессией	39
4.7	Система управления	39
4.8	Система хранения файлов	40
4.9	Система хранения данных	40
4.10	Система балансировки нагрузки	41
4.11	Система вычисления	42

4.12	Выводы	42
4.13	Использование системы	43
4.14	Выводы	48
5	Заключение	49

Глоссарий

- пользователь – человек, формирующий задание комплексу на проведение некоего расчёта;
- задача – программа научно–прикладного характера, предоставленная в виде исполняемого файла;
- расчёт – процесс выполнения задачи, результатом которого являются некие файлы (зависящие от задачи), содержащие результаты его работы. Подразумевается, что расчёт занимает значительное (от нескольких часов и до нескольких дней) время;
- комплекс - вся система распределённых вычислений
- пользовательский интерфейс, ПИ – интерфейс, используемый для постановки задач комплексу и управления ходом их расчётов;
- база данных, БД – выделенный сервер или программный компонент, отвечающий за хранение и доступ к данным;
- персональный компьютер, ПК – электронно–вычислительная машина архитектуры IBM PC;
- вычисляющий компьютер, ВК – ПК с установленным ПО, обеспечивающим взаимодействие данного ПК с комплексом и проведение расчётов на данном ПК;
- СОА – “сервис-ориентированная архитектура(SOA)”, подход к разработке программного обеспечения на основе слабосвязанных компонентов, взаимодействующих посредством стандартизованных интерфейсов;
- сервер – объект клиент – серверного взаимодействия, осуществляющий обслуживание клиентов
- клиент – объект клиент – серверного взаимодействия, инициирующий запрос серверу
- бекенд – сервер, элемент декомпозиции СОА, отвечающий за выполнение определенной подзадачи(работы с определенным типом данных, балансировку и т.д.);

- фронтенд – сервер, элемент декомпозиции СОА, отвечающий за перенаправление запросов бекендам и предоставление ПИ и/или интерфейса приложения.
- СХД – система хранения данных
- СХФ – система хранения файлов
- СУ – система управления
- ФП – фронтенд пользователей
- СУС – система управления сессией
- СБН – система балансировки нагрузки
- ФВУ – фронтенд вычислительных узлов
- ВУ – вычислительный узел

1. Введение

В ходе исследовательских работ в разных областях у членов научно-исследовательских и технических коллективов часто возникают задачи расчёта небольших программ, призванных проверить какую-либо гипотезу. Время работы подобных программ, несмотря на их простоту, может достигать нескольких часов, и в рамках коллектива часта ситуация, когда в любой момент времени кто-либо проводит какие-либо расчёты. В то же время, для каждого конкретного исследователя время расчёта подобных программ не занимает всё доступное. Част режим работы, в котором конкретный сотрудник несколько дней планирует вычислительный эксперимент, после чего ему необходимо поставить его на выполнение на несколько часов. В эти несколько часов его компьютер находится под высокой нагрузкой; однако во время нескольких дней планирования он по большей части простаивает.

В связи с этим возникла необходимость реализации программного комплекса, позволяющего исследователям в рамках коллектива загружать вычислительными задачами компьютеры друг друга. Комплекс должен быть прост в обращении и не требовать особой доработки программного обеспечения вычислительных экспериментов для их расчёта.

2. Аналитический раздел

В данном разделе обосновывается актуальность задачи и выполняется анализ предметной области. Результаты анализа представляются в виде диаграмм прецедентов и деятельности.

2.1. Существующие аналоги

Подобные системы разрабатываются с 1994 года, и в общем случае их называют системами “добровольных вычислений”. Среди программного обеспечения, используемого для организации таких вычислений, наиболее распространены системы XtremWeb, Xgrid, Grid MP и BOINC. Все подобные программы работают по одному и тому же принципу – пользователь в заданном формате передаёт системе свою программу; система отправляет эту задачу на выполнение какому-либо из вычислительных узлов, получает ответ и отдаёт его пользователю.

Xgrid – технология, разработанная компанией Apple, позволяющая объединять группу компьютеров в виртуальный суперкомпьютер для проведения распределённых вычислений. Из преимуществ данной системы можно выделить наличие предустановленных клиентов на компьютерах под управлением MAC OS X; однако её недостатки весьма существенны – во-первых, существует только реализация для MAC OS X; во-вторых, для доступа к каким-либо функциям комплекса, кроме просто однопоточного запуска программы, исполняемая программа должна быть специально спроектирована с учётом особенностей системы и только на языке Objective-C.

Grid MP – технология, разработанная компанией Univa. Символы MP в названии не имеют официальной расшифровки. Предоставляет web API для манипулирования объектами системы, что позволяет вести разработку для комплекса на практически любом языке программирования, но в то же время требует разработки программ специально под комплекс.

BOINC – открытая программная платформа университета Беркли для GRID вычислений. Обеспечивает валидацию вычислений за счёт избыточности, отслеживание конкретного вклада пользователей в расчёты, управление участием в различных экспериментах; однако рассчитан на огромные по масштабам проекты (тысячи и сотни тысяч вычислителей; наиболее крупные проекты насчитывают до 15 миллионов участников). В связи с этим, процесс настройки проекта занимает значительное время.

TORQUE – (Terascale Open-Source Resource and QUEUE Manager) – менеджер распределённых ресурсов для вычислительных кластеров из машин под управлением Linux и других Unix-подобных операционных систем. Существует порт под Windows.

Таким образом, из известных систем подобного рода ни одна не занимает целевую нишу ввиду следующих особенностей:

- Xgrid – поддерживает только MAC OS системы, исполняемая программа должна быть написана специально для работы с данной системой;
- Grid MP – коммерческий продукт, данных о ценах нет в наличии; исполняемая программа должна быть написана специально для работы с данной системой;
- BOINC – избыточная для поставленной задачи функциональность, исполняемые прикладные программы должны сильно дорабатываться для совместимости с проектом;
- TORQUE – система не предусматривает механизм деградации функциональности.

2.2. Определение общей структуры комплекса

Комплекс должен удовлетворять следующим требованиям:

- структура системы должна следовать принципам COA;
- количество качественно различных сервисов, из которых должна состоять система, должно быть не меньше 4-х;
- не менее 3-х сервисов(бекендов) должны быть горизонтально масштабируемыми;
- взаимодействие сервисов должно осуществляться по HTTP протоколу с учётом рекомендаций REST, если не доказана необходимость отказа от такого решения;
- отслеживание авторизационных ключей должно осуществляться отдельно выделенным сервисом;
- база данных комплекса должна поддерживать репликацию;

В результате анализа требований, была определена структура сети. Топология проектируемой системы представлена на рис. 1.

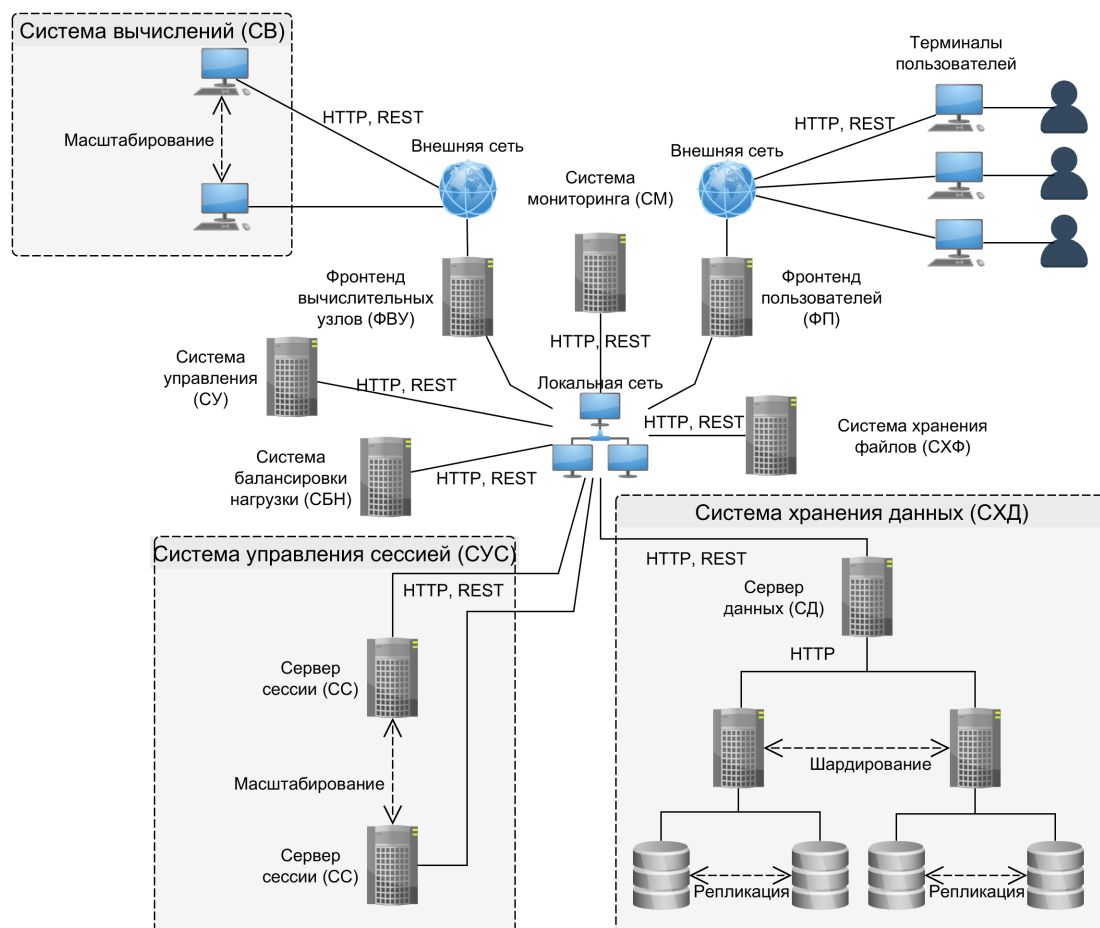


Рисунок 1: Топология проектируемой системы.

2.3. Возможные прецеденты

Комплекс при его работе предоставляет пользователю следующие варианты использования:

- регистрация пользователя;
- авторизация пользователя;
- постановка задачи на исполнение;
- просмотр статуса задачи;
- отмена задачи.

Диаграмма этих и дополнительных служебных прецедентов приведена на рис. 2.

С учётом требований к разделению внутреннего функционала комплекса, диаграмма прецедентов на рис. 2 расщепляется на набор диаграмм, соответствующих каждой из выделенных подсистем. Соответствующие диаграммы приведены на рисунках 3,4,5.

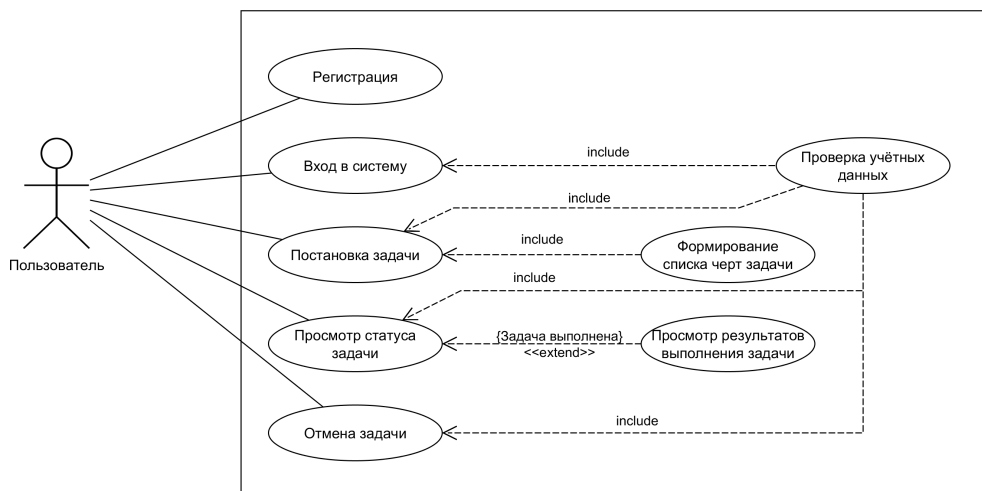


Рисунок 2: Диаграмма прецедентов всего комплекса в целом

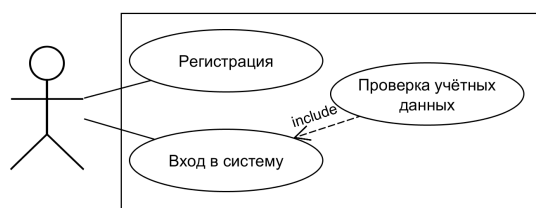


Рисунок 3: Диаграмма прецедентов СУС

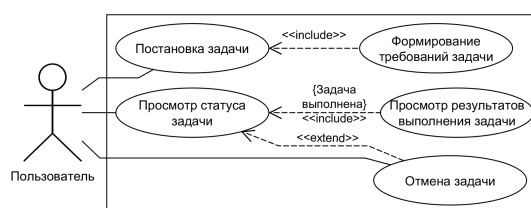


Рисунок 4: Диаграмма прецедентов СУ

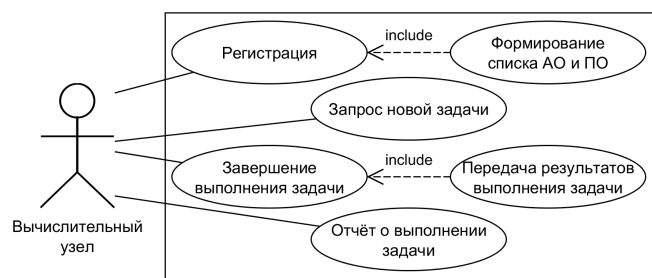


Рисунок 5: Диаграмма прецедентов СБН

2.4. Осуществляемая деятельность

Прецеденты, описанные в предыдущем пункте, отвечают определённой деятельности. Диаграмма деятельности на рис. 6 описывает полный процесс взаимодействия пользователя с комплексом.

С учётом требований к разделению внутреннего функционала комплекса, диаграмма деятельности на рис. 6 расщепляется на набор диаграмм, соответствующих определённым подсистемам из выделенных.

Диаграммы действий прецедентов подсистемы управления сессией “регистрация” и “вход в систему” приведены на рисунках 7 и 8 соответственно.

Диаграммы действий прецедентов системы балансировки нагрузки “регистрация”, “запрос новой задачи” и “завершение выполнения задачи” приведены на рисунках 9, 10 и 11 соответственно.

Диаграммы действий прецедентов системы управления “постановка задачи” и “просмотр статуса задачи” приведены на рисунках 12 и 13 соответственно.

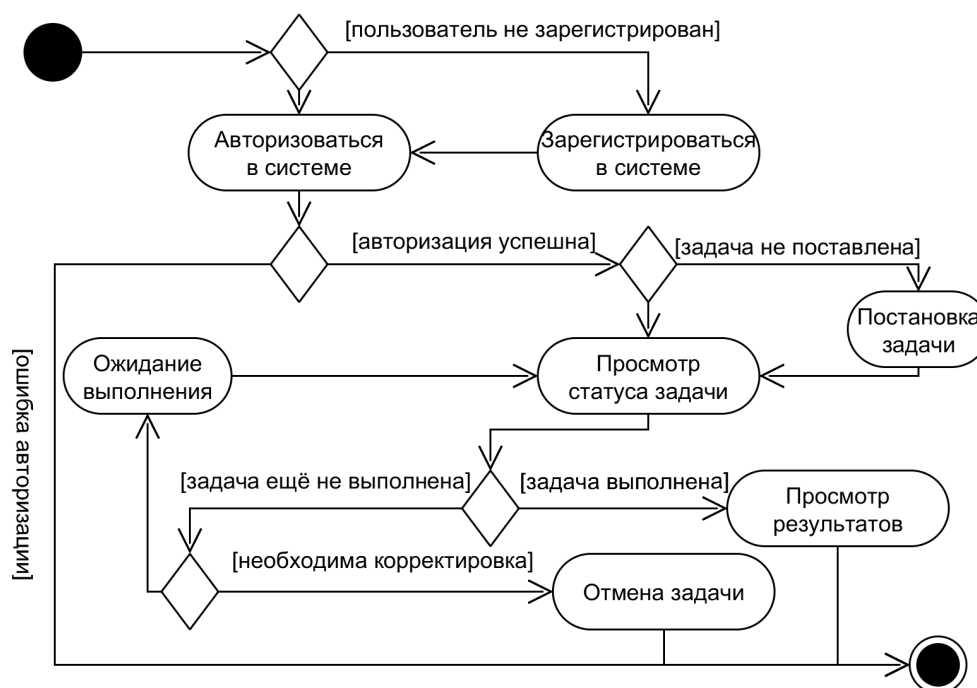


Рисунок 6: Диаграмма действий прецедента “общая деятельность” для системы в целом

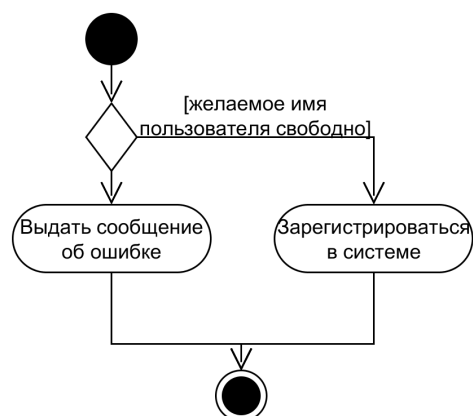


Рисунок 7: Диаграмма действий прецедента “регистрация” СУС

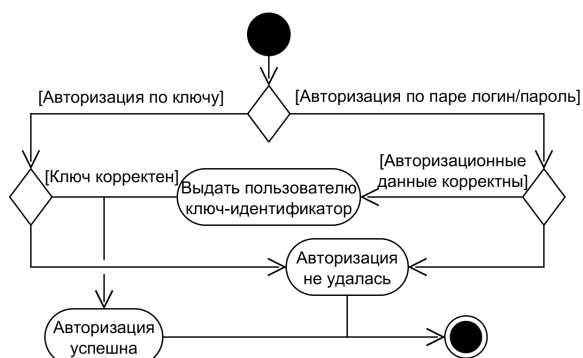


Рисунок 8: Диаграмма действий прецедента “вход в систему” СУС

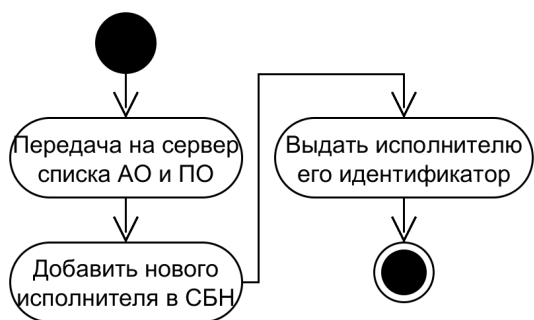


Рисунок 9: Диаграмма действий прецедента “регистрация” СБН

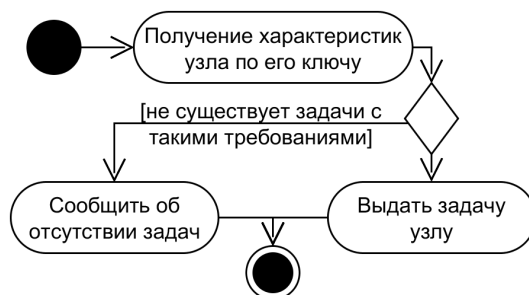


Рисунок 10: Диаграмма действий прецедента “запрос новой задачи” СБН

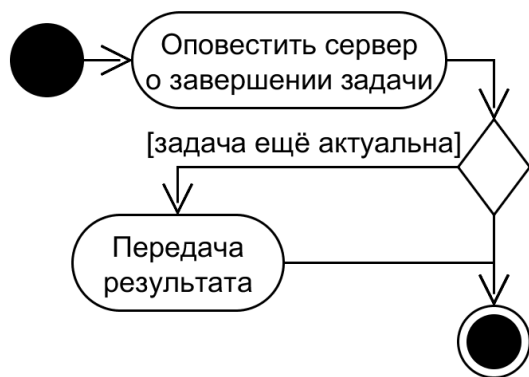


Рисунок 11: Диаграмма действий прецедента “завершение выполнения задачи” СБН

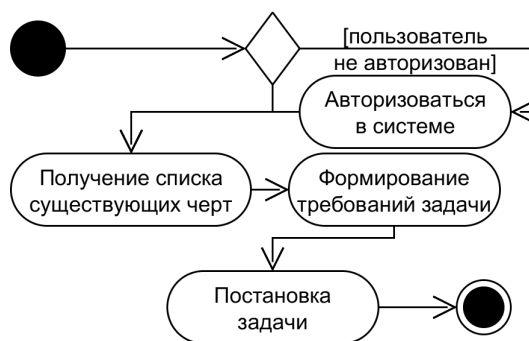


Рисунок 12: Диаграмма действий прецедента “постановка задачи” СУ

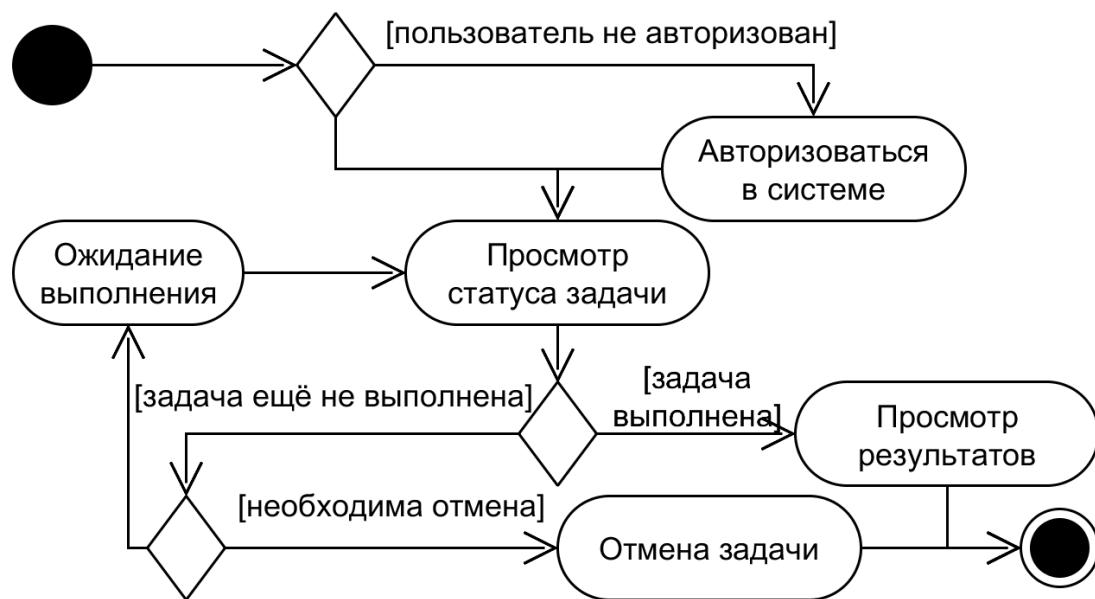


Рисунок 13: Диаграмма действий прецедента “просмотр статуса задачи” СУ

2.5. Выводы

В данном разделе были приведены диаграммы, описывающие функционал основных узлов системы. Данный анализ в дальнейшем используется для более строгой формализации функционала подсистем.

3. Конструкторский раздел

В данном разделе приводятся результаты проектирования системы. С применением UML-диаграмм описывается общая структура комплекса и требуемый функционал отдельных узлов системы.

3.1. Общая структура системы

Для того, чтобы удовлетворить требованиям по предоставлению механизма деградации функциональности, а также для упрощения процесса разработки, комплекс должен быть разделен на отдельные слабосвязанные элементы.

Различные подсистемы комплекса имеют некую модель поведения. Поведение подсистемы описывается её активной и пассивной частью. Активная часть соответствует действиям, которые подсистема выполняет разово либо с некоторой периодичностью, в автоматическом режиме. Пассивная часть соответствует API подсистемы. Взаимосвязи между различными компонентами системы приведены на диаграмме компонентов на рис. 14. Физическое размещение компонент по отдельным узлам проиллюстрировано на диаграмме развёртывания на рис. 15.

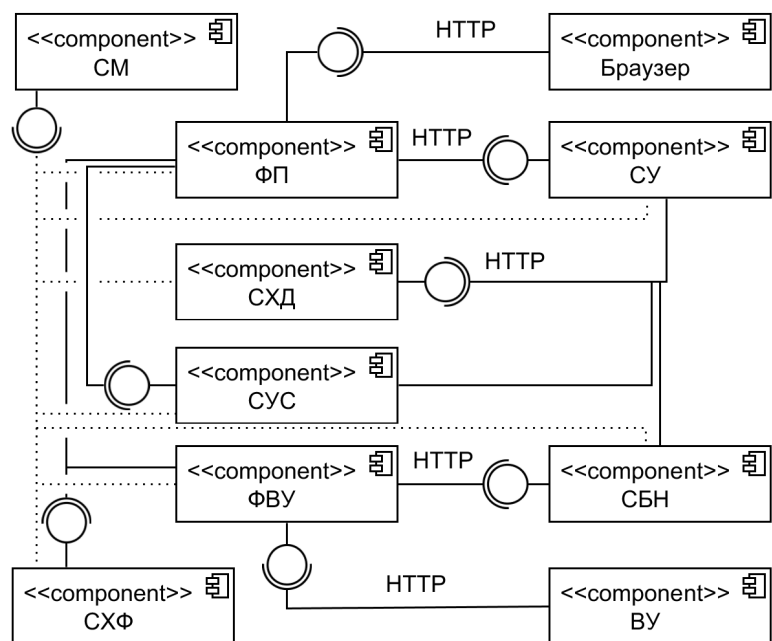


Рисунок 14: Диаграмма компонент комплекса

3.2. Система мониторинга

Задача данной подсистемы – отслеживание топологии сети. Все узлы комплекса должны оповещать СМ о своём статусе работы, и любой узел может получить от комплекса список активных в данный момент узлов. Данная система является полностью пассивной.

Невозможность любой другой подсистемы связаться с системой мониторинга рассматривается как ошибка сети, нарушающая нормальное функционирование комплекса.

Пассивная часть

Исходя из требований к СМ и с учётом REST-методик, она должна предоставлять следующее API:

- Ресурс: `/services`
Метод: GET
Результат: список активных сервисов
- Ресурс: `/services/type`
Метод: GET
Результат: список активных сервисов такого типа
- Ресурс: `/services/type`
Метод: POST
Параметры: `port`, `state`?
Результат: сообщение об успешной регистрации сервиса и распознанный адрес сервиса
Ошибки: отсутствует параметр “`port`”: HTTP 422
- Ресурс: `/services/type/address`
Метод: GET
Результат: статусное сообщение выбранного сервиса, аннотированное временем создания
Ошибки: сервис не найден: HTTP 404
- Ресурс: `/services/type/address`
Метод: PUT
Параметры: `state`?

```

1  {
2  "type1": {
3  "ipaddr1:port1": {
4  "state": "message1",    // arbitrary string
5  "lastbeat": "datetime1" // ex. "2015-05-25 01:46:35.857670"
6  },
7  "ipaddr2:port2": {
8  "state": "message2",
9  "lastbeat": "datetime2"
10 }
11 },
12 "type2": {
13 "ipaddr3:port3": {
14 "state": "message3",
15 "lastbeat": "datetime3"
16 }
17 }
18 }

```

Листинг 1: Пример JSON-представления коллекции верхнего уровня сервиса мониторинга

Результат: сообщение об успешном обновлении статусного сообщения

Коллекция верхнего уровня **/services** имеет словарь JSON-формата, приведённого на примере в листинге 1. Запросы к коллекциям более глубокого уровня **/services/type** и **/services/type/address** отображаются в подсловари **root[type]** и **root[type][address]** словаря верхнего уровня, соответственно.

3.3. Фронтэнд пользователей

Задача данной подсистемы – проверки безопасности и перенаправление запросов от пользователей к системе управления, а также отрисовка веб-интерфейса.

Активная часть

- В ходе конфигурирования данной системы необходимо в ручном порядке указать адрес системы мониторинга.

- ФП должен регистрироваться на СМ и оповещать её о своём состоянии с некоторой периодичностью.
- В ходе работы ФП должен получать со стороны СМ информацию о текущем адресе СУ, СУС и СХФ.

Пассивная часть

Исходя из требований к ФП, он должен предоставлять следующий функционал:

- регистрация пользователя
- авторизация пользователя
- просмотр списка существующих в системе черт
- постановка задачи с передачей в систему описания задачи, содержащего:
 - файл со списком черт задачи
 - файл-архив, содержащий задачу в виде исполняемого файла с именем **start** и необходимые для её работы дополнительные файлы
- просмотр статусов поставленных задач
- для выполненных задач – просмотр результатов выполнения
- удаление задач

На файлы, составляющие описание задачи, накладываются следующие ограничения:

- Файл **traits.txt** должен содержать набор строк в кодировке UTF-8. В каждой строке должна быть записана одна черта задачи. Черта задачи состоит из имени черты и версии черты, разделёнными одним или несколькими пробелами. И черта, и версия описываются произвольными строками без пробелов. Файл может заканчиваться пустой строкой. Строки, не подпадающие под такое описание, игнорируются. Пример корректного файла дан в листинге 2.
- Файл **start** с расширением **.sh**, **.bat**, **.exe** или **.py** должен находиться в **.tar.gz** - архиве, в корневой директории.

```

1  cuda_version          5.5
2  opengl_version       4.5
3
4  this_string_will_be_ignored
5
6  platform              linux
7  GlasgowHaskellCompiler 7.10.1
8

```

Листинг 2: Пример корректного файла со списком черт задачи

- Файл `start` должен корректно исполняться на вычислительном узле, удовлетворяющем требованиям, описанных чертами задачи. Именно этот файл будет запущен вычислительным узлом для начала расчётов. По завершении выполнения задачи она должна закрыть все созданные окна и уничтожить все порождённые процессы.

3.4. Фронтэнд вычислительных узлов

Задача данной подсистемы – перенаправление запросов от вычислительных узлов на балансировщик нагрузки.

Активная часть

- В ходе конфигурирования данной системы необходимо в ручном порядке указать адрес системы мониторинга.
- ФВУ должен зарегистрироваться на СМ и оповещать её о своём состоянии с некоторой периодичностью.
- В ходе работы ФВУ должен получать со стороны СМ информацию о текущем адресе СБН и СХФ.

Пассивная часть

Исходя из требований к ФВУ и с учётом REST-методик, он должен предоставлять следующее API:

- Ресурс: `/nodes`
Метод: POST

Параметры: список черт и ключ вычислительного узла в виде JSON-объекта { "traits": [{ "name": "name1", "version": "version1" }, ...], "key": "... " }

Результат: сообщение об успешной регистрации узла и назначенный идентификатор { "status": "success", "agent_id": "identifier " }

- **Ресурс:** /nodes/nodeid

Метод: PUT

Параметры: состояние расчёта, form/query-параметры "state", "key_old". Для обновления ключа также добавлен параметр "key".

Результат: сообщение об успешном обновлении статуса { "status": "success" }

- **Ресурс:** /tasks/newtask

Метод: GET

Параметры: идентификатор вычислительного узла (form/query - параметр "nodeid", совпадает с полученным при регистрации "agent_id"), ключ вычислительного узла (form/query - параметр "key")

Результат: пакет данных, описывающих задачу (gz - архив, переданный комплексу при создании задачи)

Ошибки: подходящих задач нет: HTTP 404; идентификатор не распознан либо отсутствует: HTTP 422

- **Ресурс:** /tasks/taskid

Метод: POST

Параметры: идентификатор вычислительного узла (form/query - параметр "nodeid"), ключ ВУ (form/query - параметр "key") результат выполнения задачи (file- параметр "file")

Результат: сообщение об успешном приёме результата

Ошибки: ошибка идентификатора узла, формата задачи либо идентификатора задачи: HTTP 422

Методы API POST /nodes, PUT /nodes/nodeid, GET /tasks/newtask и POST /tasks/taskid соответствуют прецедентам "регистрация", "запрос новой задачи", "отчёт о выполнении задачи" и "завершение выполнения задачи" соответственно.

Диаграммы последовательности действий при выполнении прецедентов "регистрация", "запрос новой задачи" и "завершение выполнения задачи" приведены на рис. 16, 17 и 18.

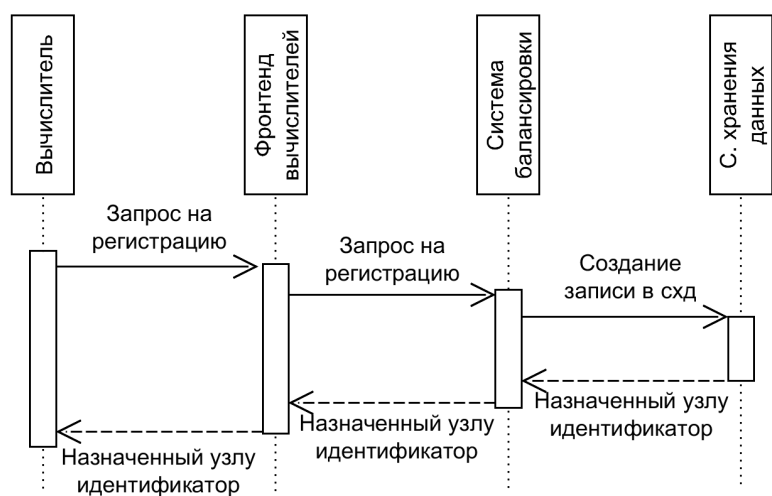


Рисунок 16: Диаграмма последовательности действий прецедента “регистрация” ФВУ

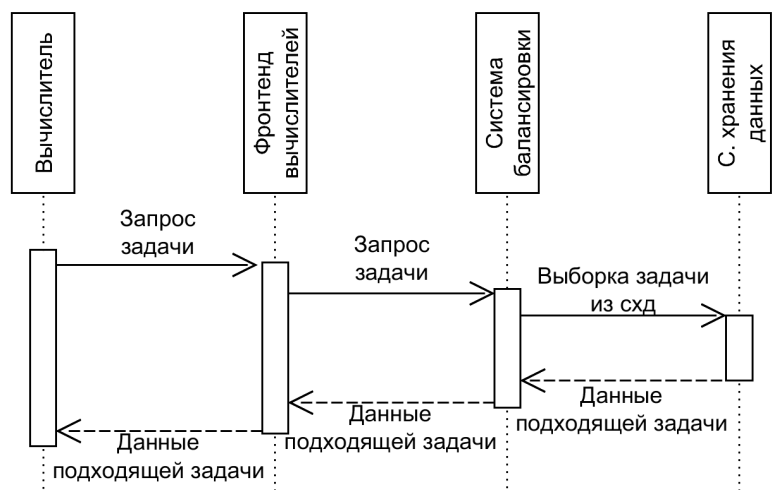


Рисунок 17: Диаграмма последовательности действий прецедента “запрос новой задачи” ФВУ

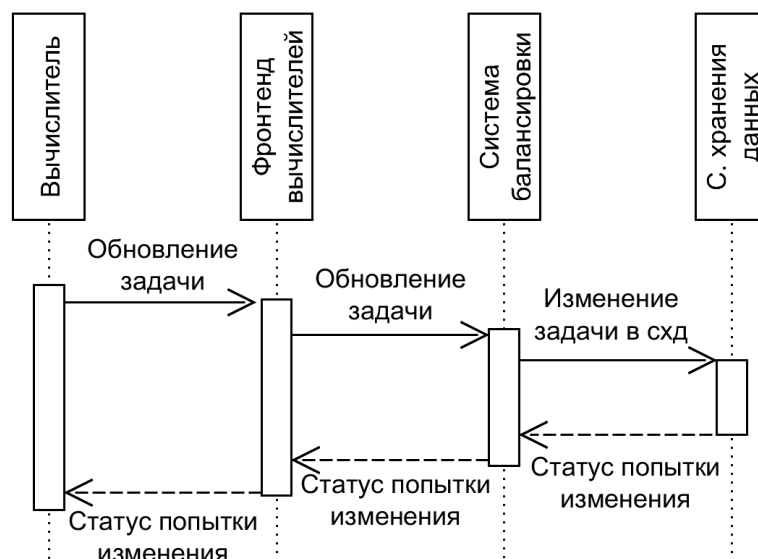


Рисунок 18: Диаграмма последовательности действий прецедента “завершение выполнения задачи” ФВУ

3.5. Система управления сессией

Задача данной подсистемы – регистрация, авторизация и аутентификация пользователей в сети.

Активная часть

- В ходе конфигурирования данной системы необходимо в ручном порядке указать адрес системы мониторинга.
- СУС должна зарегистрироваться на СМ и оповещать её о своём состоянии с некоторой периодичностью.
- В ходе работы СУС должна получать со стороны СМ информацию о текущем адресе СХД.

Пассивная часть

Исходя из требований к СУС и с учётом REST-методик, она должна предоставлять следующее API:

- Ресурс: `/users`

Метод: POST

Параметры: желаемая пара логин / пароль (хешированный). Form/query/JSON
“username”, “pw_hash”

Результат: сообщение об успешной регистрации пользователя { “status
”:”success”, “user_id”:” identifier ” }

Ошибки: пользователь с таким именем уже зарегистрирован: HTTP
422

- **Ресурс:** /login

Метод: GET

Параметры: пара логин / пароль (хешированный). Form/query/JSON
“username”, “pw_hash”

Результат: сгенерированный ключ доступа { “status”:”success”, “session_id
”:” identifier ” }

Ошибки: некорректная пара логин / пароль: HTTP 403, некоррект-
ный синтаксис запроса: HTTP 422

- **Ресурс:** /auth

Метод: GET

Параметры: ключ доступа: Form/query/JSON “session_id”

Результат: сообщение об успешной проверке ключа, соответствующий
ключу идентификатор пользователя { “status”:”success”, “user_id”:
” identifier ” }

Ошибки: некорректный ключ: HTTP 403, некорректный синтаксис
запроса: HTTP 422

- **Ресурс:** /logout

Метод: GET

Параметры: ключ доступа: Form/query/JSON “session_id”

Результат: сообщение об успешной разрегистрации ключа { “status”:
”success”, “user_id”:” identifier ” }

Ошибки: некорректный ключ: HTTP 403, некорректный синтаксис
запроса: HTTP 422

3.6. Система управления

Задача данной системы – предоставление API, позволяющего интерфейсной части (фронтенду вычислительных узлов) осуществлять взаимодействие пользователя с комплексом.

Активная часть

- В ходе конфигурирования данной системы необходимо в ручном порядке указать адрес системы мониторинга.
- СУ должна регистрироваться на СМ и оповещать её о своём состоянии с некоторой периодичностью.
- В ходе работы СУ должна получать со стороны СМ информацию о текущем адресе СХД.

Пассивная часть

Исходя из требований к СУ и с учётом REST-методик, она должна предоставлять следующее API:

- **Ресурс:** /traits

Метод: GET

Результат: Список всех известных системе черт в формате JSON. {
"result": [{ "name": "...", "version": "... " }, ...] }

- **Ресурс:** /tasks

Метод: POST

Параметры: идентификатор пользователя, список черт, допустимое время простоя, число экземпляров задачи, имя архива с содержанием задачи и отображаемое имя задачи в формате JSON. { "uid": "...", "traits": [...], "task_name": "...", "archive_name": "...", "max_time": "...", "subtask_count": "... " }

Результат: сообщение об успешном создании задачи { "status": "success", "task_id": "... " }

Ошибки: Неверный синтаксис запроса: HTTP 422

- **Ресурс:** /tasks

Метод: GET

Параметры: идентификатор пользователя. JSON { "uid": "... " }

Результат: Список задач пользователя, их черт, дат постановки комплексу и статусов их подзадач: { "status": "success", "tasks": [{ "taskname": "...", "traits": [...], "statuses": [{ "status": "... " }, { "result": "... " }, ...], "id": "...", "dateplaced": "... " }, ...] }

Ошибки: Неверный синтаксис запроса: HTTP 422

- Ресурс: /tasks/task_id

Метод: DELETE

Параметры: идентификатор пользователя. JSON { "uid": "..." }

Результат: Сообщение об успешной отмене задачи { "status": "success" }

Ошибки: Неверный синтаксис запроса, нет такой пары пользователь / задача: HTTP 422

- Ресурс: /system/state

Метод: GET

Результат: такой же, как и у СМ при запросе по ресурсу /services

3.7. Система хранения данных

Задача данной системы – хранение данных о задачах, вычислительных узлах и их чертах, а также предоставление API по доступу к этим данным.

Связи между разными типами хранимых данных предоставлены в виде ER-диаграммы на рис. 19.

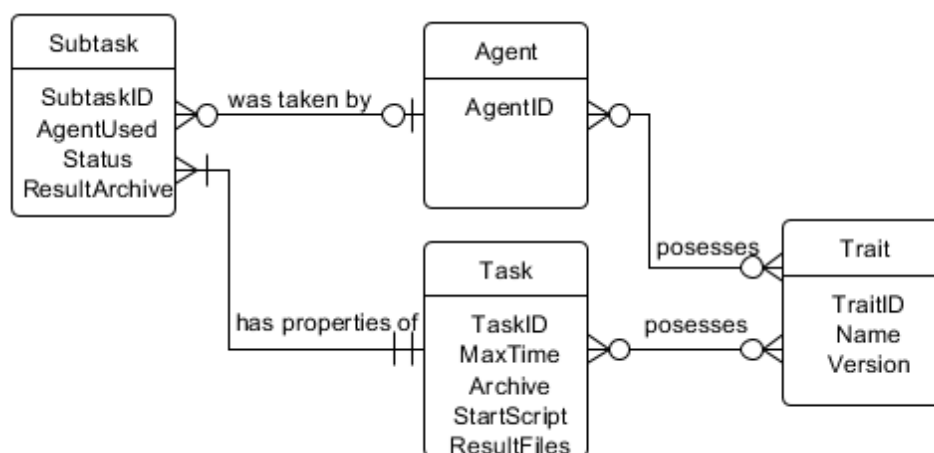


Рисунок 19: ER-диаграмма сущностей, хранимых в СХД

Активная часть

- В ходе конфигурирования данной системы необходимо в ручном порядке указать адрес системы мониторинга.

- СХД должна зарегистрироваться на СМ и оповещать её о своём состоянии с некоторой периодичностью.

Пассивная часть

Исходя из требований к СХД и с учётом REST-методик, она должна предоставлять следующее API:

Условные обозначения:

- table – имя таблицы в БД
- PKC – поле, являющееся (целочисленным) первичным ключом
- GFSbAI – get_free_subtask_by_agent_id
- object – {field: value, ...}, JSON-представление табличного объекта
- object <table> – JSON-представление объекта определенной таблицы <table>
- value <name> – значение, передаваемое любым возможным способом
- arrfilter object – {field: list value, ...}
- filter object – object, содержащий не более чем полный набор полей табличного объекта
- filter put object – {field:value, ..., “changes”: filter object}
- list object – список объектов
- int – целое число

Адрес	Метод	Ввод	Код	Вывод
/table	POST	object	200	empty
			400	error: Incorrect / insufficient input
			500	error: Postgres error or “entry already exists”
	GET	—	200	{“result”: list object}
			404	error: Not found

/table/filter	GET	filter object	200	{“result”: list object}
			400	error: Incorrect fields/values specified
	PUT	filter put object	200	{“count”: int}
			400	error: Incorrect fields/values specified
	DELETE	filter object	200	{“count”: int}
			400	error: Incorrect fields/values specified
/table/arrayfilter	GET	arrfilter object	200	{“result”: list object}
			400	error: Incorrect fields/values specified
/table/PKC/value	GET	object	200	object
			400	error: Incorrect fields/values specified
			404	error: Not found
	PUT	object	200	object
			400	error: Incorrect fields/values specified
			404	error: Not found
	DELETE	object	200	object
			400	error: Incorrect fields/values specified
			404	error: Not found
/custom/GFSbAI	GET	value agent_id	200	object subtask
			400	error: Incorrect value specified

Кроме указанных кодов ошибок, также все запросы могут вернуть в ответ ошибки со следующими кодами:

- 408 – Таймаут попытки доступа к некоторому шарду;

- 456 – Получены различные ошибки от шардов при выполнении запроса. Эта ошибка является следствием нарушения согласованности данных.

3.8. Система хранения файлов

Задача данной системы – хранение архивов задач и их результатов, и выдача их по запросу.

Активная часть

- В ходе конфигурирования данной системы необходимо в ручном порядке указать адрес системы мониторинга.
- СХФ должна зарегистрироваться на СМ и оповещать её о своём состоянии с некоторой периодичностью.

Пассивная часть

Исходя из требований к СХФ и с учётом REST-методик, она должна предоставлять следующее API:

- Ресурс: `/static`

Метод: POST

Параметры: файл, переданный с помощью параметра типа `multipart/form-data` “file”

Результат: Назначенный идентификатор файла { “name”:”... ” }

Ошибки: Неверный синтаксис запроса: HTTP 400

- Ресурс: `/static/path`

Метод: GET

Результат: файл по пути `path`

Ошибки: Файл отсутствует: HTTP 404

- Ресурс: `/static/path`

Метод: DELETE

Результат: Сообщение об успешном удалении файла: HTTP 200

Ошибки: Файл отсутствует: HTTP 404, ошибка при удалении файла: HTTP 500

3.9. Система балансировки нагрузки

Данная система отвечает за координацию задач и отслеживание состояний активных вычислительных узлов.

Активная часть

- В ходе конфигурирования данной системы необходимо в ручном порядке указать адрес системы мониторинга.
- СБН должна зарегистрироваться на СМ и оповещать её о своём состоянии с некоторой периодичностью.
- В ходе работы СБН должна получать со стороны СМ информацию о текущем адресе СХД.

Пассивная часть

Исходя из требований к СБН и с учётом REST-методик, она должна предоставлять следующее API:

- **Ресурс:** /nodes
Метод: POST
Параметры: Список черт узла, ключ узла { "key":" ... ", "traits " :[...] }
Результат: Идентификатор узла { "agent_id":" ... " }
Ошибки: Ошибка при создании узла с заданными параметрами: HTTP 422
- **Ресурс:** /nodes/nodeid
Метод: PUT
Параметры: Состояние узла (form-параметр "state")
Результат: Сообщение об успешном изменении состояния узла { "status ":"success" }
Ошибки: Узел не распознан: HTTP 404
- **Ресурс:** /tasks/newtask
Метод: GET
Параметры: Идентификатор узла (form-параметр "nodeid")
Результат: Имя архива назначенной задачи, JSON { "archive_name":" ... " }
Ошибки: Не найдено подходящей задачи: HTTP 404, неверный синтаксис запроса: HTTP 422
- **Ресурс:** /tasks

Метод: POST

Параметры: Идентификатор узла (form-параметр “nodeid”), имя ари-
ва с результатом задачи

Результат: Сообщение об успешном принятии результатов задачи { “
status”:”success” }

Ошибки: неверный синтаксис запроса: HTTP 422

3.10. Система вычисления

Данная система представлена набором вычислительных узлов с установ-
ленным на них специальным ПО, осуществляющем взаимодействие с осталь-
ными сервисами системы и управление ходом выполнения задачи.

Активная часть

В ходе конфигурирования данной системы необходимо в ручном порядке
указать адрес фронтенда вычислительных узлов.

ПО, обеспечивающее функционирование системы, должно удовлетворять
следующим требованиям:

- До подключения к серверу балансировки приложение должно предо-
ставлять возможность формирования списка черт, характеризующих
АО и ПО вычислительного узла
- После подключения к балансировщику (через фронтенд вычислитель-
ных узлов), с определённой периодичностью вычислительный узел дол-
жен опрашивать комплекс на предмет наличия доступных задач
- По получении задачи, вычислительный узел должен с определённой
периодичностью оповещать балансировщик о ходе выполнения задачи
- По завершении выполнения задачи, вычислительный узел должен пе-
редать балансировщику сведения о результате выполнения задачи

Диаграмма состояний ПО вычислительного узла, иллюстрирующая при-
ведённые выше соображения, приведена на рис. 20.

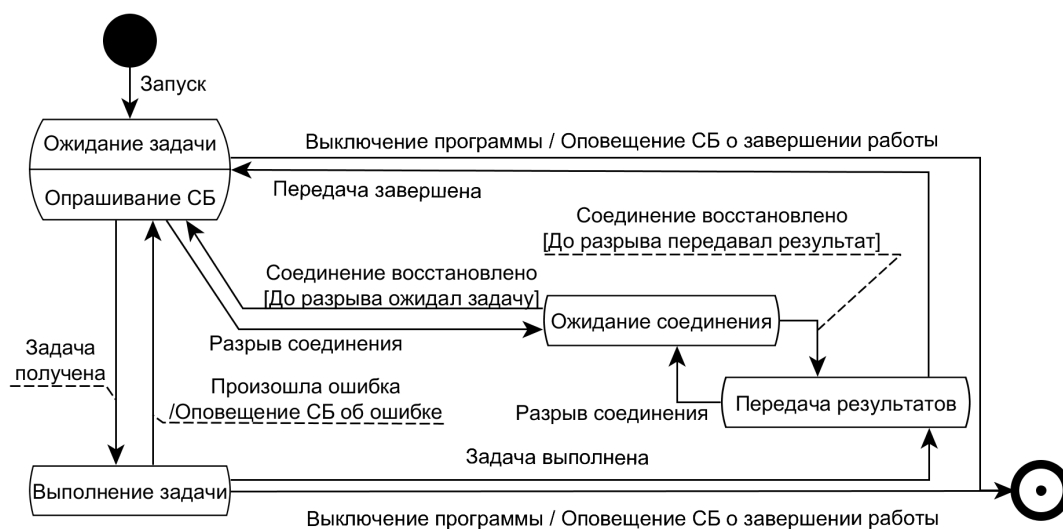


Рисунок 20: Диаграмма состояний ВУ

3.11. Выводы

В данном разделе были приведены результаты проектирования системы. Была описана общая структура комплекса и формализован требуемый функционал отдельных узлов системы. Было определено API отдельных сервисов.

4. Технологический раздел

В данном разделе производится выбор языка программирования и сопутствующих программных средств. Описываются основные моменты программной реализации и описываются основные из протестированных для каждой из подсистем сценарии. Поясняются действия пользователя по взаимодействию с комплексом. Описываются процедуры развертывания комплекса, постановки комплексу задачи, регистрации вычислительного узла в комплексе и получения результатов задачи, а также её отмены.

4.1. Выбор языка программирования

В качестве языка программирования был выбран python. Выбор был обусловлен наличием фреймворков для быстрого прототипирования веб-сервисов (Flask) и большого количества онлайн-документации (как по языку, так и по фреймворку), а также простотой развёртки сервисов.

4.2. Выбор программных средств

При разработке использовались следующие библиотеки:

- Python 3.4.3 [1] – основной дистрибутив языка
- Flask Microframework 0.10.1 [2] – фреймворк для разработки веб-сервисов
- jsonpickle 0.8.0 [3] – библиотека для работы с JSON-форматом
- hashlib [4] – библиотека для работы с криптографическими функциями
- Requests 2.7.0 [5] – библиотека для работы с HTTP-запросами
- psutil [6] – библиотека для работы с информацией о системе
- subprocess [7] – библиотека для работы с процессами
- SQLAlchemy 1.0.4 [8] – набор инструментов для работы с БД

Также использовалось следующее ПО:

- Yed 3.14.1 [9] – редактор диаграмм
- Inkscape 0.91 [10] – редактор векторной графики
- PostgreSQL 9.4.2 [11] – система управления базой данных
- SQLite 3.8.10.2 [12] – система управления базой данных

- tar [13, 14] – утилита для работы с .tar - архивами
- gzip [15, 16] – утилита для работы с .gz - архивами
- Nginx 1.8.0 [17] – веб-сервер

4.3. Система мониторинга

Реализация

Система была реализована с помощью фреймворка Flask. По ходу работы система осуществляет автоматическое удаление записей о давно не отвечавших узлах.

Тестирование

В ходе тестирования проверялись следующие сценарии:

- доступ к отсутствующему элементу через ресурсы `/services`, `/services/type` и `/services/type/address`;
- попытка создания записи по ресурсу `/services/test` без указания порта;
- попытка создания записи по ресурсу `/services/test`;
- выборка всех активных сервисов по ресурсу `/services`;
- выборка активных сервисов роли test по ресурсу `/services/test`;
- изменение статусного сообщения сервиса по ресурсу `/services/test/address`;
- проверка удаления сервиса из списка активных после определённого времени неактивности.

4.4. Фронтэнд пользователей

Реализация

Система была реализована с помощью фреймворка Flask. Веб-интерфейс был реализован с помощью механизма html-шаблонов Jinja2, встроенного в фреймворк. Сервис имеет директорию временных загрузок в директории временных файлов используемой операционной системы. Сервис отслеживает статус авторизации пользователя и стадию создания задачи (начало

создания, архив задачи загружен, черты задачи загружены) с помощью механизма кук. Сервис автоматически очищает директорию временных загрузок.

Тестирование

В ходе тестирования проверялись следующие сценарии:

- регистрация пользователя;
- регистрация пользователя с уже занятым именем;
- вход в систему с верными учётными данными;
- попытка входа в систему с неверными учётными данными;
- постановка задачи комплексу;
- попытка постановки задачи комплексу с неверными параметрами задачи;
- отмена поставленной задачи;
- просмотр состояния системы;
- загрузка результатов выполнения задачи.

4.5. Фронтэнд вычислительных узлов

Реализация

Система была реализована с помощью фреймворка Flask. Сервис имеет директорию временных загрузок в директории временных файлов используемой операционной системы. Сервис автоматически очищает директорию временных загрузок.

Тестирование

В ходе тестирования проверялись следующие сценарии:

- регистрация вычислительного узла;
- получение узлом новой задачи;
- оповещение узлом о состоянии расчёта задачи;
- загрузка узлом данных о результате расчёта задачи.

4.6. Система управления сессией

Реализация

Система была реализована с помощью фреймворка Flask. Сервис хранит данные о активных сессиях в базе данных SQLite.

Тестирование

В ходе тестирования проверялись следующие сценарии:

- регистрация пользователя;
- регистрация пользователя с уже занятым именем;
- получение номера сессии по верной паре логин/пароль;
- попытка получения идентификатора сессии по неверной паре логин/-пароль;
- получение идентификатора пользователя по активному идентификатору сессии;
- попытка получения идентификатора пользователя по отозванному идентификатору сессии;
- отзыв идентификатора сессии по активному идентификатору сессии;
- попытка отзыва идентификатора сессии по отозванному идентификатору сессии.

4.7. Система управления

Реализация

Система была реализована с помощью фреймворка Flask.

Тестирование

В ходе тестирования проверялись следующие сценарии:

- просмотр всех известных системе черт задач и узлов;
- постановка задачи с корректными данными задачи;
- получение данных о всех задачах пользователя по идентификатору пользователя;

- отзыв задачи от имени пользователя – владельца задачи;
- попытка отзыва задачи от имени пользователя, не являющегося владельцем задачи;
- получение данных о состоянии сервисов системы.

4.8. Система хранения файлов

Реализация

Система была реализована с помощью фреймворка Flask. Сервис имеет директорию временных загрузок в директории временных файлов используемой операционной системы. Сервис автоматически очищает директорию временных загрузок.

Тестирование

В ходе тестирования проверялись следующие сценарии:

- загрузка файла в систему;
- получение доступа к файлу по корректному идентификатору файла;
- попытка получения доступа к файлу по некорректному идентификатору файла;
- удаление файла по корректному идентификатору файла;
- попытка удаления файла по некорректному идентификатору файла.

4.9. Система хранения данных

Реализация

Система была реализована с помощью фреймворка Flask и ORM SQLAlchemy, с использованием СУБД PostgreSQL. Система состоит из одного или нескольких бекендов данных, предоставляющих REST-интерфейс к СУБД, и шардирующего бекенда, осуществляющего шардирование данных на бекенды данных. Шардирование производится функционально по целочисленным первичным ключам.

При запросах доступа к данным (GET/PUT/DELETE) с с передачей первичного ключа, номер шарда определяется отстакот от деления значения первичного ключа на количество шардов. При запросах добавления данных (POST) номер шарда выбирается по принципу round robin. При любых

фильтрующих запросах(без указания первичного ключа) шардирующий бекенд опрашивает все шарды и конкатенирует ответы перед отправкой пользователю.

Для сохранения непрерывности множества значений первичных ключей на шардах, бекенды данных производят преобразование глобальных значений первичных ключей в локальные по формуле $ID_{local} = \frac{ID - SHARD_NUM}{SHARD_COUNT}$ перед попыткой доступа по первичному ключу, и обратное преобразование $ID = ID_{local} \cdot SHARD_COUNT + SHARD_NUM$ перед ответом шардирующему бекенду.

Тестирование

В ходе тестирования проверялись следующие сценарии:

- запросы доступа/изменения/удаления данных по первичному ключу;
- фильтрующие запросы;
- запросы с фильтрацией по типу включения значения некоторого поля в передаваемое в запросе множество значений;
- специальные запросы;
- контролируемая деградация системы: при отключении некоторых шардов, запросы на добавление записей в БД и на прямой доступ по идентификатору (при условии, что идентификатор не принадлежал упавшему шарду) всё ещё работают.

4.10. Система балансировки нагрузки

Реализация

Система была реализована с помощью фреймворка Flask. Сервис автоматически отзывает задачи у неактивных вычислительных узлов и переназначает их свободным узлам.

Тестирование

В ходе тестирования проверялись следующие сценарии:

- регистрация вычислительного узла;
- получение вычислительным узлом задачи для выполнения;

- попытка получения вычислительным узлом задачи, при отсутствии подходящих задач;
- оповещение вычислительным узлом о состоянии расчёта задачи;
- попытка оповещения вычислительным узлом о состоянии расчёта задачи, в случае, если задача уже была отозвана;
- оповещение вычислительным узлом о завершении расчёта задачи.

4.11. Система вычисления

Реализация

Система была реализована с помощью фреймворка Flask. Сервис автоматически отзывает задачи у неактивных вычислительных узлов и переназначает их свободным узлам.

Тестирование

В ходе тестирования проверялись следующие сценарии:

- регистрация в комплексе;
- получение задачи для выполнения;
- попытка получения задачи, при отсутствии подходящих задач;
- оповещение комплекса о состоянии расчёта задачи;
- попытка оповещения комплекса о состоянии расчёта задачи, в случае, если задача уже была отозвана;
- оповещение вычислительным узлом о завершении расчёта задачи;
- исполнение корректно поставленной задачи;
- попытка исполнения некорректно поставленной задачи.

4.12. Выводы

В данном разделе был обоснован выбор языка программирования и перечислены использованные программные средства. Были описаны способы реализации отдельных сервисов и описаны основные из протестированных для каждой из подсистем сценарии. Были описаны процедуры развертывания

комплекса, постановки комплексу задачи, регистрации вычислительного узла в комплексе и получения результатов задачи, а также отмены задачи.

4.13. Использование системы

В данном разделе поясняются действия пользователя по взаимодействию с комплексом. Описываются процедуры развертывания комплекса, постановки комплексу задачи, регистрации вычислительного узла в комплексе и получения результатов задачи, а также её отмены.

Развертывание системы

Для развертывания системы необходимо запустить следующие сервисы в любом порядке, на ПК (одном или разных) с установленными Python 3.4.3 и библиотеками Flask, jsonpickle, Requests, и psutil. Большинство сервисов запускаются однотипно и не требуют настройки. Отличия в запуске у сервисов СМ, СХД и других оговорены ниже. Системы могут быть запущены в любом порядке.

Системы должны находиться в одной локальной сети. Соединение с внешней сетью требуется только сервисам фронтендов пользователей и вычислительных узлов, в случае, если конечные пользователи и / или вычислительные узлы не находятся в локальной сети.

Система мониторинга. Данная система реализована в виде программы на языке Python. Запуск данной системы происходит путём выполнения команды `beacon_backend.py port_number`, где `beacon_backend` является именем скрипта, реализующего подсистему, а `port_number` – номером порта, на котором будет работать данная подсистема. Полный адрес данной подсистемы в локальной сети (к примеру, `localhost:1666` в случае развертывания всех подсистем на одной машине и при использовании `port_number = 1666`) необходим для запуска остальных систем.

Развертывание СХД. Наиболее трудоёмкая операция в развертывании системы - развертывание СХД. Это обусловлено тем, что СХД состоит из нескольких отдельных серверов (которые могут быть развёрнуты на одной вычислительной машине).

Перед началом работы необходимо в файле `/common/sharding_settings.py` установить переменную `SHARDS_COUNT` равной необходимому количеству шардов базы данных (минимум 1).

СХД состоит из следующих серверов (возможно, развёрнутых на одной машине):

- Сервер (несколько, в случае использования шардирования) PostgreSQL, с созданным пользователем `pg_user` и паролем `pg_password`, а так же базой данных `grid_calc_db`. Также необходимо для каждого такого сервера провести его конфигурирование с помощью однократного запуска скрипта `init_db.py`
- Сервис (несколько, в случае использования шардирования) `data_backend`, запущенный командой **`data_backend.py shard_number beacon_address postgres_address port_number`**, где `shard_number` – номер шарда БД, соответствующего данному экземпляру сервиса `data_backend` (0 в случае одного шарда), `beacon_address` – полный адрес сервиса мониторинга, `postgres_address` – полный адрес соответствующего данному шарду экземпляра PostgreSQL, `port_number` – номер порта, на котором будет работать данная подсистема. На сервере, на котором развёрнута данная система, необходима установка пакета `SQLAlchemy`.
- Сервис `sharding_backend`, запущенный командой **`sharding_backend.py beacon_address port_number data1_address, data2_address, ...`**, где `beacon_address` – полный адрес сервиса мониторинга, `port_number` – номер порта, на котором будет работать данная подсистема, `dataN_address` – полный адрес очередного сервиса `data_backend`, соответствующего очередному шарду СХД.

Остальные подсистемы комплекса. Остальные подсистемы комплекса (СХФ, СУ, СУС, ФП, СБН, ФВУ) разворачиваются сходным образом. Для их запуска выполняется команда **`required_backend.py balancer_address port_number`**, где `balancer_address` – полный адрес вида `ipaddress:port`, по которому сервис может получить доступ к сервису мониторинга, а `port_number` – номер порта, на котором будет работать данная подсистема.

На сервере, где разворачивается СУС, необходима установка пакета `SQLAlchemy` и СУБД `SQLite`.

СУ может быть развёрнута в режиме масштабирования серверов сессии. В таком случае сервер СУ должен иметь установленным ПО веб-сервера `Nginx`, настроенного в режиме `round-robin` балансировки запросов к нескольким запущенным экземплярам СУ.

Постановка задачи

Формирование необходимых комплексу файлов. Для постановки задачи системе необходимо сформировать файлы `traits.txt`, содержащий список черт задачи, и архив `archive.tar.gz`, содержащий всё необходимое для запуска задачи.

Список черт задачи должен иметь формат текстового документа, в котором каждая строка соответствует очередной черте, а сама строка представляет собой разделённые одним или несколькими пробелами имя и версию черты.

Архив задачи должен содержать файл с именем `start`, который будет запущен вычислительным узлом для проведения расчётов. Это может быть файл с расширениями `.sh`, `.bat`, `.py`, `.exe`; подразумевается что вычислительный узел должен быть настроен для возможности прямого запуска таких файлов. Требования к соответствующей настройке должны быть указаны посредством механизма черт задачи (к примеру, чертой `'os Linux'` для `.sh` – файлов).

Задача должна сохранять результаты в произвольном формате в каталог `result/`. Все файлы из такого каталога будут заархивированы и отправлены комплексу как результат задачи. Задача должна самостоятельно завершаться по выполнении.

Архив `archive.tar.gz` может быть сформирован с помощью последовательного выполнения команд `tar -cf archive.tar file1 file2` и `gzip archive.tar`, где `file1` и `file2` – файлы, необходимые задаче (к примеру, `start.exe`).

Постановка задачи комплексу. После формирования необходимых файлов, необходимо передать их комплексу. Для этого необходимо, зайдя на веб-интерфейс фронтенда пользователей по адресу, соответствующему сервису `user_frontend`, выполнить следующие действия:

1. Выполнить вход в систему (предварительно зарегистрировавшись). Интерфейс окна входа в систему представлен на рис. 21.
2. Со страницы просмотра состояния о системе (рис. 22) перейти на страницу просмотра поставленных задач (рис. 23).
3. Перейти на страницу создания задачи (рис. 25).
4. С использованием интерфейса создания задачи, передать комплексу архив `archive.tar.gz` и список задач `traits.txt`. Список уже известных комплексу черт можно просмотреть, нажав на соответствующую кнопку в интерфейсе. Пример такого списка приведён на рис. 24. После передачи комплексу файлов, и если не возникло проблем при их передаче, страница примет вид, указанный на рис. 26.
5. Необходимо указать имя задачи (используется для отображения в интерфейсе), число экземпляров задачи (один или более), и максимальное время в секундах, после которого задача будет отображена у предоставляющего узла и переназначена другому.

Имя пользователя:

Пароль:

Вход

Регистрация

Рисунок 21: Интерфейс страницы входа в систему

Тип сервера	Адрес сервера	Состояние сервера
logic_backend	127.0.0.1:1670	Operating normally
shard	127.0.0.1:1668	Operating normally
filesystem	127.0.0.1:1667	Operating normally
user_frontend	127.0.0.1:1671	Operating normally
session_backend	127.0.0.1:1672	Operating normally
database	127.0.0.1:1669	Operating normally
balancer	127.0.0.1:1673	Operating normally
node_frontend	127.0.0.1:1674	Operating normally

Просмотр задач

Выход из системы

Рисунок 22: Интерфейс страницы просмотра информации о системе

- После подтверждения параметров, будет произведена переадресация на страницу просмотра поставленных задач, содержащую информацию о созданной задаче (рис. 27).

Регистрация вычислительного узла

Для регистрации вычислительного узла необходимо создать файл `traits.txt`, сходный с таковым для задач. После этого, необходимо выполнить команду **`client.py client_number node_frontend traits.txt`**, где `client_number` – произвольный ключ, используемый для различия разных экземпляров клиентского ПО, запущенных на одном ВУ, `node_frontend` – полный адрес фрон-тенда вычислительных узлов, `traits.txt` – файл черт данного вычислительного узла.

Название задачи	Дата создания	Черты задачи	Состояния подзадач
<div>Создать новую задачу</div> <div>Просмотр состояния сети</div> <div>Выход из системы</div>			

Рисунок 23: Интерфейс страницы просмотра задач, задач нет

Название черты	Версия черты
test1	123
test2	345b
os	Windows
os_version	8
architecture	AMD64
netFramework	3
netFramework	3.5
netFramework	4
netFramework	4.5

Назад к созданию задачи

Рисунок 24: Интерфейс страницы просмотра известных комплексу черт

Укажите путь к архиву задачи

Выберите файл

Файл не выбран

Отправить

Укажите путь к файлу черт

Выберите файл

Файл не выбран

Отправить

Просмотреть список черт, известных системе

Просмотр задач

Выход из системы

Рисунок 25: Интерфейс страницы создания задачи, до загрузки архива и черт задачи

Архив выбран

Удалить

Список черт выбран

Удалить

Имя задачи

logarithm

Максимальное время простоя

400

Количество экземпляров задачи (минимум 1)

3

Создать задачу

Просмотр задач

Выход из системы

Рисунок 26: Интерфейс страницы создания задачи, после загрузки архива и черт задачи

Название задачи	Дата создания	Черты задачи	Состояния подзадач	
logarithm	Thu, 04 Jun 2015 20:56:45 GMT	os Windows netFramework 4	queued queued queued	Отменить

Создать новую задачу
Просмотр состояния сети
Выход из системы

Рисунок 27: Интерфейс страницы просмотра задач, задача поставлена

Название задачи	Дата создания	Черты задачи	Состояния подзадач	
logarithm	Thu, 04 Jun 2015 20:56:46 GMT	os Windows netFramework 4	queued Выполнена(finished) assigned	Отменить

Создать новую задачу
Просмотр состояния сети
Выход из системы

Рисунок 28: Интерфейс страницы просмотра задач. Одна подзадача находится в очереди, одна выполняется, одна выполнена.

Помимо указанных в файле `traits.txt` черт, вычислительный узел также в качестве черт автоматически добавляет черты `'os'`, `'os_version'` и `'architecture'` с значениями, соответствующими таковым параметрам вычислительного узла.

Получение результатов

Результаты завершённой задачи доступны по ссылке “Результат” в окне просмотра статусов задач, отображаемой в строке статуса подзадачи на рис. 28. Скачанный по данной ссылке архив может быть разархивирован с помощью последовательного выполнения команд `gzip -d archive.tar.gz` и `tar -xf archive.tar`.

4.14. Выводы

В данном разделе были пояснены действия пользователя по взаимодействию с комплексом. Были описаны процедуры развертывания комплекса, постановки комплексу задачи, регистрации вычислительного узла в комплексе и получения результатов задачи, а также отмены задачи.

5. Заключение

Была разработана система кроссплатформенная система распределения вычислительных мощностей. Были выделены и реализованы отдельные сервисы и предусмотрены механизмы горизонтального масштабирования некоторых из них. Разработанная система была описана с помощью диаграмм языка UML, таблиц и других средств. Было проведено тестирование как отдельных компонентов системы, так и всей системы в целом. Были приведены инструкции по взаимодействию с системой в рамках различных реализуемых ей прецедентов.

Система может быть использована в небольших коллективах без доработок. Для работы в больших масштабах рекомендуется объединить функционал некоторых серверов с целью уменьшения накладных расходов на сетевое взаимодействие, а также провести профилирование сервисов с целью выявления проблемных с точки зрения производительности мест.

Список литературы

- [1] Python Software Foundation. *Python Release Python 3.4.3*. 25 февр. 2015. URL: <https://www.python.org/downloads/release/python-343/> (дата обр. 03.06.2015).
- [2] Armin Ronacher и др. *Flask (A Python Microframework) – 0.10.1*. 14 июня 2014. URL: <http://pypi.python.org/packages/source/F/Flask/Flask-0.10.1.tar.gz> (дата обр. 03.06.2015).
- [3] John Paulett, David Aguilar и др. *jsonpickle 0.8.0*. 6 сент. 2014. URL: <https://jsonpickle.github.io/changelog.html#version-0-8-0-september-6-2014> (дата обр. 03.06.2015).
- [4] Python Software Foundation. *hashlib — Secure hashes and message digests*. 25 февр. 2015. URL: <https://docs.python.org/3/library/hashlib.html> (дата обр. 03.06.2015).
- [5] Kenneth Reitz и др. *Requests: HTTP for Humans*. 3 мая 2015. URL: <http://docs.python-requests.org/en/latest/community/updates/#software-updates> (дата обр. 03.06.2015).
- [6] Giampaolo Rodola. *psutil 2.2.1 — cross-platform library for retrieving information on running processes and system utilization in Python*. 25 февр. 2015. URL: <https://github.com/giampaolo/psutil/releases/tag/release-2.2.1> (дата обр. 02.02.2015).
- [7] Python Software Foundation. *subprocess — Subprocess management*. 25 февр. 2015. URL: <https://docs.python.org/3.4/library/subprocess.html> (дата обр. 03.06.2015).
- [8] the SQLAlchemy authors и contributors. *SQLAlchemy: the Database Toolkit for Python*. 7 мая 2015. URL: http://docs.sqlalchemy.org/en/latest/changelog/changelog_10.html#change-1.0.4 (дата обр. 03.06.2015).
- [9] yWorks. *yEd – Graph Editor*. URL: <http://www.yworks.com/en/downloads.html#yEd> (дата обр. 03.06.2015).
- [10] Inkscape collective. *Inkscape – an SVG editing program*. 30 янв. 2015. URL: <https://inkscape.org/ru/news/2015/01/30/inkscape-version-091-is-released/> (дата обр. 03.06.2015).
- [11] The PostgreSQL Global Development Group. *PostgreSQL: The world's most advanced open source database*. 22 мая 2015. URL: <http://www.postgresql.org/docs/9.4/static/release-9-4-2.html> (дата обр. 03.06.2015).

- [12] *SQLite Release 3.8.10.2*. 20 мая 2015. URL: https://www.sqlite.org/releaselog/3_8_10_2.html (дата обр. 04.06.2015).
- [13] GNU. *Tar*. 28 июля 2014. URL: <http://www.gnu.org/software/tar/> (дата обр. 03.06.2015).
- [14] GNU. *Tar for Windows*. 3 окт. 2014. URL: <http://gnuwin32.sourceforge.net/packages/gtar.htm> (дата обр. 03.06.2015).
- [15] Sebastien Luttringer. *gzip 1.6-1*. 20 июня 2013. URL: <https://www.archlinux.org/packages/core/i686/gzip/> (дата обр. 03.06.2015).
- [16] GNU. *gzip 1.3.12 for Windows*. 15 окт. 2013. URL: <http://gnuwin32.sourceforge.net/packages/gzip.htm> (дата обр. 03.06.2015).
- [17] Nginx Inc. *nginx 1.8.0*. 21 апр. 2015. URL: <http://nginx.org/> (дата обр. 05.06.2015).