САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

РАЗРАБОТКА СРЕДЫ ДЛЯ ОБЛАЧНЫХ ВЫЧИСЛЕНИЙ

Дипломная работа студента 545 группы

Чуновкина Фёдора Дмитриевича

Научный руководитель	…… / подпись /	Бондарев А.В.
Рецензент	…… / подпись /	к.ф.-м.н., доцент Иванов А.Н.
“Допустить к защите” заведующий кафедрой	…… / подпись /	д.ф.-м.н., проф. Терехов А.Н.

Санкт-Петербург
2012

SAINT PETERSBURG STATE UNIVERSITY
Mathematics & Mechanics Faculty

Software Engineering Chair

CLOUD COMPUTING INFRASTRUCTURE DEVELOPMENT

by

Fedor, Chunovkin

Graduate paper

Supervisor	……	A. V. Bondarev
Reviewer	……	Ass. Prof. A. N. Ivanov
"Approved by" Head of Department	……	Professor A. N. Terekhov

Saint Petersburg
2012

Оглавление

1. Введение

В современном мире человек повсеместно окружён вычислительными устройствами. На протяжении всего периода развития вычислительной техники, её мощности постоянно возрастают. Согласно закону Мура [50], количество транзисторов, а, следовательно, и мощность вычислительных устройств растёт экспоненциально с течением времени. И даже несмотря на этот неумолимый рост, всегда существовали и до сих пор существуют задачи, которым текущих мощностей одиночных компьютеров оказывается недостаточно. Например, одними из самых трудоёмких и в то же время распространённых и востребованных задач и по сей день остаются задачи потоковой обработки мультимедийной информации. Для эффективного решения таких вычислительно-трудоёмких задач компьютеры объединяются в различные вычислительные сети или кластеры. Огромные распределённые вычислительные кластеры часто именуют собирательным модным словечком – «облачные системы», а вычисления, выполняемые в них, соответственно, - «облачными вычислениями». К большой популярности таких распределённых платформ привели содержащийся в них огромный вычислительный потенциал и удобство для пользователя, который получает необходимые ему вычислительные ресурсы по требованию, избавлен от необходимости разбираться в технологических особенностях реализации и инфраструктуры используемых средств, и платит исключительно за то, чем пользуется. Всё это послужило причиной активного развития данной отрасли информационных технологий.

С другой стороны, огромное количество бытовых вычислительных ресурсов большую часть времени простаивает без дела: в учебных заведениях, офисах, интернет-кафе и других местах. Выгодно было бы иметь простой, быстрый и удобный способ объединять подобные простаивающие ресурсы в «облака» для совместного решения более сложных задач или оказания различных интернет-услуг.

На данный момент существует множество продуктов и технологий, предлагающих те или иные решения в области распараллеливания вычислений. Среди них можно выделить следующие классы решений:

- Закрытые коммерческие «облака-гиганты»
- Платформы для распределённых вычислений
- Специализированные средства для создания распределённых веб-сервисов

Существующие решения чаще всего предоставляют либо традиционную сервис-ориентированную модель, либо возможности по пакетной обработке заданий. При этом задача, отправляемая на решение в «облако», обычно описывается в виде некоторого запроса конечной длины и должна быть полностью известна до непосредственного начала выполнения вычислений, связанных с её решением. Это серьёзно ограничивает спектр задач, доступных для решения в подобных системах. Ярким примером задач, не решаемых в существующих «облаках» общего назначения, является различная потоковая обработка данных, например, обработка «живых» мультимедийных потоков: транскодирование, наложение видео-фильтров, применение алгоритмов распознавания образов в видеопотоке и т.д.

Для специфических задач часто можно найти специализированные решения, возможности которых обычно строго ограничены под решение одной или нескольких конкретных задач. В случае вычислительно-трудоёмких задач такие решения часто предоставляют свои собственные способы для распараллеливания вычислений. В качестве примера таких специализированных продуктов в области обработки «живых» мультимедийных потоков можно назвать Adobe Flash Media Server [4] или его аналог с открытым исходным кодом – Red5 Media Server [66].

Даже в случае задач, доступных для решения в «облаках» общего назначения, заметна ярко выраженная тенденция к узкой специализации отдельных вычислительных узлов. В частности, в сервис-ориентированных системах сервисы обычно представлены обособленными инстанциями, каждая из которых решает строго определённую задачу. Распараллеливание в таком случае достигается путём запуска большого количества экземпляров этих обособленных сервисов, так или иначе взаимодействующих друг с другом в рамках сервисов одного типа. При этом организация взаимодействия сервисов разных типов, то есть решающих разные задачи, сильно затруднена.

Также стоит отметить, что процесс развёртывания практически любой распределённой «облачной» системы не отличается простотой, а потому требует наличия квалифицированного персонала. Данное требование часто является ключевым аргументом против объединения простаивающих ресурсов в единые вычислительные кластера, способные решать куда более сложные задачи, чем любая из одиночных машин.

В летней школе 2009 года на математико-механическом факультете Санкт-Петербургского Государственного Университета (СПбГУ) зародился проект Cloud Observer [20] – совместная разработка двух студентов третьекурсников факультетов мат-меха и ПМ-ПУ (прикладной математики-процессов управления). Изначально проект

предполагал создание распределённой вычислительной среды для организации систем видеонаблюдения и видеоконференций. Тематика проекта обернулась перспективной и многообещающей и стала темой для двух моих курсовых работ. В рамках данной дипломной работы я продолжаю развитие проекта Cloud Observer, решая значительно более общую задачу. Предложено создание децентрализованной распределённой вычислительной сети, построенной на базе сервис-ориентированной архитектуры. Одной из основных идей стало создание такой среды, которая могла бы расширяться за счёт неиспользуемых ресурсов её пользователей и при этом предъявлять слабые требования к характеру решаемых в ней задач. Проект распространяется под свободной лицензией и доступен каждому из открытого репозитория на GoogleCode [20]. Объём автоматически генерируемой из исходных кодов (с помощью утилиты Doxygen [22]) технической документации к проекту составляет на данный момент уже более трёхсот страниц [18]. Также доступна онлайн версия документации в формате HTML [19].

2. Обзор существующих решений

Стремительное развитие информационных технологий за последние несколько десятков лет приучили людей к удобству вычислительной техники, которая сейчас используется повсеместно. Однако вычислительные ресурсы стоят денег, и, столкнувшись с очередной задачей, требующей применения подобных благ цивилизации, современный человек обладает целым рядом альтернативных возможностей.

Наиболее распространённым и традиционным решением в кругу конечных потребителей является использование локальных вычислительных мощностей – приобретение в личную собственность технических средств и использование их по-своему усмотрению для решения различных задач. Из очевидных минусов подобного подхода – малые мощности, которых рано или поздно начнёт не хватать для интересующего класса вычислительных задач.

Другим подходом является использование вычислительных мощностей на удалённых машинах - мощных серверах, обслуживающих тысячи пользователей единовременно. Такой подход более удобен неискушенному пользователю, так как освобождает его от необходимости самому иметь дело с аппаратными средствами, но по-прежнему не защищён от перегрузок самих серверов ввиду ограниченности их мощностей.

В последнее время широкое распространение получили так называемые «облачные вычисления» - предоставление пользователю абстрактных вычислительных мощностей, которые физически распределены на многих удалённых устройствах, образующих так называемое «облако». Для пользователя данный подход значительно удобнее предыдущего, так как он платит не за покупку или пользование конкретными аппаратными средства, а лишь за непосредственно выполненные для него расчёты. Стоит отметить, что в настоящее время термин «облачные вычисления» трактуется разными людьми по-разному. Обобщая, под «облачными системами» будем понимать огромные распределённые платформы. На самом деле, такие платформы совсем не обязательно должны быть вычислительными – существуют, например, многочисленные облачные хранилища данных (Amazon Simple Storage Service (Amazon S3) [8], Dropbox [23], Google Drive [29]).

Отдельного внимания заслуживают ставшие активно развиваться в конце прошлого тысячелетия проекты распределённых вычислений. В отличие от рассмотренных выше

моделей, в которых конечный пользователей (клиент) организовывал или арендовал вычислительные мощности для своих задач, в этих проектах заинтересованное в вычислениях лицо устанавливает сервер и привлекает людей поучаствовать (чаще всего практически на безвозмездной основе) в решении поставленной задачи. Сервер раздаёт подключающимся клиентам-исполнителям фрагменты общей задачи и потом собирает результаты проведённых ими вычислений. Появление такого рода проектов обусловлено предположением, что большую часть времени компьютеры простых пользователей простаивают.

Каждый из рассмотренных подходов требует наличия под него специального программного обеспечения, в котором прослеживаются две противоположные тенденции: предоставление программного обеспечения как продукта или как услуги. Сейчас уже с уверенностью можно сказать, что второй подход стремительно завоёвывает все более стойкую позицию.

Рассмотрев вышеперечисленные модели использования вычислительных ресурсов, можно проследить две кардинально противоположные проблемы:

- Большинство вычислительных ресурсов простаивает.
- Для ряда задач одиночных ресурсов часто бывает недостаточно.

В проекте Cloud Observer [20] рассматривается решение этих проблем путём объединения идей проектов распределённых вычислений и облачной архитектуры. Подробная история развития проекта и эволюции его идей представлена в Приложении 1.

Рассмотрим более подробно различные подходы к построению распределённых вычислительных сетей.

2.1. Закрытые коммерческие «облака-гиганты»

Наиболее популярными являются огромные публичные «облака» от таких гигантов IT-индустрии, как Google, Microsoft и Amazon. Эти продукты доступны каждому, кто готов за них платить, и отличаются очень высоким качеством, надёжностью и быстродействием. Однако все они являются закрытыми и не допускают развёртывание за пределами контролируемой их владельцами территории. Не представляется никакой возможности развернуть подобное «облако» на своём собственном оборудовании.

Продукты данного класса запущены в единственном экземпляре в виде глобального сервиса, доступного пользователям со всего мира. В их архитектуре особое внимание уделено аппаратным особенностям оборудования, тщательно контролируется инфраструктура вычислительных узлов, используется многочисленное низкоуровневое программное обеспечение, написанное специально для данных систем (например, специфические операционные и файловые системы). Всё это обуславливает качественно иной уровень предоставляемых услуг, по сравнению с любыми другими существующими или создаваемыми продуктами.

Яркими представителями этого класса решений являются следующие всемирно известные продукты:

- Amazon Elastic Compute Cloud (Amazon EC2) [7]
- Google App Engine [28]
- Microsoft Windows Azure [44]

2.2. Платформы для распределённых вычислений

Менее популярными, но также хорошо известными являются многочисленные, в большинстве своём свободные (с открытым исходным кодом и свободной лицензией), продукты для развёртывания больших распределённых сетей на массовых аппаратных средствах. Такие вычислительные сети часто называют «частными облаками». Другое название подобных продуктов – grid toolkit. Решения данного класса позволяют объединить множество вычислительных узлов в единую сеть для дальнейшего запуска заданий в ней. Большинство из них нацелены на выполнение пакетных заданий, составленных из запуска различных приложений на узлах и пересылки данных между ними, что обуславливает возможность использования таких средств для решения широкого спектра задач. Важной особенностью подобных решений является объединённая защищённая среда, в которой оказываются выполняемые задания. Доступ к данным и результатам вычислений строго ограничивается в соответствии с настройками системы. Однако такие системы не поддерживают работу с потоковыми данными и не удобны для решения одного из самых распространённых ныне класса задач, а именно предоставления различных повседневных и не очень услуг многочисленным пользователям. Для таких задач больше всего подходят сервис-ориентированные решения,

в то время как рассматриваемые платформы не позволяют организовать поверх себя подобную архитектуру.

В качестве примера платформ для распределённых вычислений, можно привести следующие продукты: Globus [26], UNICORE [74].

2.3. Специализированные средства для создания распределённых веб-сервисов

Также достойны упоминания специализированные продукты для написания распределённых веб-сервисов.

Здесь стоит остановиться поподробнее на вопросе о том, в чём заключается распределённость веб-сервисов. Любой веб-сервис подразумевает множественность, в том смысле, что имеет место запуск многочисленных его экземпляров. Каждый такой экземпляр обладает всеми возможностями сервиса и способен решать соответствующие ему задачи. Однако множественность ещё не делает сервис распределённым. Под распределённостью веб-сервисов будем понимать наличие взаимодействия между их экземплярами. Например, наипростейший сервис-калькулятор, складывающий числа, не является распределённым. А сервис веб-чата – является, так как его экземпляры находятся в активном взаимодействии.

Но вернёмся к специализированным средствам для создания распределённым веб-сервисов. В основном они представлены надстройками над языками программирования или же сами являются языками программирования. В обоих случаях, на уровне языка предоставляются встроенные возможности к распределению данных между экземплярами конкретного сервиса. Однако подобные продукты достаточно слабо развиты и предоставляют весьма скудную функциональность для разработки востребованных сервисов. Невозможность использования существующей кодовой базы и необходимость в большинстве случаев писать веб-сервис «с нуля» отталкивает потенциальных разработчиков от подобных решений и препятствует их дальнейшему развитию.

Представителями данного класса являются следующие продукты: Opa [56], Swarm [73].

3. Постановка задачи

В рамках данной дипломной работы передо мной были поставлены следующие задачи:

1. Спроектировать и реализовать архитектуру кроссплатформенной децентрализованной сервис-ориентированной распределённой вычислительной среды
2. Реализовать возможность решения задач потоковой обработки данных
3. Реализовать возможность общения по произвольным протоколам передачи данных
4. Реализовать возможность взаимодействия сервисов путём обмена произвольными сериализуемыми данными
5. Разработать средства для развёртывания и поддержания работы распределённой среды

4. Архитектура

4.1. Используемые технологии

Первым решаемым в данной работе вопросом стал выбор технологий для разработки распределённой вычислительной среды. В силу необходимости обеспечить кроссплатформенность создаваемого программного продукта, выбор был ограничен языками программирования C/C++, предоставляющими возможности по сборке исходного кода почти под все известные аппаратные платформы, и языком Java, обеспечивающим кроссплатформенность при помощи компиляции в промежуточный байт-код, который исполняется представленной на множестве аппаратных платформ виртуальной машиной.

Язык программирования Java [34] плохо подходит для решения задач, связанных с потоковой обработкой данных. К тому же кроссплатформенность получаемого в данном случае решения зависит от наличия и качества реализации виртуальной машины Java (JVM) под целевую платформу.

Использование языков C/C++ требует от программиста больших временных затрат и квалификации, однако получаемые при этом средства для глубокой системной интеграции и возможности написания высокопроизводительного программного кода, оправдывают их выбор в качестве главного инструмента для решения поставленных передо мною задач.

Более подробные обоснования выбора технологий были сделаны на основе нескольких прототипов, разработанных в рамках проекта Cloud Observer [20]. О них можно прочитать в Приложении 1.

Для ускорения разработки некритичных к производительности фрагментов создаваемого решения активно использовались кроссплатформенные библиотеки Boost [12], предоставляющие огромное количество программных средств для решения многих задач общего назначения, как то работа с файловой системой, сетью, потоками исполнения, сериализацией/десериализацией данных, а также для подгрузки динамических библиотек.

В созданных для демонстрации сервисах также использовались следующие библиотеки:

- FFmpeg [24] – преобразования мультимедийных форматов и контейнеров

- OpenAL [57] – захват и воспроизведение звука
- OpenCV [58] – захват и воспроизведение видео, обработка изображений
- OpenSSL [59] – шифрование и кодирование данных

Унифицированный пользовательский веб-интерфейс распределённой вычислительной среды создавался с использованием языков HTML5 [32], CSS [17] и JavaScript [35], а также платформы Adobe Flash [5].

4.2. Организация распределённой среды

Распределённая вычислительная среда представляет собой множество взаимодействующих друг с другом элементарных составляющих – узлов.

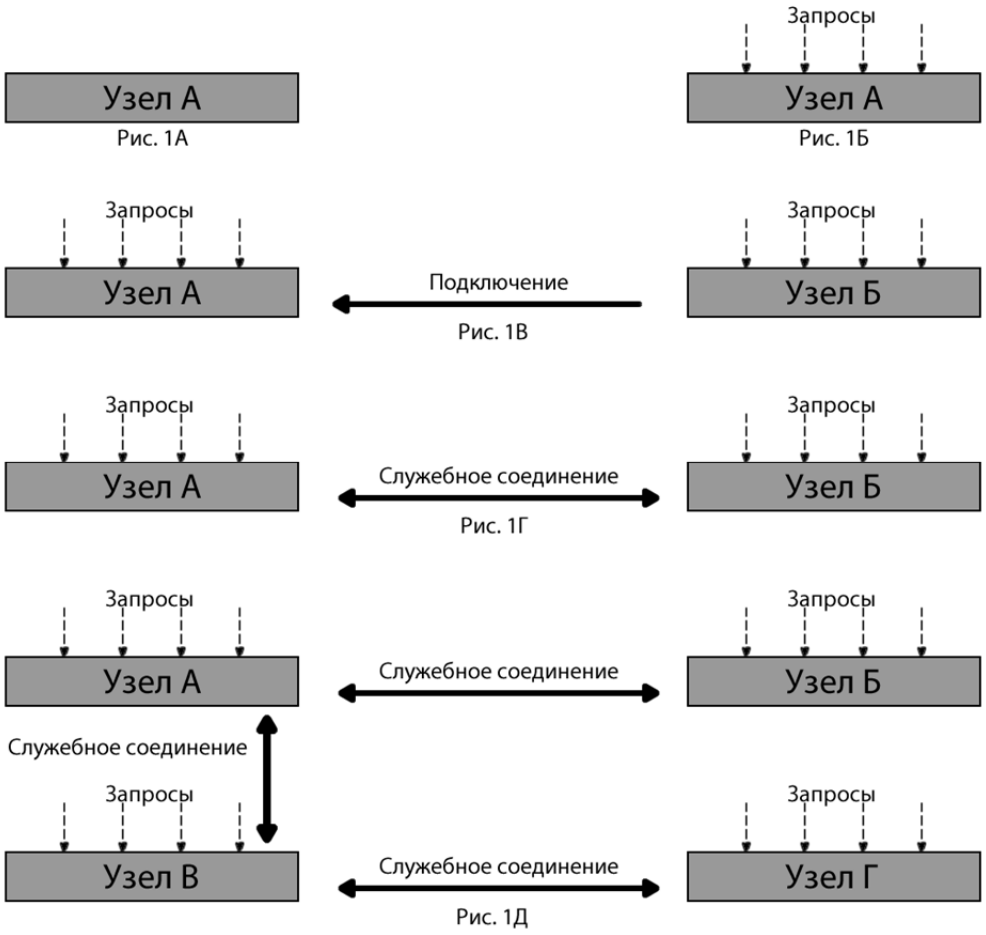

Рисунок 1. Процесс соединения узлов

Создание распределённой среды начинается с запуска одиночного узла (Рис. 1А), уже в этот момент она способна к обслуживаю запросов пользователей (Рис. 1Б). Запуск следующего узла происходит с указанием ему адреса первого узла, что позволяет

произвести подключение нового узла к уже имеющейся части среды (Рис. 1В). Данный процесс приводит к образованию двустороннего служебного соединения между узлами (Рис. 1Г). При запуске каждого последующего узла ему также передаётся адрес одного из уже запущенных узлов (такой узел будем в дальнейшем называть узлом-коннектором для данного узла), что приводит к образованию очередного служебного соединения (Рис. 1Д).

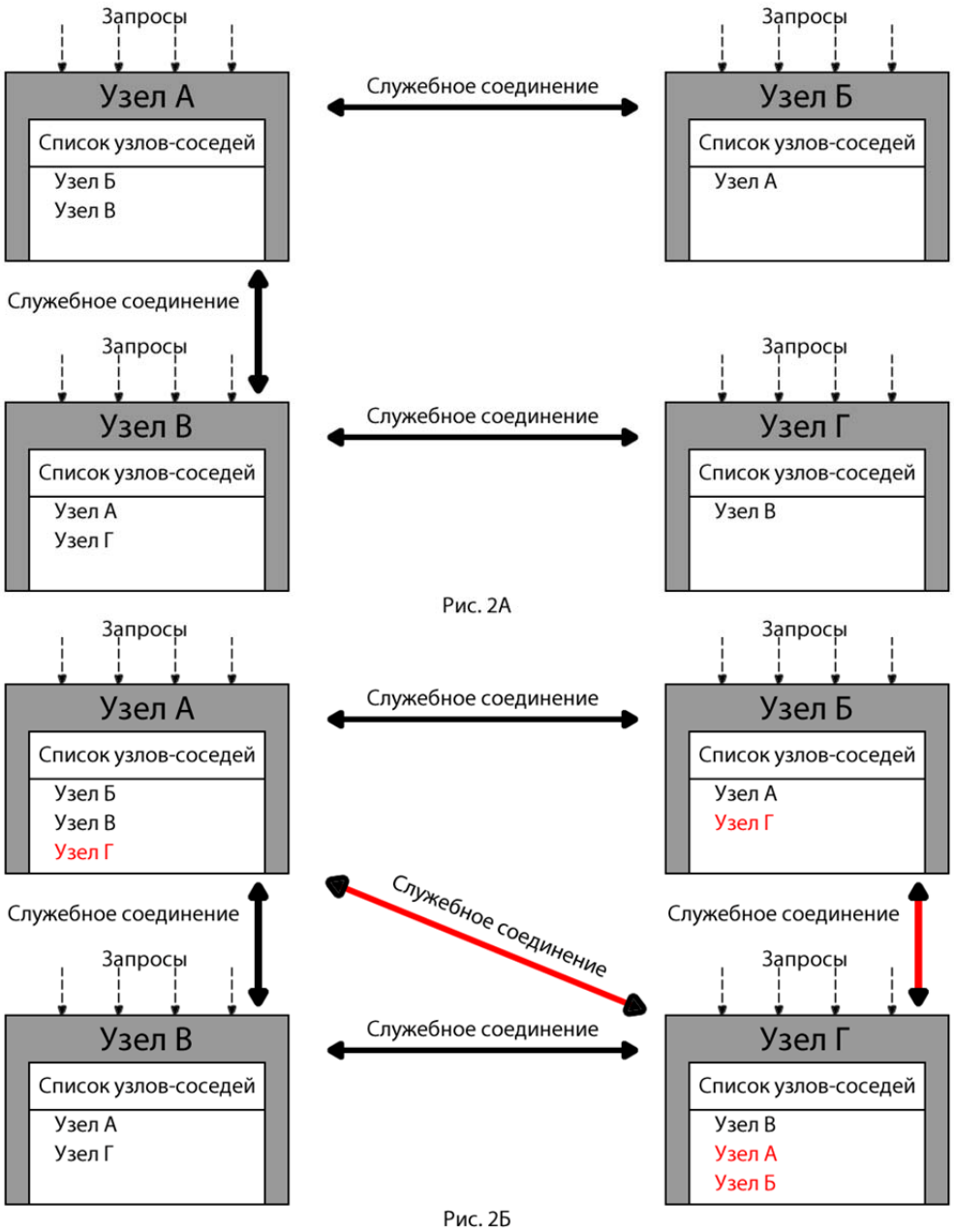

Рисунок 2. Списки узлов-соседей

Каждый узел содержит в себе список известных ему «соседних» узлов, в котором он сохраняет адреса узлов, с которыми были установлены служебные соединения

(Рис. 2А). С течением времени, узлы обмениваются данными из своих списков узлов-соседей, что приводит к образованию всё большего числа служебных соединений и увеличению связности распределённой среды (Рис. 2Б).

Организованная таким образом вычислительная сеть является децентрализованной, так как отдельные её узлы сохраняют потенциальную самостоятельность и ничего не знают о полной топологии сети, количестве вычислительных узлов в ней и других общих данных о распределённой системе в целом. Узлам доступна лишь информация, получаемая от узлов-соседей, с которыми постоянно поддерживаются служебные соединения.

4.3. Узел распределённой системы

Узел системы представлен написанным на языке программирования C++ кроссплатформенным приложением Cloud Server. Каждый узел может принимать запросы пользователей, в том числе и от других узлов этого или другого экземпляра среды Cloud Observer. Необходимым требованием для взаимодействия узлов (установлению служебных соединений и обмену запросами от пользователей) является лишь наличие у них одинакового средства для коммуникации (в простейшем случае - стека протоколов TCP/IP и возможности прямого общения по нему). Получаемые от пользователей запросы узел может обслуживать самостоятельно или перенаправлять известным ему узлам-соседям.

Главной особенностью архитектуры узла распределённой системы является разделение выполняемого им полезного вычислительного функционала на множество элементов – сервисов. Узел является контейнером для набора работающих в нём сервисов. Группировка сервисов в узлах позволяет существенно сократить накладные расходы на взаимодействие сервисов по сравнению с традиционными сервис-ориентированными архитектурами, в которых сами сервисы играют роль узлов распределённой системы. Основным фактором, позволяющим снизить накладные расходы, является взаимодействие сервисов через оперативную память, минуя затратные процессы копирования информации. Иначе говоря, сервисы обмениваются лишь указателями на данные в памяти, но не самими обрабатываемыми данными. Наибольший выигрыш подобное архитектурное решение обеспечивает в задачах, связанных с потоковой обработкой

информации и используется практически во всех мультимедийных фреймворках (например, в DirectShow [38]).

Сервис представляет собой программный модуль (класс), содержащий некоторую вычислительную логику, используемую для решения определённых задач. Сервисы хранятся в динамических библиотеках и подгружаются по мере необходимости одним из модулей узла. Архитектура сервиса спроектирована с учётом возможности решения в нём широкого спектра вычислительных задач.

Узел выполняет сбор различной статистики о загруженности ресурсов оборудования, на котором он исполняется. Обмен собранными данными с узлами-соседями позволяет каждому узлу осуществлять локальную (в рамках известных ему узлов-соседей) балансировку нагрузки распределённой вычислительной сети.

В конфигурации каждого узла определяется произвольное количество транспортов – входных точек в узел, по которым он получает запросы. Поддержка различных типов транспортов осуществляется путём подгрузки динамических библиотек-расширений. В качестве примера типов транспортов можно назвать открытые на прослушку (listen) сокеты или конвейеры (pipes).

Рассмотрим более подробно архитектуру узла распределённой системы (Рис. 3).

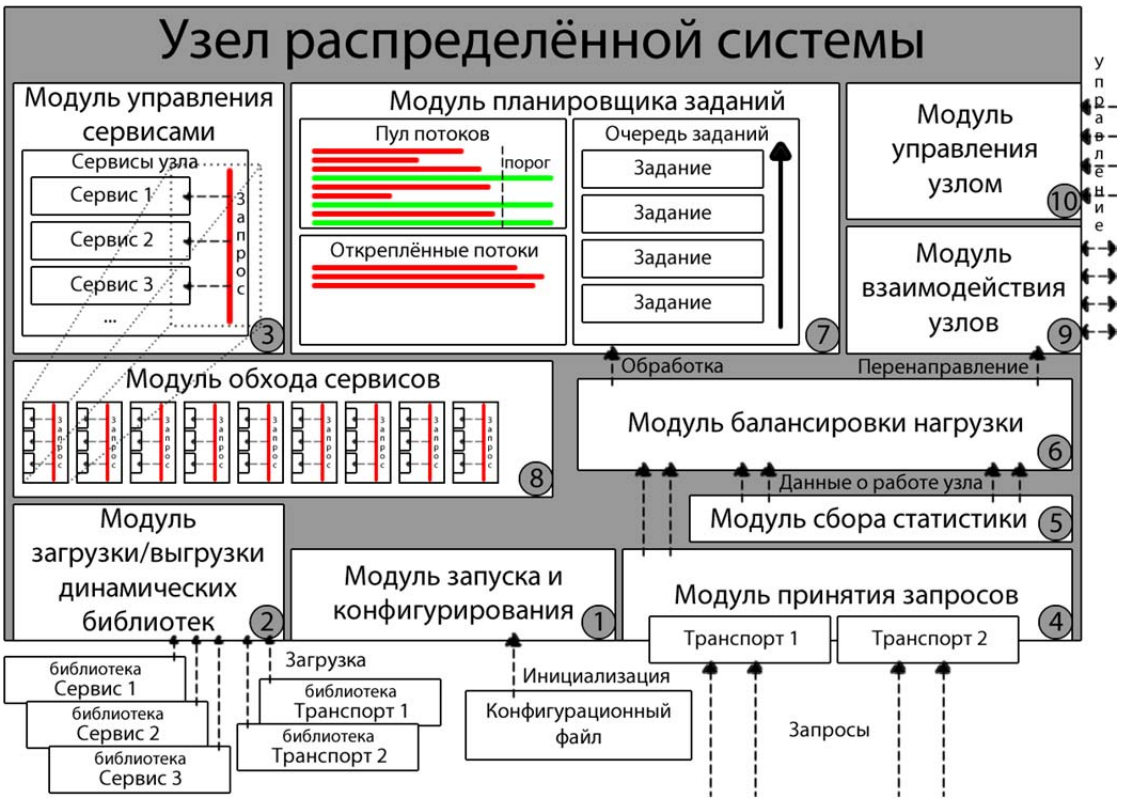

Рисунок 3. Архитектура узла распределённой системы

Узел состоит из следующих функциональных блоков:

1. Модуль запуска и конфигурирования
2. Модуль загрузки/выгрузки динамических библиотек
3. Модуль управления сервисами
4. Модуль принятия запросов
5. Модуль сбора статистики
6. Модуль балансировки нагрузки
7. Модуль планировщика заданий
8. Модуль обхода сервисов
9. Модуль взаимодействия узлов
10. Модуль управления узлом

4.3.1. Модуль запуска и конфигурирования

В процессе запуска узла производится чтение конфигурационного файла, путь к которому передаётся ему в качестве аргумента командной строки. В случае отсутствия данного аргумента производится попытка прочесть настройки из местоположения по-умолчанию, а именно из файла `config.xml` или `config.json` (предпочтение отдаётся формату XML), расположенного в каталоге, содержащем исполняемый бинарный файл узла. При невозможности загрузить конфигурационный файл происходит аварийное завершение работы узла.

Узел конфигурируется в соответствии с прочитанной информацией. Конфигурационный файл представляет собой файл в формате XML или JSON. В нём содержится ряд общих настроек: имя узла, адрес узла-коннектора, множество транспортов - входных точек для получения узлом запросов - и некоторых других параметров. Главной же информацией, хранимой в конфигурационном файле, является список сервисов, которые необходимо запустить на узле, и их настройки.

Каждый сервис описывается в виде имени библиотеки и конкретного класса, в котором он содержится, краткого текстового комментария, типа сервиса (публичный или приватный), приоритета сервиса и набора произвольных индивидуальных настроек, представленных в виде коллекции типа ключ-значение, в которой в качестве ключа и значения могут быть использованы произвольные строки.

Тип сервиса определяет его видимость для окружающего мира. Список и информация о публичных сервисах узла может быть получена при помощи специального зарезервированного служебного запроса (HTTP-GET запроса на URL `/services.json` узла). Приватные сервисы скрыты от внешнего мира, но участвует в процессе обхода сервисов.

Приоритет сервиса представляет собой натуральное число, определяющее порядок обхода сервисов на данном узле. Этот порядок очень важен для корректной работы описанного ниже модуля обхода сервисов.

4.3.2. Модуль загрузки/выгрузки динамических библиотек

Модуль загрузки/выгрузки динамических библиотек начинает работу сразу после завершения конфигурирования узла. Первым делом, он пытается загрузить динамические библиотеки сервисов, которые требуется запустить. Затем производится загрузка библиотек, отвечающих за типы транспортов, указанных в конфигурационном файле.

При невозможности найти или загрузить ту или иную библиотеку, информация об этом событии заносится в журнал, но работа узла не прерывается.

В дальнейшем модуль загрузки/выгрузки динамических библиотек продолжает работу в фоновом режиме, будучи готовым загрузить дополнительные библиотеки или выгрузить уже загруженные в ответ на изменение конфигурации узла.

4.3.3. Модуль управления сервисами

Модуль управления сервисами хранит список загруженных на узле экземпляров сервисов, их конфигурацию, коллекции выполняемых в данный момент каждым сервисом заданий и информацию об их состоянии.

Модуль управления сервисами начинает работу после выполнения первичной загрузки динамических библиотек и производит запуск (создаёт экземпляр класса сервиса и вызывает у него функцию «`start`») всех загруженных сервисов. Каждый сервис конфигурируется индивидуальными настройками (функцией «`apply_config`»), которые читаются из конфигурационного файла.

Модуль производит мониторинг функционирования сервисов на протяжении всего времени работы узла.

4.3.4. Модуль принятия запросов

Получив управление сразу после запуска всех необходимых сервисов, модуль принятия запросов начинает ожидание запросов со всех указанных транспортов.

При получении входящего соединения, оно оборачивается в структуру так называемого транспортного канала. Конкретные типы транспортов, подгружаемые в виде динамических библиотек-плагинов, обеспечивают возможности преобразования произвольного протокола передачи данных в структуру транспортного канала и обратно (Рис. 4).

Рисунок 4. Транспортные каналы с различными транспортами

Транспортный канал представляет собой пару непрерывных потоков данных (входящий - от пользователя к исполнителю – и исходящий – в обратном направлении), а также коллекцию произвольных сериализуемых «общих данных», обеспечивающих возможности для активного взаимодействия между сервисами, в том числе сервисами разных типов (Рис. 5). Оба потока буферизируются, предоставляя произвольный доступ к содержащимся в них данным. Созданный транспортный канал отправляется в модуль балансировки нагрузки.

Рисунок 5. Структура транспортного канала

В качестве наиболее примитивного примера конкретного типа транспорта можно привести TCP-сокеты. Преобразование TCP-сокета в структуру транспортного канала заключается лишь в реализации буферизации прочитанных из сокета и отправленных в него данных. Коллекция общих данных в данном случае остаётся пустой.

Более сложным примером транспорта является HTTP-транспорт. В процессе преобразования входящего соединения в транспортный канал, такой транспорт выполняет разбор HTTP-запроса, после чего помещает его в коллекцию общих данных в сериализованном виде. В процессе обратного преобразования, происходит поиск сформированного сервисами HTTP-ответа в коллекции общих данных, его десериализация и отправка.

Процесс ожидания новых соединений продолжается до момента остановки узла.

4.3.5. Модуль сбора статистики

Модуль сбора статистики производит мониторинг аппаратных ресурсов, на которых исполняется приложение узла. Ведётся наблюдение за процентом загрузки центрального процессора и отдельных его ядер, количеством доступной оперативной памяти, свободным местом на жёстких дисках, а также загруженностью и пропускной способностью сети.

Производится усреднение полученных данных для анализа работы узла за длительные промежутки времени и построения прогнозов по доступности тех или иных ресурсов в будущем.

4.3.6. Модуль балансировки нагрузки

Модуль балансировки нагрузки содержит средства для комплексной оценки загруженности данного узла и его узлов-соседей на базе информации, получаемой из модулей сбора статистики этих узлов. Руководствуясь данной информацией, модуль производит балансировку нагрузки, распределяя поступающие на данный узел запросы между самим узлом и узлами-соседями. Если узел принимает решение об обслуживании запроса на данном узле, происходит формирование нового задания по обработке входящего запроса, которое направляется в модуль планировщика заданий. Если узел принимает решение о передачи запроса на один из узлов-соседей, производится выбор наиболее подходящего их них и переброс транспортного канала запроса при помощи модуля взаимодействия узлов.

4.3.7. Модуль планировщика заданий

Модуль планировщика заданий предназначен для управления потоками исполнения, в которых выполняется обслуживание запросов, оставленных на обработку на данном узле модулем балансировки нагрузки.

Планировщик заданий содержит внутри себя пул потоков и очередь заданий, переданных ему на выполнение. Размер пула потоков ограничен. Точное количество потоков в нём задаётся в конфигурационном файле. Изначально все потоки в пуле помечены как свободные (зелёные полоски на Рис. 3).

При получении нового задания, планировщик проверяет наличие свободных потоков в пуле. При наличии свободного потока, задание запускается на выполнение в этом потоке, а поток помечается как занятый (красные полоски на Рис. 3). Иначе задание помещается в очередь заданий.

По завершению выполнения задания, поток возвращается в пул и помечается как свободный. Планировщик заданий проверяет наличие ожидающих заданий в очереди, и при наличии таковых предоставляет освободившийся поток первому заданию в ней.

Планировщик заданий замеряет время выполнения каждого задания (длина полосок на Рис. 3). Если задание выполняется дольше некоторого заданного в конфигурационном файле интервала времени («порога»), исполняющий это задание поток

отсоединяется от пула потоков и продолжает выполняться отдельно (такой поток будем называть «откреплённым»). При этом создаётся новый поток, который помещается на освободившееся в пуле потоков место.

Подобная схема позволяет узлу обслуживать длинные запросы (например, запросы по обработке «живого» видео) и при этом не препятствовать обслуживанию множества «обычных» короткоживущих заданий. В то же время, наличие пула потоков ограничивает количество одновременно обслуживаемых «обычных» заданий, что существенно снижает вероятность перегрузки узла.

4.3.8. Модуль обхода сервисов

Модуль обхода сервисов является самым интересным компонентом архитектуры узла распределённой вычислительной среды. Каждое задание по обслуживанию запроса, получив свободный поток исполнения из пула потоков планировщика заданий, проходит через сложный процесс совместной обработки сервисами узла.

Транспортный канал, созданный в момент принятия запроса в соответствующем модуле, последовательно проходит по отсортированному списку сервисов данного узла. Каждый сервис при этом опрашивается на предмет того, что он может сделать с данным запросом (с точки зрения реализации происходит вызов функции «`service_check`» объекта сервиса). Возможными вариантами ответа являются:

- обслужить запрос полностью («`accept`»)
- сопутствовать обслуживанию запроса («`assist`»)
- отвергнуть запрос, как неподходящий («`reject`»)

В случае готовности сервиса полностью обслужить запрос, данный сервис объявляется «терминальным» для запроса. В случае положительного ответа сервиса на возможность полного или частичного обслуживания запроса, модуль обхода сервисов вызывает функцию «`service_call`» данного сервиса и тем самым задействует вычислительную логику, заложенную в сервисе. Если сервис не является терминальным, процесс обхода продолжается со следующим сервисом из отсортированного по приоритетам списка сервисов узла.

Если ни один из сервисов не стал терминальным, запрос отправляется в модуль взаимодействия узлов для перенаправления на один из узлов-соседей.

Вызовы функций «service_check» и «service_call» сервисов могут приводить к изменению общих данных, содержащихся в передаваемом им транспортном канале запроса. Функция «service_call» также может видоизменять и сам запрос, модифицируя данные в буферизируемых непрерывных потоках данных транспортного канала.

Каждый сервис может в процессе работы над запросом потребовать создания дополнительных транспортных каналов для осуществления вспомогательных запросов на другие узлы сети.

Предложенный алгоритм взаимодействия сервисов способствует формированию, так называемых, «каскадов сервисов», в которых множество сервисов совместно способствуют обслуживанию запроса.

4.3.9. Модуль взаимодействия узлов

Основной задачей модуля взаимодействия узлов является пересылка запросов между узлами-соседями. Пересылка выполняется при помощи так называемого прокси-модуля (от англ. proxy). Транспортный канал сериализуется и направляется на узел назначения.

Процесс сериализации транспортного канала требует дополнительных пояснений. Как уже было описано выше, транспортный канал состоит из двух буферизированных непрерывных потоков данных и коллекции «общих данных». Общие данные всегда хранятся в сериализованном состоянии, а потому их передача по сети не представляет трудностей. Иначе дело обстоит с потоками данных. Буферизированные части потоков элементарным образом сериализуются и передаются на узел назначения. Также пересылаются позиции, в которых находились оба потока данных в момент пересылки. Прокси-модуль продолжает поддерживать двустороннее соединение с узлом назначения до полного обслуживания запроса. Все данные, получаемые во входящий поток, передаются на узел назначения. Все данные, записываемые в исходящий поток узлом назначения, передаются обратно.

Такая схема вместе с концепцией транспортных каналов делает перенаправленный запрос абсолютно неотличимым от исходного запроса с точки зрения узла назначения. Узел назначения воспринимает ситуацию так же, как если бы запрос изначально был

направлен в его модуль принятия запросов пользователем и не проходил процесса пересылки между модулями взаимодействия узлов. Информация о перенаправлении, тем не менее, сохраняется в коллекции общих данных транспортного канала запроса, что позволяет модулям взаимодействия узлов избегать ситуаций повторного посещения запросом одного и того же узла, то есть зацикливания пути следования запроса по графу узлов.

4.3.10. Модуль управления узлом

Модуль управления узлом предназначен для его динамического конфигурирования. Конфигурирование осуществляется через интерфейс командной строки или через TCP-сокеты с использованием утилиты `telnet`. Возможность конфигурирования узла через `telnet` по-умолчанию выключена и требует явного включения в конфигурационном файле узла. Возможно внесение изменений как в общие настройки узла (название, транспорты и т.д.), так и в настройки конкретных сервисов. В последнем случае, настройки сервиса задаются так же, как в конфигурационном файле, - парами строк ключ-значение. Модуль управления узлом позволяет приостанавливать и возобновлять работу сервисов, а также добавлять новые сервисы и удалять имеющиеся. Возможно комплексное конфигурирование узла путём подачи ему дополнительных конфигурационных файлов. Структура таких файлов аналогична структуре первичного конфигурационного файла, используемого при запуске узла.

4.4. Интерфейс базового сервиса

Каждый сервис представляет собой класс, наследующийся от абстрактного (pure virtual) класса базового сервиса. В базовом сервисе объявлены функции для запуска («`start`»), остановки («`stop`») и конфигурирования («`apply_config`») сервиса, а также две основополагающие абстрактные функции:

- «`service_check`» - осуществляет анализ транспортного канала запроса на предмет возможности его полного или частичного обслуживания данным сервисом

- «service_call» - осуществляет непосредственно полную или частичную обработку запросов данным сервисом

Классы наследники обязаны предоставлять реализацию функций «service_check» и «service_call». Переопределение остальных функций базового сервиса опционально и зависит от потребностей и возможностей конкретных сервисов.

4.5. Вспомогательные модули

В процессе реализации описанной архитектуры возникла необходимость в разработке ряда вспомогательного функционала, который был объединён в программные библиотеки.

В модуле «extension_utils» были собраны средства для работы с подключаемыми модулями (динамическими библиотеками), из которых производится загрузка сервисов и типов транспортов и которые обеспечивают расширяемость функционала системы.

Модуль «threading_utils» предоставляет удобные возможности для работы с пулами потоков и управлению группами вычислительных заданий.

Модуль «http_utils» обеспечивает возможности для коммуникации по протоколу HTTP - одному из самых распространённых протоколов прикладного уровня стека TCP/IP.

В модуле «log_utils» представлены инструменты для журналирования процесса работы системы. Поддержаны возможности ведения множества журналов, расстановки приоритетов, сохранения информации в файлы и вывода в стандартные потоки вывода и ошибок (stdout и stderr соответственно).

Модуль «general_utils» содержит ряд прочих прикладных алгоритмов, использованных при реализации распределённой сервис-ориентированной архитектуры системы.

Представленные модули также являются перспективными с точки зрения расширения сервисной базы проекта, так как весьма упрощают разработку дополнительных сервисов.

4.6. Унифицированный пользовательский веб-интерфейс

В современном мире наибольшую популярность в сервис-ориентированных системах имеют сервисы, работающие по протоколу HTTP. Основным средством общения пользователя с HTTP-сервисами являются веб-браузеры. В связи с этим, на языках HTML5 [32], CSS [17] и JavaScript [35], а также с активным использованием библиотеки jQuery [36], был разработан унифицированный пользовательский веб-интерфейс для доступа к таким сервисам.

Наличие пользовательского интерфейса не является обязательным, так как общение с сервисами может осуществляться в автоматическом режиме. Однако представленный унифицированный интерфейс как минимум будет полезен разработчикам сервисов для тестирования их работы.

4.7. Развёртывание и поддержание работы узлов

Для развёртывания и поддержания работы узлов распределённой среды был написан специальный скрипт под командную оболочку Bash [14] – Cloud Observer Loader. Он позволяет произвести сборку узла системы и всех необходимых для неё библиотек. Возможно разворачивание локальной среды для разработки самой системы или сервисов под неё. Также скрипт позволяет задействовать «режим поддержки» (Рис. 6), который следит за работой узла и поддерживает его в актуальном состоянии, выполняя автоматическое обновление до последней версии, доступной в системе контроля версий.

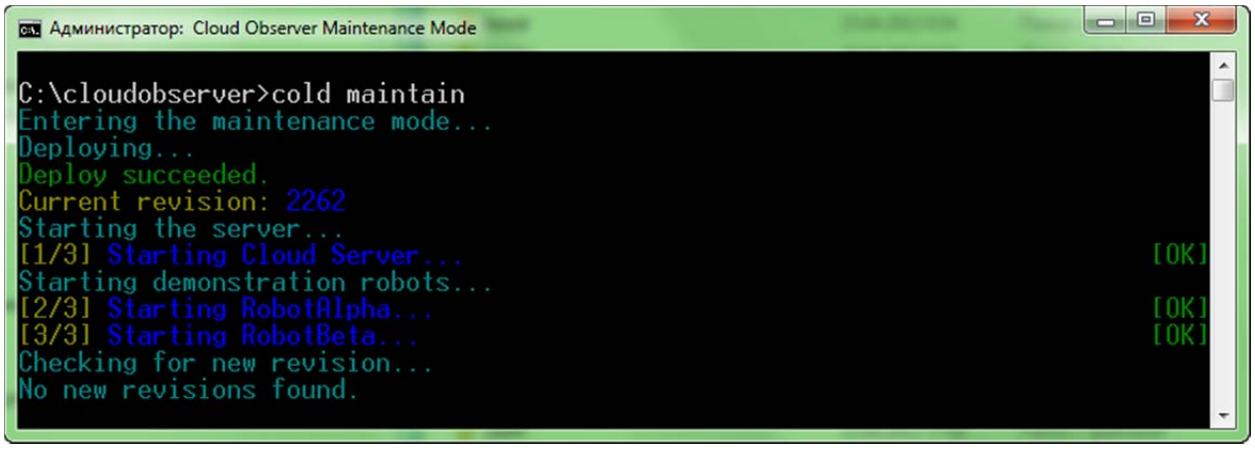

Рисунок 6. Режим поддержки в действии

`cold` (Cloud Observer LoaDer) - это универсальное средство управления системой Cloud Observer на всех поддерживаемых платформах (Microsoft Windows [46], Apple Mac OS X [10], дистрибутивы Linux).

`cold.bat` - это небольшая обёртка над `cold` для его запуска из под командной строки Windows. Обёртка написана на языке командного интерпретатора Windows и, частично, на языке программирования VBScript [75] (в основном для загрузки и разархивирования zip-архива с окружением MinGW [47]). Обёртка также в состоянии самостоятельно загружать основной скрипт из системы контроля версий. Работа скрипта-обёртки заключается в загрузке и установке локальной копии окружения MinGW [47] и последующей передаче управления командному интерпретатору Bash [14], который начинает исполнять основной скрипт. Все аргументы командной строки передаются основному скрипту, что обеспечивает одинаковый интерфейс для работы со скриптом на всех поддерживаемых платформах.

Вся деятельность скрипта ведётся в той папке, в которой он расположен, вне зависимости от того, откуда производится запуск. При первом использовании, рекомендуется создать пустую папку и положить скрипт под нужную платформу в неё. Данную папку будем далее называть рабочим каталогом. Необходимо избегать длинных путей и нестандартных символов в них (в том числе русских букв, так что, например, если имя пользователя в операционной системе семейства Windows их содержит, не стоит класть скрипт в "Мои документы", на "Рабочий стол" и т.д.). В процессе работы скрипт использует абсолютные пути, так что перемещение папки после работы скрипта не рекомендуется и может приводить к пересборке некоторых компонентов системы.

Для работы скриптов необходимо наличие компилятора. На операционных системах семейства Windows поддержана работа с любой коммерческой редакцией Microsoft Visual Studio [43] 2008 или 2010, а также с бесплатными редакциями Microsoft Visual C++ Express Edition 2008 и 2010. Все они определяются в системе автоматически. Если скрипт находит несколько версий, то предпочтение отдаётся более новым и полным. На остальных системах требуется наличие набора компиляторов GCC [27] с поддержкой языка программирования C++.

В `Приложении 2` приведено подробное описание команд и опций созданного средства развёртывания и поддержания работы узлов.

5. Демонстрационные сервисы

Далее рассмотрим несколько демонстрационных сервисов, реализованных для созданной распределённой вычислительной среды.

5.1. File Service (FS)

File Service является одним из самых простых HTTP-сервисов в системе. В его обязанности входят предоставление и контроль доступа к определённым фрагментам файловой системы узла, задаваемым в настройках сервиса. В частности, File Service может быть использован в качестве простейшего веб-сервера.

В сервисе реализован функционал по индексации файлов в подконтрольных ему фрагментах файловой системы узла и кэшированию небольших часто запрашиваемых файлов.

В практических ситуациях данный сервис обладает наименьшим приоритетом среди всех сервисов узла и, таким образом, получает возможность обслуживания запроса только в том случае, если ни один из предыдущих сервисов не смог обслужить запрос. Если и данный сервис не может найти подходящий файл для обслуживания запроса, запрос будет передан на один из узлов-соседей.

Возможно дальнейшее усовершенствование сервиса путём объединения кэшей экземпляров сервисов на узлах-соседях при помощи модуля взаимодействия сервисов.

5.2. User Accounts Service (UAS)

User Accounts Service обеспечивает контроль доступа к остальным сервисам узла и предоставляет средства для управления пользователями. Система транспортов позволяет сервису поддерживать авторизацию и регистрацию пользователей не только по протоколу HTTP, но и по собственному простейшему алгоритму общения через TCP-сокеты.

В практических ситуациях данный сервис обычно запускается на каждом узле системы и имеет наивысший приоритет.

Сервис предъявляет особые требования к синхронизированной работе узлов. В частности, в процессе реализации сервиса встали проблемы возможной регистрации одного и того же имени пользователя на двух удалённых узлах сети. Гарантированное исключение подобной ситуации требует возможности синхронизации информации о пользователях на всех узлах сети, что практически не реализуемо в децентрализованной архитектуре. Поэтому имеет смысл говорить лишь о минимизации вероятности подобной ситуации, для обеспечения которой были предложены активные средства по распределению информации по сети. В частности, сервис постоянно осуществляет синхронизации данных о пользователях с узлами-соседями. В связи с выполнением подобной активной синхронизации на каждом из узлов сети, процесс распространения информации о пользователях существенно ускоряется.

5.3. User Files Service (UFS)

User Files Service предоставляет возможности защищённого файлового хранилища и средства для управления личными файлами пользователей. В рамках распределённой системы сервис обеспечивает избыточность хранимых данных. Файлы пользователей копируются на соседние узлы в фоновом режиме, что решает сразу две задачи: увеличивает надёжность хранения данных и обеспечивает большую их доступность.

5.4. Observer Service (OS)

Observer Service является, пожалуй, самым сложным из представленных сервисов. Именно на примере данного сервиса демонстрируется возможность решения задач, связанных с потоковой обработкой данных. Сервис предназначен для осуществления обработки мультимедийных потоков в режиме реального времени. Конкретно, сервис использует возможности библиотеки FFmpeg [24] для транскодирования потоков: смены контейнеров и форматов сжатия аудио и видео составляющих. Простейшей задачей для данного сервиса является трансляция мультимедийных потоков без изменений.

5.5. Image Renderer Service (IRS)

Image Renderer Service предоставляет средства для обработки изображений. Сервис базируется на библиотеке OpenCV [58] и потому обладает обширными возможностями.

Данный сервис является ярким примером ситуации, в которой распараллеливание не требует никаких дополнительных затрат. Запуск экземпляров такого сервиса на нескольких узлах не требует никаких дополнительных расходов на синхронизацию.

5.6. Run Service (RS)

Run Service предоставляет возможности для удалённого запуска приложений. В настройках сервиса на каждом запускающем его узле указывается список приложений, доступных для удалённого запуска, а также аргументы, которые могут быть переданы им.

Данный сервис тесно интегрирован с User Files Service, что позволяет запускаемым приложениям работать с личными файлами пользователей, хранимыми UFS-сервисом.

Run Service в связке с User Files Service позволяет организовывать в созданной системе пакетную обработку заданий, составленных из запуска различных приложений на узлах и пересылки данных между ними, что позволяет говорить о реализации функционала, предоставляемого рассмотренными во второй части обзора существующих решений платформами для распределённых вычислений (grid toolkit).

6. Демонстрационная система

Тестирование созданной распределённой вычислительной среды проводилось на кластере из высокопроизводительных серверов факультета прикладной математики-процессов управления СПбГУ. Для анализа работы системы было развёрнуто средство контроля загрузки кластера Ganglia [25].

На базе созданных демонстрационных сервисов был протестирован функционал среды в качестве веб-сервера, системы учёта пользователей и предоставления им функций файлового хранилища, веб-сервиса обработки изображений, а также сервиса видеоконференций.

Особое внимание было уделено тестированию работы системы на примере сервиса видеоконференций, организованного с помощью множества экземпляров сервисов типа Observer Service. Вещание мультимедийных потоков осуществлялось с помощью созданного вспомогательного приложения Cloud Client, позволяющего производить захват аудио/видео данных с микрофона и веб-камеры. В приложение также были заложены возможности по генерации аудио/видео данных (режим робота). Генерируемый видео-поток представляет собой электронные часы, отсчитывающие текущее время на вещающем их компьютере, а также отображающие прошедшее с начала вещания время. Генерируемый аудио-поток представлен чистым звуковым тоном в 440 Герц. Было проведено нагрузочное тестирование со 100 одновременно вещающими с разных компьютеров демонстрационными мультимедийными потоками (роботами).

В Приложении 3 представлены снимки экрана фрагментов работы с демонстрационной системой.

7. Заключение

В результате проделанной работы были получены следующие результаты:

Была спроектирована и реализована архитектура кроссплатформенной децентрализованной сервис-ориентированной распределённой вычислительной среды. Конкретно, на языке программирования C++ было написано кроссплатформенное приложение Cloud Server, представляющее узел распределённой сети, реализованы алгоритмы подключения узлов друг к другу и их последующего взаимодействия по обработке запросов пользователей.

Были предоставлены возможности решения в «облаке» задач, связанных с потоковой обработкой данных, проведена демонстрация данной возможности на примере обработки «живых» мультимедийных потоков (Observer Service).

Была спроектирована и реализована концепция транспортов и транспортных каналов, позволившая поддержать возможности общения по произвольным протоколам передачи данных. Были представлены примеры сервисов, работающих по протоколам TCP и HTTP.

Для обеспечения возможности активного взаимодействия сервисов, в том числе сервисов различных видов, решающих разные задачи, была спроектирована и реализована система передача произвольных сериализованных данных между сервисами, что привело к архитектуре, в которой различные сервисы совместно способствуют решению поставленных перед ними задач.

Для развёртывания и поддержания работы узлов распределённой системы было разработано специальное средство – Cloud Observer Loader.

Полученная система была продемонстрирована в качестве веб-сервера, системы учёта пользователей и предоставления им функций файлового хранилища, веб-сервиса обработки изображений, а также сервиса видеоконференций.

8. Приложение 1. История проекта Cloud Observer

Проект Cloud Observer [20] зародился 1 июля 2009 года в рамках летней школы, проводимой на математико-механическом факультете Санкт-Петербургского Государственного Университета (СПбГУ). Изначально проект предполагал создание распределённой вычислительной сети для организации систем видеонаблюдения и видеоконференций.

С самого своего появления на свет проект разрабатывается двумя студентами факультетов мат-меха и ПМ-ПУ – мной, Чуновкиным Фёдором Дмитриевичем, выпускником кафедры системного программирования математико-механического факультета СПбГУ, и Якушкиным Олегом Олеговичем, являющимся на момент написания данного текста студентом четвёртого курса кафедры компьютерного моделирования и многопроцессорных систем факультета прикладной математики-процессов управления СПбГУ. На момент начала проекта мы оба только-только стали студентами третьего курса университета и поступили на соответствующие кафедры.

За три года проект Cloud Observer обзавёлся солидной историей и к моменту начала данной дипломной работы насчитывал уже 4 версии. Все они были прототипами различных аспектов предложенной идеи, на базе которых оценивалась её перспективность, делались судьбоносные архитектурные решения, изучались и сравнивались существующие решения, происходил процесс выбора технологий, наиболее подходящих для реализации, а также накапливался опыт работы по разработке программных продуктов вообще и конкретно в данной сфере информационных технологий. Всё это привело к становлению идеи проекта в том виде, в котором она представлена в качестве обзора в данной работе.

Условно, эволюцию проекта и его идей до момента начала данной дипломной работы можно разбить на три этапа. На первом этапе (версии 0.1 и 0.2) был создан и продемонстрирован первый прототип системы, показана работа прототипа в качестве очень примитивного сервера видеонаблюдения. Второй этап (версия 0.3) заключался в создании прототипа распределённой системы, первоначальному продумыванию её архитектуры. Третий этап (версия 0.4) был посвящён тщательному исследованию существующих решений в данной предметной области, обдуманному выбору технологий для создания производительной системы и дальнейшему развитию архитектуры распределённой сети.

Основной идей проекта с самого его зарождения является объединение принципов, используемых в проектах распределённых вычислений, с облачной архитектурой.

На данный момент, пройдя в процессе своего развития множество переосмыслений, идеи проекта Cloud Observer заключаются в следующем:

Предложено создание децентрализованной распределённой сети, построенной на базе сервис-ориентированной архитектуры. Предполагается создать среду, которую легко можно было бы развернуть на массовых потребительских вычислительных мощностях, объединив их, таким образом, в «облако», позволяющее проводить в себе различные вычислительные расчеты. При этом планируется предъявлять слабые требования к характеру решаемых задач и, в частности, предоставлять возможности по работе с потоковыми данными в ситуациях, когда задание не может быть полностью сформулировано к моменту начала обработки данных. В архитектуру решения закладываются обширные возможности для взаимодействия элементарных блоков системы – сервисов, в том числе сервисов различных типов, решающих разные задачи. Также, одной из ключевых идей является создание такой среды, которая могла бы расширяться за счёт неиспользуемых ресурсов своих пользователей.

От облачной архитектуры предлагаемое решение заимствует уровень абстракции аппаратных средств от конечных пользователей, в том смысле, что пользователь не знает, где именно и как будет выполнено его задание. В то же время предполагается реализовать возможность поддержания среды вычислительными ресурсами самих пользователей, по аналогии с проектами распределённых вычислений.

8.1. Первый прототип (Cloud Observer v0.1)

Первый прототип, созданный в рамках проекта Cloud Observer [20], представлял собой стационарный сервер для вещания и просмотра последовательностей изображений, формирующих иллюзию видео-потока. Прототип был написан с обширным использованием множества технологий и продуктов компании Microsoft: Microsoft .NET Framework [37], Microsoft Windows Communication Foundation (WCF) [45], Microsoft SQL Server [42], Microsoft Language-Integrated Query (LINQ) [40], Microsoft Silverlight [41] и других.

Несмотря на заложенную в прототип модульную архитектуру, он не являлся распределённой системой. Модули системы представляли собой отдельные исполняемые файлы. Процессы, запускаемые из этих файлов, взаимодействовали друг с другом путём отправки сообщений посредством технологии WCF с использованием HTTP протокола.

Для автоматизации процесса запуска сервера был создан набор скриптов под интерпретатор командной строки Windows.

Захват последовательности изображений осуществлялся из нескольких клиентских приложений, написанных на платформах Adobe Flash [5] и Adobe AIR [3]. Была реализована возможность захвата изображений с веб-камеры и со встроенного в AIR-приложение полнофункционального веб-браузера на базе движка WebKit [76]. Также была реализована возможность ретрансляции последовательности изображений с удалённых веб-камер, вещающий в интернет в формате Motion JPEG (MJPEG) [51].

Просмотр последовательности изображений осуществлялся через предоставляемый сервером веб-интерфейс, реализованный в виде приложения на Silverlight.

Сервер предоставлял базовые возможности по учёту пользователей, а именно их регистрацию и авторизацию. Также были представлены встроенные возможности по записи последовательности изображений на жёсткий диск и сохранению информации о них в базе данных Microsoft SQL Server.

Первый прототип был написан очень быстро и вдохновил своих создателей на дальнейшее развитие идеи создания распределённого сервера для организации систем видеонаблюдения и видеоконференций.

8.2. Усовершенствованный прототип (Cloud Observer v0.2)

Через некоторое время после создания первого прототипа, он был практически полностью переписан и усовершенствован. Была значительно переработана кодовая база, внесены существенные изменения в архитектуру программного решения. Система получила простейшие возможности для размещения взаимодействующих модулей на разных компьютерах. Так было положено начало распределённости в проекте Cloud Observer.

На данном этапе не было представлено никаких средств для балансировки нагрузки между вычислительными узлами. К тому же сами вычислительные узлы были крайне не

стабильны и часто самопроизвольно отключались. Большинство проблем было обусловлено использованием технологии WCF, которая, как впоследствии выяснилось, совершенно не предназначена для обработки большого количества запросов в единицу времени. В данном же прототипе, каждый кадр последовательности изображений передавался в виде отдельного WCF-сообщения. Огромные накладные расходы на передачу сообщений перегружали систему и выводили её из строя.

В рамках усовершенствованного прототипа также весьма успешно велись работы по захвату и передаче звукового ряда вместе с последовательностью изображений. На базе технологии DirectSound [39] было создано клиентское приложение, вещающее звук на сервер. Звук передавался небольшими порциями, длиной примерно в половину секунды. Каждая порция передавалась в виде отдельного WCF-сообщения. Итогом данной работы стала версия 0.2b, поддерживающая передачу как видео (в виде последовательности изображений), так и звуковой информации.

8.3. Прототип распределённой сети (Cloud Observer v0.3)

После реализации пробных прототипов, реализующих распределённость на очень примитивном уровне, встал вопрос о проектировании более серьёзной распределённой системы, способной к саморегуляции и балансировке нагрузки.

8.3.1. Концепция

Распределённая сеть на данном этапе проектировалась из расчёта на максимальную гибкость и расширяемость. Увеличить вычислительную мощность можно было простейшим подключением любого персонального компьютера или сервера, однако на стадии прототипа было решено ограничиться использованием операционной системы Microsoft Windows [46] с установленным .NET Framework [37].

Среда предоставляла открытые интерфейсы для создания различных клиентских приложений, а также для расширения возможностей функциональных блоков самой системы. Посредством библиотек-плагинов, разработкой которых могли заниматься сторонние разработчики, было возможно добавление в систему поддержки различных

физических устройств хранения данных, систем управления базами данных, а также наделение системы любой интересующей вычислительной функциональностью.

В качестве клиентов системы могли выступать сторонние приложения под любые платформы и устройства, имеющие доступ к сети и поддерживающие HTTP-протокол передачи данных. Вокруг системы было возможно построение различных окружений, реализующих удобный пользовательский интерфейс к вычислительным мощностям. При этом система не была завязана на одно единственное окружение - один экземпляр работающей системы мог обслуживать множество различных окружений с различными клиентскими приложениями. Это обеспечивало возможность на базе развёрнутой среды предоставлять различные комплексные услуги пользователям.

Для обеспечения универсальности упор был сделан на поддержку массового аппаратного обеспечения. В отличие от продуктов, работающих с профессиональным оборудованием, данная среда не могла полагаться на надёжность и защищённость используемых аппаратных ресурсов, а потому реализовывала поддержку этих качеств на программном уровне.

8.3.2. Архитектура

Среда представляла собой распределённую совокупность веб-сервисов шести различных типов, взаимодействующих друг с другом посредством обычного стека протоколов TCP/IP (Рис. 7). Запущенная система сама управляла добавлением и удалением сервисов, в соответствии с количеством поступающих запросов и своими нуждами. Каждый из типов сервисов мог присутствовать в системе в любом количестве экземпляров.

Сердцем системы являлись сервисы типа Cloud Controller (в дальнейшем CC), управляющие ресурсами системы и следящие за её целостностью и правильным функционированием. Для повышения надёжности системы была реализована возможность создания множества CC-сервисов, как дублирующих, так и образующих древовидную структуру. CC-сервисы управляли созданием, восстановлением и удалением всех сервисов в системе, в том числе других CC-сервисов.

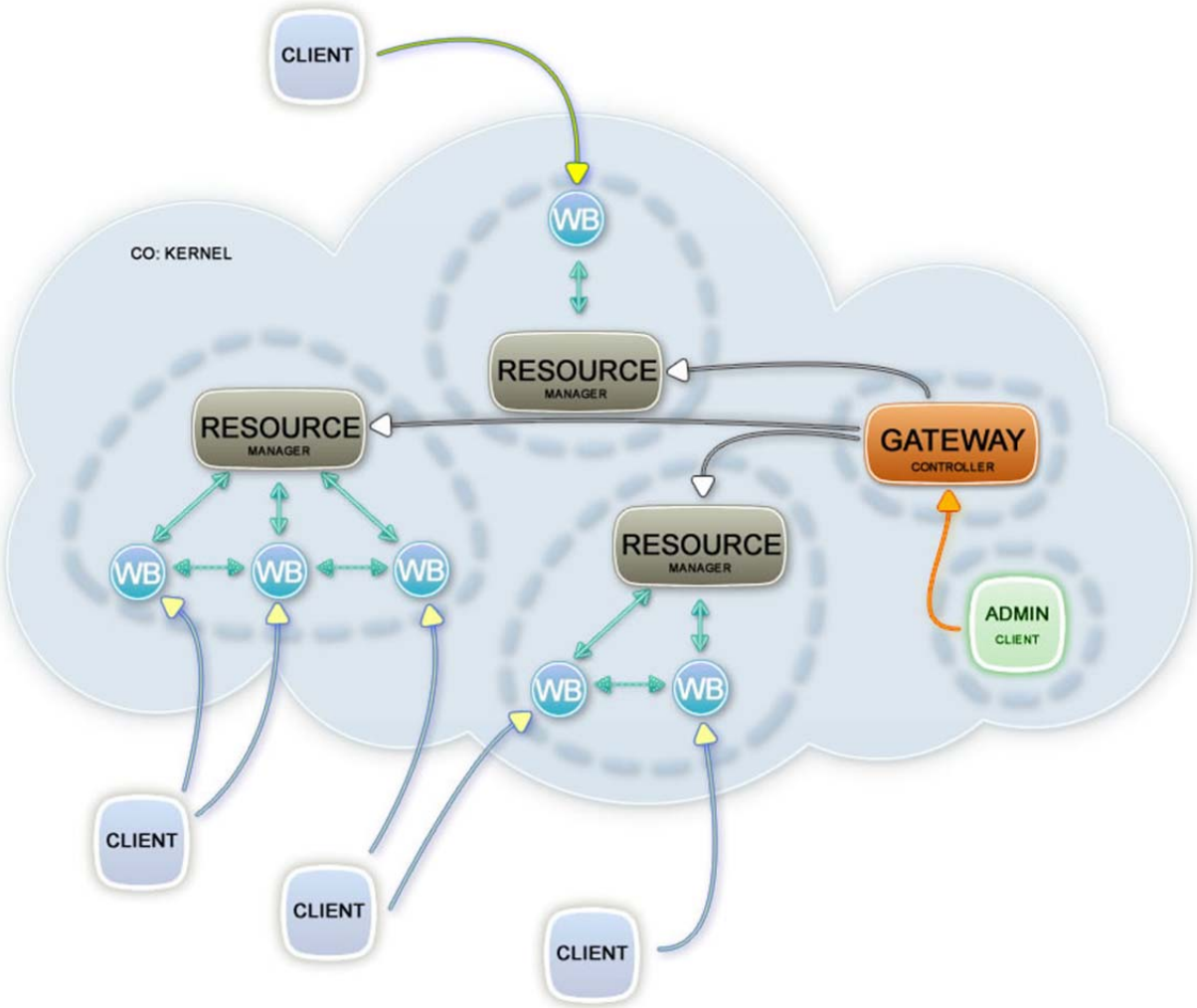

Рисунок 7. Архитектура Cloud Observer v0.3

На каждой машине (персональном компьютере или сервере), подключаемой к системе, в том числе и на машине, инициировавшей создание экземпляра системы, запускался сервис типа `Resource Manager` (RM), управляющий ресурсами данной конкретной машины. Одна машина могла быть подключена сразу к нескольким экземплярам системы. В отличие от CC-сервисов, которые были привязаны к конкретному экземпляру системы, RM-сервис руководил распределением ресурсов конкретной машины между всеми системами, к которым она была подключена. Таким образом, на каждой машине могло быть запущено не более одного RM-сервиса, а во всей системе количество RM-сервисов совпадало с количеством подключённых машин.

Для журналирования работы системы, хранения информации о текущем состоянии, а также структурированных пользовательских данных использовались сервисы типа `Database Manager` (DM), которые обеспечивали доступ к конкретным системам управлениям базами данных. Расширяемость DM-сервисов подключением

дополнительных библиотек-плагинов позволяла потенциально использовать в системе любые СУБД и обеспечивать при этом единый интерфейс доступа к ним.

Хранение информации в системе обеспечивалось сервисами типа `Storage Manager` (SM), которые управляли конкретными физическими хранилищами данных. Расширяемость SM-сервисов подключением дополнительных библиотек-плагинов позволяла потенциально использовать в системе любые физические хранилища данных и обеспечивать при этом единый интерфейс доступа к ним.

Непосредственные вычисления внутри системы осуществлялись сервисами типа `Work Block` (WB). Функциональность WB-сервисов расширялась подключением дополнительных библиотек-плагинов. Всем клиентским приложениям для выполнения необходимых им операций системой выделялись один или более WB-сервисов, причём дальнейшее общение между клиентами и WB-сервисами происходило напрямую. Сама система также могла использовать WB-сервисы для своих вычислительных задач.

Подключение к системе клиентских приложений происходило через сервисы типа `Gateway` (GW), адреса которых представляли собой входные точки в систему. GW-сервисы реализовывали единый стандартизированный интерфейс для общения клиентов с системой. Используя этот интерфейс, наряду с интерфейсом к WB-сервисам, сторонние разработчики могли писать свои собственные клиентские приложения для системы.

8.3.3. Технологии

В свете обширности идеи на этапе создания данного прототипа было решено ограничиться готовыми технологиями организации сетевого взаимодействия, а конкретно уже использованной ранее технологией WCF, входящей в Microsoft .NET Framework. Это сузило круг машин, которые могли использоваться для поддержания работы системы, до имеющих операционную систему Microsoft Windows с установленным .NET Framework 3.5, что, однако, предполагалось достаточным для оценки перспективности идеи универсальной распределённой вычислительной сети. Также рассматривалась возможность оптимизации кода прототипа распределённой сети для запуска отдельных компонент под Mono [49], что в дальнейшем (в версии 0.4) было использовано для создания первого кроссплатформенного решения.

8.3.4. Предметная область для демонстрации

Хотя ключевой целью при создании прототипа являлось написание непосредственно среды для построения распределённых вычислительных сетей, трудно было бы демонстрировать систему в действии без выбора некоторой предметной области, в которой, в качестве примера, можно было бы показать и проанализировать достигнутые результаты.

На сегодняшний день на рынке всё большее место занимает обработка и хранение мультимедиа содержимого. Например, видеохостинги, видеоконференции, системы видеонаблюдения, и т.д. Объём видеоданных в мире очень быстро растёт, растут и затраты на их хранение и обработку. Применение в данной сфере предложенных идей распределённых облачных вычислений выглядит хорошей и перспективной задумкой.

Подобный выбор предметной области привёл к необходимости изучения и использования некоторых дополнительных технологий, в частности технологии Microsoft DirectShow [38], на базе которой реализованы почти все мультимедийные составляющие операционной системы Windows. Использование DirectShow позволило уйти от передачи последовательности изображений и впервые начать работу с настоящими видео-потоками.

8.3.5. Проблемы и их решения

Несмотря на предполагаемую простоту, на первых стадиях работы возникали проблемы с использованием технологии WCF. Много усилий было потрачено на изучение разнообразных типов привязок (binding), определяющих используемые для сетевого взаимодействия протоколы. В конце концов, выбор был сделан в пользу наиболее простого и самого распространённого типа привязки (так называемый BasicHttpBinding), работающего поверх POST и GET запросов протокола HTTP. Как оказалось, многие заманчивые возможности WCF крайне слабо поддержаны сторонними производителями. Так, создание простейшего RIA (Rich Internet Application) клиента уже становилось совсем не тривиальной задачей с каким-либо другим типом привязки. Рассматривалось использование таких популярных технологий как Adobe Flash [5] и Microsoft Silverlight [41], уже зарекомендовавших себя в рамках первых двух прототипов.

Следующая проблема встала при первых шагах в выбранную предметную область и заключалась в передаче непрерывных потоков мультимедиа данных. Направленность WCF на общение при помощи сообщений оказалась очень неудобна в данной ситуации. Первым решением был переход на использование протокола HTTP вручную, который сразу же решил проблему передачи данных от «облака» (некоторого WB-сервиса) к клиентам системы, но оставил открытым вопрос вещания данных с клиента (например, с веб-камеры) в «облако». Следующим действием был шаг на ещё более низкий уровень – использование лишь транспортного протокола TCP с ручной реализацией логики общения поверх установленного соединения. Это решение окончательно разрешило проблему. Было решено оставить обратную совместимость передачи данных от «облака» к клиенту по протоколу HTTP, что позволило проигрывать входящие потоки стандартными средствами самих клиентов.

Наконец, следующей осознанной проблемой было соединение компьютеров, находящихся за преобразователями сетевых адресов (NAT, Network Address Translator [54]). Хотя некоторые исследования в данном направлении были проведены (протокол STUN [69]), данный вопрос остаётся до конца не разрешённым и по сей день.

8.3.6. Итоги

Версия 0.3 внесла наибольший вклад в развитие идей проекта. Именно в ней была впервые представлена распределённая вычислительная сеть и были получены первые результаты по её быстродействию.

Результатом проделанной работы стал прототип программного продукта, позволявший развернуть среду распределённых облачных вычислений. Среда запускалась на любом компьютере с операционной системой Microsoft Windows и установленным .NET Framework. Поддерживалось подключение неограниченного количества дополнительных удалённых компьютеров-помощников. Управляющие блоки среды обладали примитивными способностями к распределению нагрузки между доступными вычислительными ресурсами.

Работоспособность запущенной среды была продемонстрирована на примере обработки мультимедийной информации. Конкретно, была реализована возможность захвата аудио и видео потоков со сжатием в форматы MP3 [52] и FLV [6] соответственно клиентами и трансляции их через «облако» другим клиентам. Подобный функционал

позволял построить, например, распределённый сервис видеоконференций в облаке – достаточно было лишь снабдить запущенную среду некоторым окружением (веб-сервером), предоставляющим соответствующий пользовательский интерфейс. Такое примитивное окружение для распределённой среды также было создано.

Предложенная идея распределённой вычислительной сети показала себя вполне жизнеспособной, однако выбор технологий в угоду скорости разработки несколько смазал общее впечатление. Использование высокоуровневых технологий, таких как WCF, приводит к весьма существенным затратам производительности. Первостепенной задачей по дальнейшей работе над проектом стал перенос логики, продемонстрированной данным прототипом, на более быстрые низкоуровневые технологии.

8.4. Прототип узла (Cloud Observer v0.4)

Вскоре стало ясно, что громоздкая распределённая система, созданная на предыдущем этапе, с трудом поддаётся переносу на низкоуровневые технологии. В связи с чем, было решено начать переход с более простого программного продукта.

Довольно быстро система из множества сервисов различных типов была урезана до одного приложения, представляющего стационарный сервер, но обеспечивающая весь реализованный на данный момент функционал, за исключением, собственно возможностей к распределённой работе. Предполагалось развитие данного приложения как одиночного узла для будущей распределённой системы.

Простота реализации полученного приложения позволила быстро оптимизировать его для работы под Mono [49], что впервые привело проект Cloud Observer на операционные системы, отличные от Microsoft Windows.

Однако проблемы с производительностью оставались даже в рамках простого одиночного узла, в связи с чем началось обширное исследование технологий и существующих решений в данной предметной области.

8.4.1. Выбор направления развития

Серия различных прототипов, разработанная ранее, показала состоятельность предложенной идеи организации распределённых вычислений на потребительском оборудовании, однако выбор технологий в угоду скорости разработки сильно сказался на его производительности. Ключевым недостатком прототипа являлись существенные проблемы со скоростью работы, обусловленные использование большого числа высокоуровневых технологий, и в частности Microsoft Windows Communication Foundation (WCF). Другим существенным недостатком являлась возможность работы только на компьютерах под управлением операционных систем семейства Windows с установленным Microsoft .NET Framework версии 3.5 или выше. Таким образом, следующим логическим шагом на пути дальнейшего развития проекта должна была стать реализации системы на более низком уровне. Конкретно, встал вопрос о выборе технологий для создания действующей и конкурентно-способной реализации.

Первым же делом был поднят вопрос об отказе от платформы Microsoft .NET и переходу к кроссплатформенному в том или ином смысле решению.

8.4.2. Исследование кроссплатформенности

Термин «кроссплатформенный» обычно не вызывает видимых разночтений у людей, связанных с IT-индустрией. Однако попытки углубиться в его смысл раскрывают множество различных трактовок. Дословный перевод составного слова «кроссплатформенный» даёт нам значение «работающий на нескольких платформах». Но о каких платформах идёт речь? Существуют аппаратные платформы, чаще всего группируемые по названию архитектуры микропроцессоров, в них используемых, – ARM [11], Sparc [72], Intel x86 [33] и многие другие. Существуют программные платформы, которые сами можно условно разделить на высокоуровневые и низкоуровневые. Низкоуровневые программные платформы – это, как правило, операционные системы: Microsoft Windows [46], Apple Mac OS X [10], различные Unix-подобные операционные системы. Высокоуровневые программные платформы более точно описываются термином «фреймовик» или «фреймворк» (англ. framework). Примерам таких платформ могут быть Java [34], Microsoft .NET [37], Qt [65] и другие.

Так какой же продукт имеет право называться «кроссплатформенным»?

Многие программисты считают программы, написанные на языке программирования Java, кроссплатформенными. Их можно понять. Однажды написанный на этом языке программный код, может быть запущен на множестве различных «платформ» - от «платформы» требуется лишь предоставить реализацию виртуальной машины Java (Java Virtual Machine, JVM). Однако, имея ввиду широкий спектр возможных трактовок термина «платформа», мы видим, что программы на Java не являются кроссплатформенными. Любая программа на языке Java работает только на одной единственной платформе – на платформе JVM. Сама JVM как программный продукт является кроссплатформенной, так как существуют её реализации под множество различных аппаратных платформ.

Аналогично, .NET, будучи изначально задуманным корпорацией Microsoft как высокоуровневая программная платформа, на данный момент может считаться кроссплатформенным в связи с существованием проекта Mono [49] – свободной реализации .NET Framework, работающей на множестве аппаратных платформ. Однако любая программа, написанная под .NET (и в частности прототип распределённой среды для облачных вычислений из предыдущего этапа), даже будучи адаптированной для работы в среде Mono, не будет являться кроссплатформенной – она работает на единственной платформе, на платформе Microsoft .NET.

Программы на любом из интерпретируемых языков (Perl [60], PHP [61], Python [64] и другие) в силу своей интерпретируемости также нельзя назвать кроссплатформенными, хотя сами интерпретаторы этих языков, безусловно, таковыми являются.

Итак, что же мы можем считать «истинной» кроссплатформенностью? Я считаю, что истинно кроссплатформенная программа – это такая программа, которую можно собрать из исходных текстов на нескольких аппаратных платформах. И это приводит нас к двум почти самым популярным (в различных опросах по-разному) языкам программирования, тесно связанным друг с другом – C и C++.

Желание обеспечить «истинную» (в обозначенном выше смысле) кроссплатформенность оставляет не много выбора.

8.4.3. C vs C++

Оба этих языка традиционно ассоциируется с чем-то низкоуровневым, хотя сами бесспорно являются языками программирования высокого уровня. Такие ассоциации не случайны – именно эти языки используются для написания внутренних блоков практических всех современных операционных систем.

В процессе сравнения языков, отчётливо прорисовались две крайности: скорость (C) и удобство разработки (C++).

Язык C++, будучи объектно-ориентированным, предоставляет значительно большие возможности программисту, но в то же время обладает значительно более сложным (или правильнее сказать «громоздким») синтаксисом. Язык C, в силу своей лаконичности вместе с большими возможностями, традиционно используется в ситуациях, когда нужна максимальная производительность, хотя программисту приходится прикладывать значительно больше усилий и внимания для получения аналогичного (по сравнению с языком C++) функционала. Быстрее языка программирования C будут, разве что, программы, написанные на языке ассемблера, но такие программы уже никак не будут являться кроссплатформенными.

Весомым плюсом обоих языков является огромное число различных программных библиотек, написанных под них. Существенное значение имеет и объем открытого исходного кода на C/C++, которого гораздо больше, чем на каком-либо другом языке [55], что делает его прочным фундаментом для разработки. Подавляющее большинство этого кода «истинно» кроссплатформенно – исходные коды могут быть скачаны и собраны под множество различных аппаратных платформ.

Сегодня созданию приложений на языках низкого уровня препятствуют сложности в разработке и трудности в масштабируемости. Однако разработка приложений на языке программирования C++ с выходом нового стандарта C++11 значительно упростилась [15]. Это подтверждается заявлениями представителей IT-корпораций о «ренессансе» языка и его продвижением в рамках профессионального сообщества [16].

В связи с намерением реализации распределённой системы, важным фактором также стали доступные разработчику средства организации сетевого взаимодействия. Существует три вида библиотек программирования, которые предоставляют различные возможности для реализации компонентов сетевых архитектур на C++:

- интегрированные в операционные системы

- предоставляющие кроссплатформенные абстракции для работы с сетью: 0MQ [1], Adaptive Communication Environment (ACE) [2], Boost [12], Qt [65]
- вспомогательные библиотеки для организации работы веб приложений: Mongoose [48], Wt [77]

Очень весомым аргументом стали библиотеки Boost [12]. Будучи «истинно» кроссплатформенными, они уже много лет существенно влияют на развитие всего языка C++. Многие их части введены соответствующим комитетом в Стандарт C++, многие другие находятся в стадии заявки на включение. Библиотеки Boost часто противопоставляются фреймворку Qt [65] – он также предоставляет множество различного функционала общего назначения. Однако, в отличие от Qt, который мы уже отнесли к платформам (взять хотя бы необходимость использования специального компилятора), библиотеки Boost не требуют наличия каких-либо сторонних продуктов для сборки под новую платформу. В частности, Boost предоставляет свою систему сборки Boost.Build [13], которая, тем не менее, сама собирается из исходных кодов. Библиотеки Boost предоставляют унифицированный доступ к огромному перечню различного функционала общего назначения, оставаясь при этом «истинно» кроссплатформенными.

Также немаловажным фактором выбора языка C++ стали его возможности глубокой системной интеграции.

Итак, выбор был сделан в пользу языка программирования C++ с активным использованием библиотек Boost.

8.4.4. Сервис-ориентированная архитектура

В последнее десятилетие активно развивалась «сервероцентрическая» бизнес-модель «программного обеспечения по требованию» (SaaS) [71]. Одной из причин такого вектора развития была потребность разделения процесса исполнения задач и предоставления результатов на растущем спектре пользовательских устройств.

Для реализации SaaS архитектуры применяется модульный подход к разработке программного обеспечения SOA (сервис-ориентированная архитектура), основанный на «слабо связанных» (англ. loose coupling) заменяемых компонентах, оснащённых стандартизированными интерфейсами взаимодействия [68]. Все чаще, на ряду с классическими службами, взаимодействующими по протоколу SOAP (простой протокол

доступа к объектам) [70], встречаются RESTful сервисы, соответствующие ограничениям стиля «архитектуры передачи состояния представления» [67].

Существует множество реализаций SOA, основанных на технологиях разработки и платформах, предоставляющих высокий уровень абстракции, таких как Java [34], Microsoft .NET [37] и т. д. Архитектура подобных решений, как правило, представляет собой множество слоев. Например: промежуточный байт-код, виртуальная машина, исполняемый код для конкретного целевого процессора.

Наличие прослойки виртуальной машины позволило многим крупным компаниям создать распределенные PaaS [62] (платформы в виде услуги) для удаленного исполнения ориентированных на них служб.

В целях обеспечения безопасности и контроля крупных публичных PaaS систем, исполнение отдельных сервисов, специфически заточенных под конкретное оборудование, труднореализуемо и не получило широкого распространения. Однако, в рамках конкретных проектов, для достижения наилучшей производительности, используются сервисы, созданные с учетом особенностей платформ на которых их исполняют. Примерами служат решения от Google [30], Facebook [31] и многие другие веб-сервисы с высокой сетевой и информационной нагрузкой.

8.5. Среда для облачных вычислений (Cloud Observer v0.5)

Дальнейшим вектором развития проекта Cloud Observer стал уход от платформы Microsoft .NET к языку программирования C++ и реализация распределённой вычислительной среды с учётом эволюционировавшего к данному моменту видения идей проекта. Этот этап подробно описан в основной части данной дипломной работы.

9. Приложение 2. Команды и опции средства поддержки

В данном приложении представлен перечень и описание всех реализованных в скрипте сборки и поддержки команд и опций (Рис. 8).

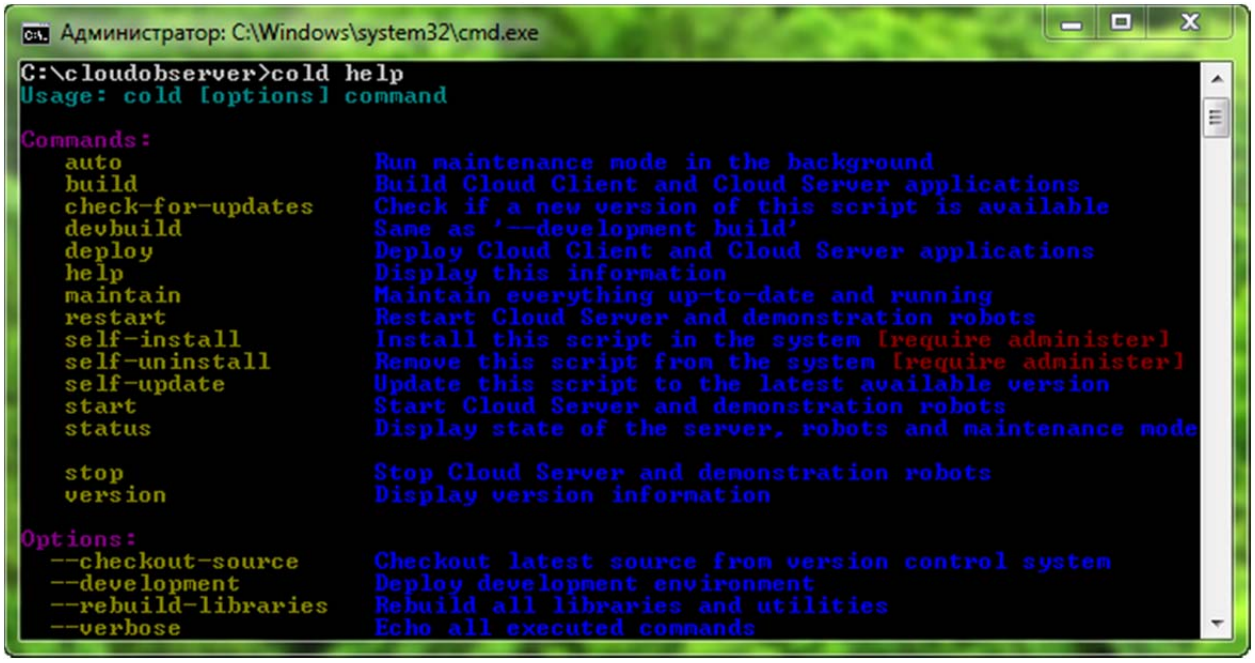

Рисунок 8. Команды и опции средства поддержки

9.1. Команды

`build`

Команда предназначена для сборки узла системы Cloud Observer в Release-конфигурации. Сначала производится загрузка и сборка исходных кодов утилит CMake [21] (необходима для сборки библиотек OpenAL [57] и OpenCV [58]), Premake [63] (необходима для сборки приложений Cloud Server и Cloud Client) и YASM [78] (необходима для сборки библиотек FFmpeg [24]). На операционных системах семейства Windows также собирается утилита NASM [53], необходимая для сборки библиотек OpenSSL [59], и загружаются бинарные файлы клиентского приложения системы контроля версий Subversion [9]. Далее загружаются исходные коды и производится сборка всех необходимых библиотек, а именно: Boost [12], FFmpeg [24], OpenAL [57], OpenCV [58] и OpenSSL [59]. Архивы всех загруженных исходных кодов утилит и библиотек сохраняются в папке downloads рабочего каталога. Наконец, загружаются из репозитория системы контроля версий Subversion и собираются исходные коды приложений Cloud

Server и Cloud Client. Собранные бинарные файлы приложений помещаются в папку install-dir рабочего каталога, а исходные коды приложений удаляются.

`devbuild`

Команда предназначена для сборки узла системы Cloud Observer в Debug-конфигурации. Действия команды аналогичны действиям команды build с опцией –development. Сначала производится загрузка и сборка исходных кодов утилит CMake [21] (необходима для сборки библиотек OpenAL [57] и OpenCV [58]), Premake [63] (необходима для сборки приложений Cloud Server и Cloud Client) и YASM [78] (необходима для сборки библиотек FFmpeg [24]). На операционных системах семейства Windows также собирается утилита NASM [53], необходимая для сборки библиотек OpenSSL [59], и загружаются бинарные файлы клиентского приложения системы контроля версий Subversion [9]. Далее загружаются исходные коды и производится сборка всех необходимых библиотек, а именно: Boost [12], FFmpeg [24], OpenAL [57], OpenCV [58] и OpenSSL [59]. Архивы всех загруженных исходных кодов утилит и библиотек сохраняются в папке downloads рабочего каталога. Наконец, загружаются из репозитория системы контроля версий Subversion и собираются исходные коды приложений Cloud Server и Cloud Client. Сборка производится в Debug-конфигурации. Собранные бинарные файлы приложений помещаются в папку install-dir рабочего каталога. Исходные коды приложений (рабочие копии системы контроля версий Subversion) сохраняются в папках cloudserver-src и cloudclient-src рабочего каталога соответственно. Проектные файлы для использованной при сборке версии Microsoft Visual Studio [43] (на ОС семейства Windows) или make-файлы (на Unix-подобных ОС) сохраняются в подпапках projects.

`deploy`

Команда производит копирование бинарных файлов собранных приложений Cloud Server и Cloud Client из папки install-dir в папку run_dir рабочего каталога. Необходимо выполнить данную команду перед запуском узла.

`start/stop/restart`

Команды производят запуск, остановку или перезапуск узла соответственно. Перед использованием данных команд необходимо выполнить команду `deploy`.

`status`

Команда отображает состояние узла, демонстрационных клиентов Observer Service (роботов) и режима поддержки. Отображается информация об идентификаторах запущенных процессов и используемой ими оперативной памяти.

`self-install/self-uninstall`

Команды производят установку или удаление ярлыков для быстрого доступа к средству поддержки. На операционных системах семейства Windows это достигается путём создания небольшого скриптового файла в каталоге `%WINDIR%` (обычно это `C:\Windows`). Файл имеет имя, аналогичное названию скрипта-обёртки для основного средства под Windows (`cold.bat`). Это позволяет избежать модификации переменной окружения PATH, но обеспечить при этом доступ к средству поддержки из любого места системы, в том числе через Пуск->Выполнить (`Win+R`) и через строку поиска меню Пуск (впервые появилась в Windows Vista). На Unix-подобных операционных системах создаётся символическая ссылка на скрипт поддержки в системном каталоге `/bin`. Выполнение данных команд требует наличия административных привилегий у пользователя.

`check-for-updates`

Команда проверяет наличие новых версий средства поддержки в репозитории системы контроля версий Subversion [9].

`self-update`

Команда производит обновление средства поддержки до последней версии, доступной в репозитории системы контроля версий Subversion [9].

`maintain`

Команда запускает режим поддержки узла системы на данном компьютере. В этом режиме производится сборка (если она ещё не была произведена) и запуск узла и

демонстрационных клиентов Observer Service (роботов) в Release-конфигурации. Далее начинается процесс регулярной (каждые 5 минут) проверки появления новых ревизий в репозитории системы контроля версий Subversion [9]. В случае обнаружения новой версии, происходит пересборка и перезапуск приложений Cloud Server и Cloud Client. Если в новой версии были внесены изменения в само средство поддержки, то предварительно производится самообновление средства поддержки и пересборка всех библиотек и утилит.

`auto`

Команда запускает режим поддержки в фоновом режиме. На Unix-подобных операционных системах это достигается использованием утилиты `nohup`. На операционных системах семейства Window данная команда аналогична команде `maintain`.

`version`

Команда отображает информацию о версии средства поддержки.

9.2. Опции

`--checkout-source`

Опция форсирует загрузку последней ревизии исходных кодов приложений Cloud Server и Cloud Client из репозитория системы контроля версий Subversion [9]. Локальные рабочие копии в папках cloudserver-src и cloudclient-src удаляются, при их наличии. Опция влияет только на поведение команд `build` и `devbuild`.

`--development`

Опция форсирует сборку приложений Cloud Server и Cloud Client в Debug-конфигурации. Влияет только на поведение команды `build`.

`--rebuild-libraries`

Опция форсирует пересборку всех утилит (CMake, NASM, Premake, YASM) и библиотек (Boost, FFmpeg, OpenAL, OpenCV, OpenSSL). Все предыдущие версии будут удалены. Опция влияет только на поведение команд `build` и `devbuild`.

`--verbose`

Опция форсирует отображение информации о вызываемых командах и их выводе в стандартные потоки вывода и ошибки (stdout и stderr). Вся эта информация всегда (даже без данной опции) заносится в файл журнала (`cold.log`), расположенный в рабочем каталоге.

10. Приложение 3. Работа с демонстрационной системой

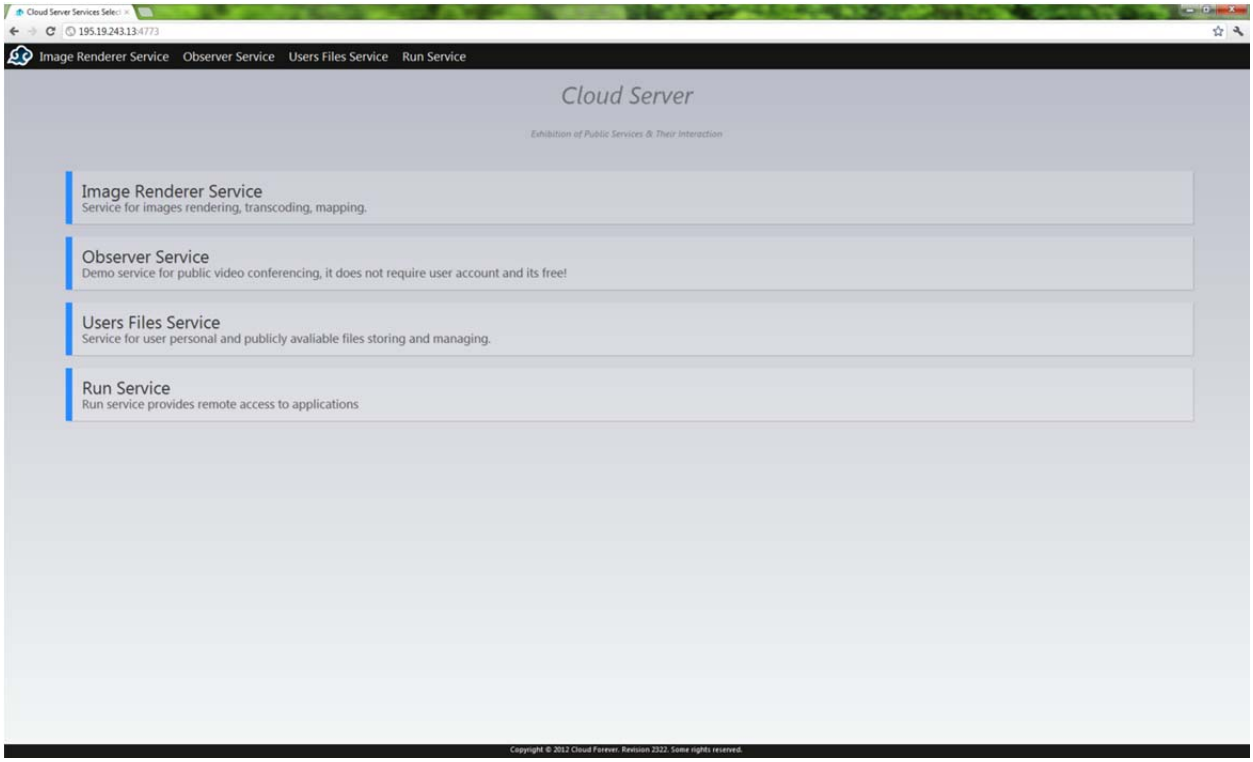

Рисунок 9. Меню выбора сервисов

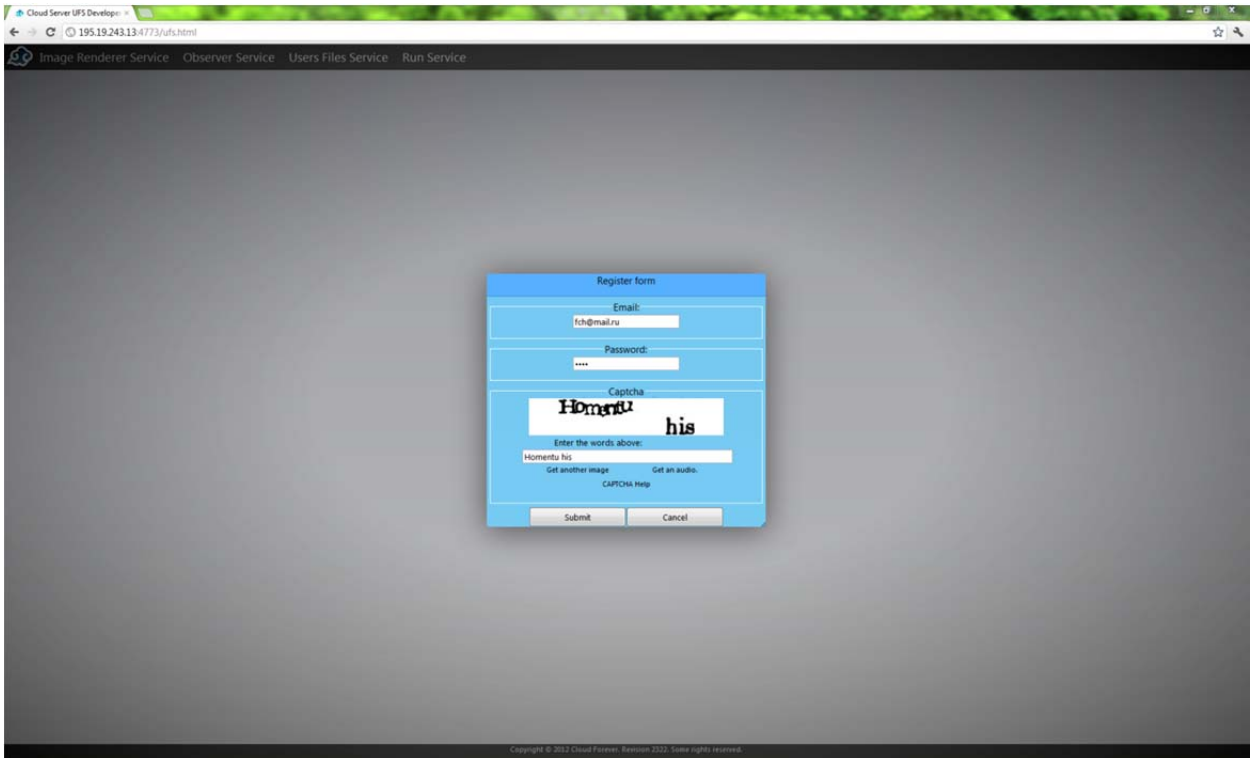

Рисунок 10. Форма регистрации пользователя

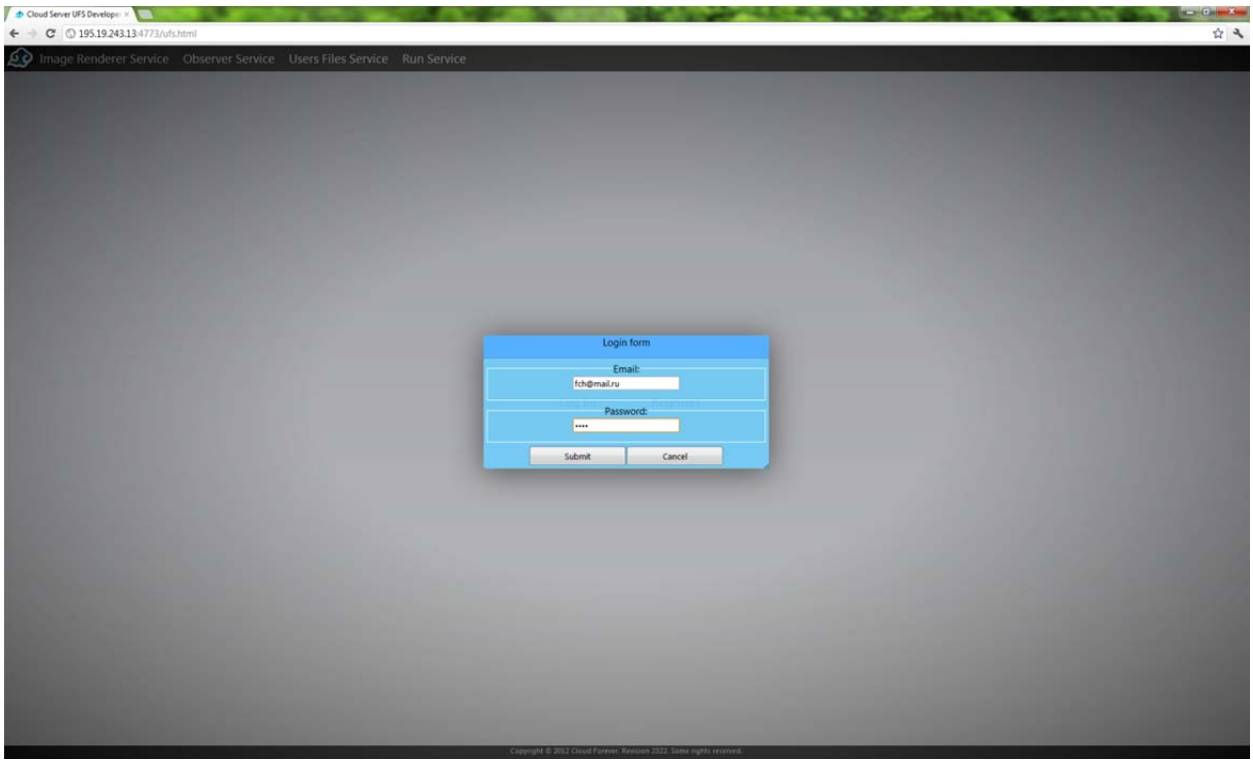

Рисунок 11. Форма авторизации

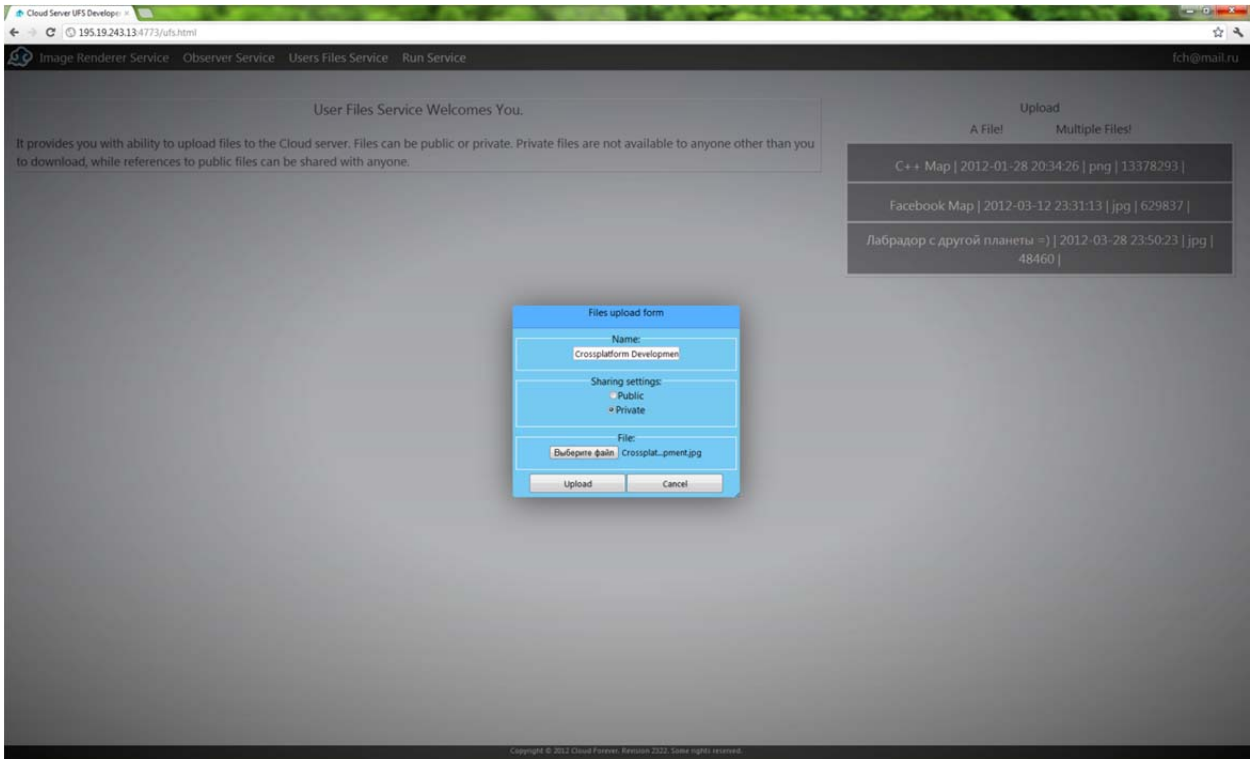

Рисунок 12. Форма загрузки файла в личное хранилище

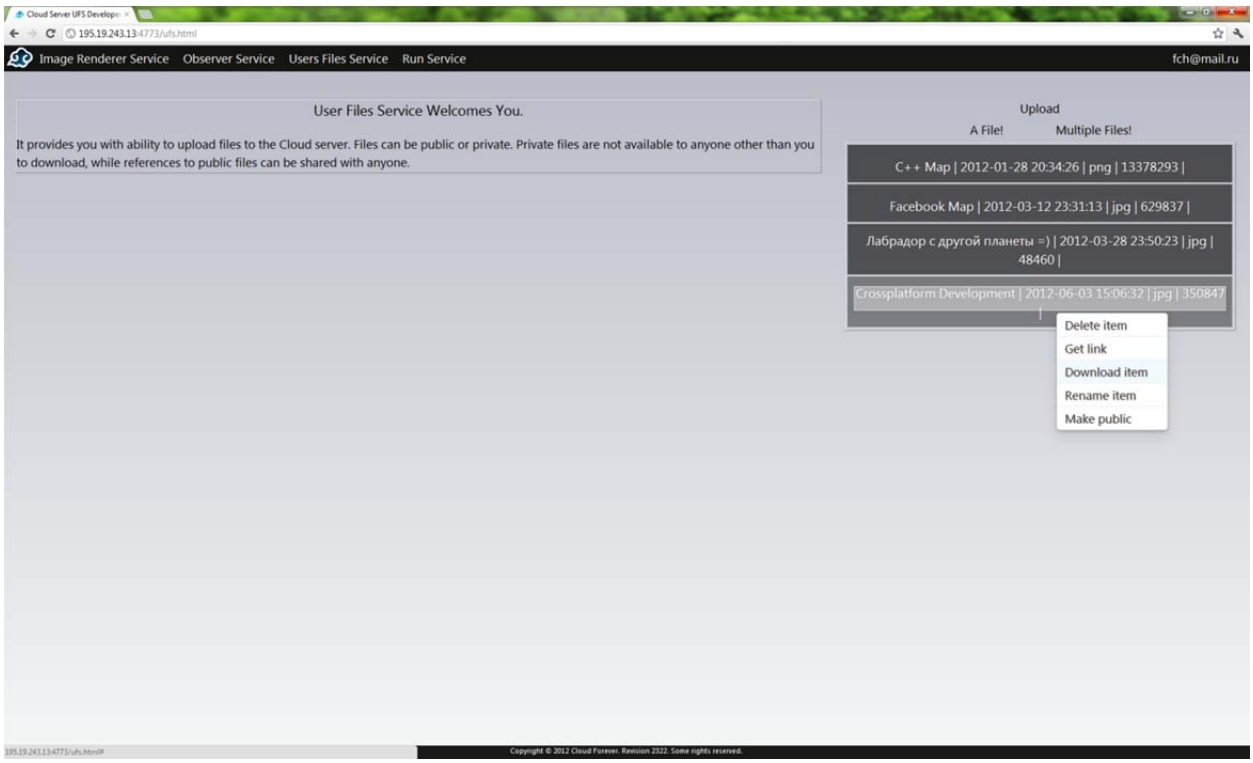

Рисунок 13. User Files Service

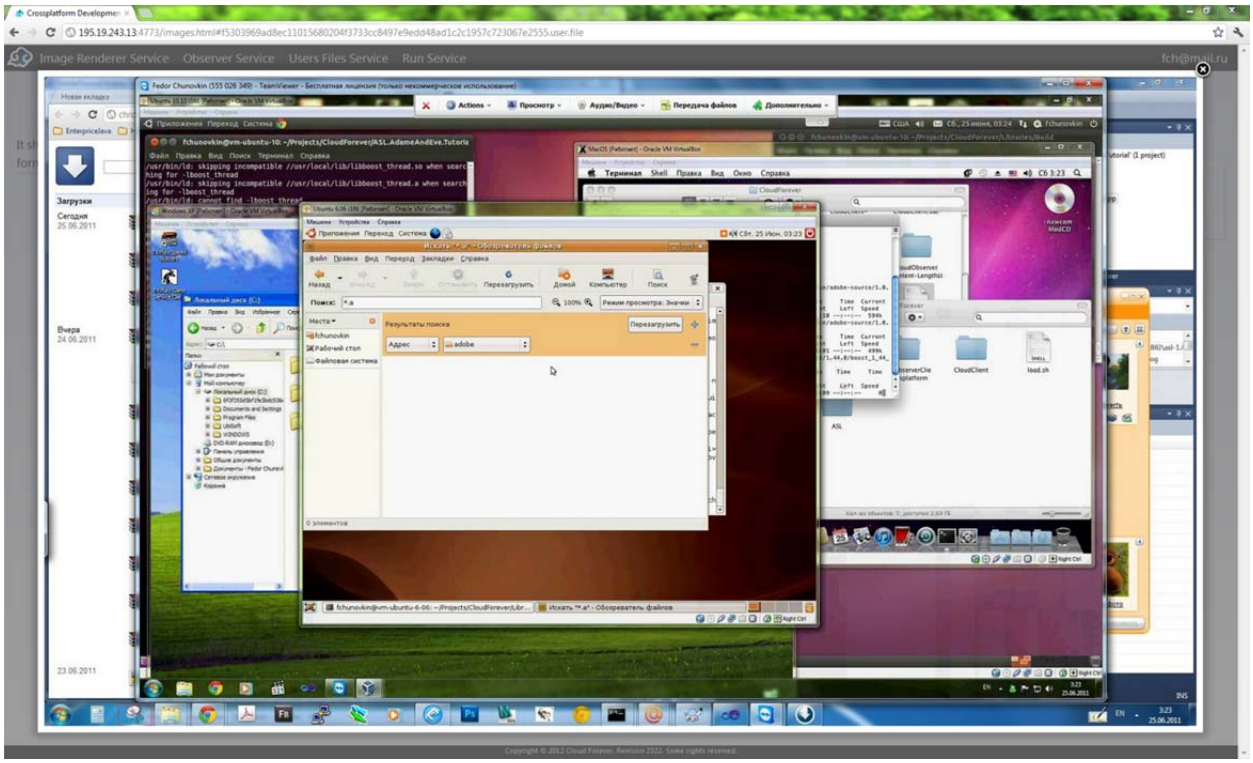

Рисунок 14. Image Renderer Service

Рисунок 15. Run Service

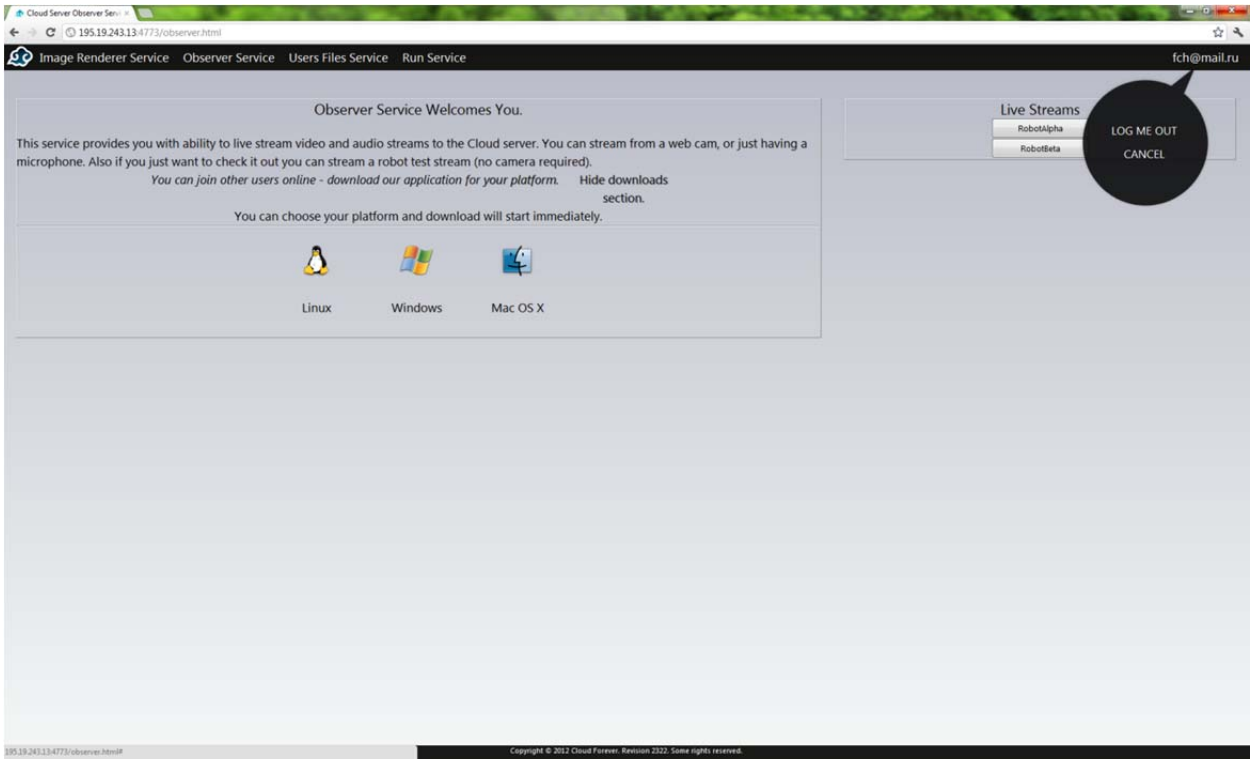

Рисунок 16. Observer Service

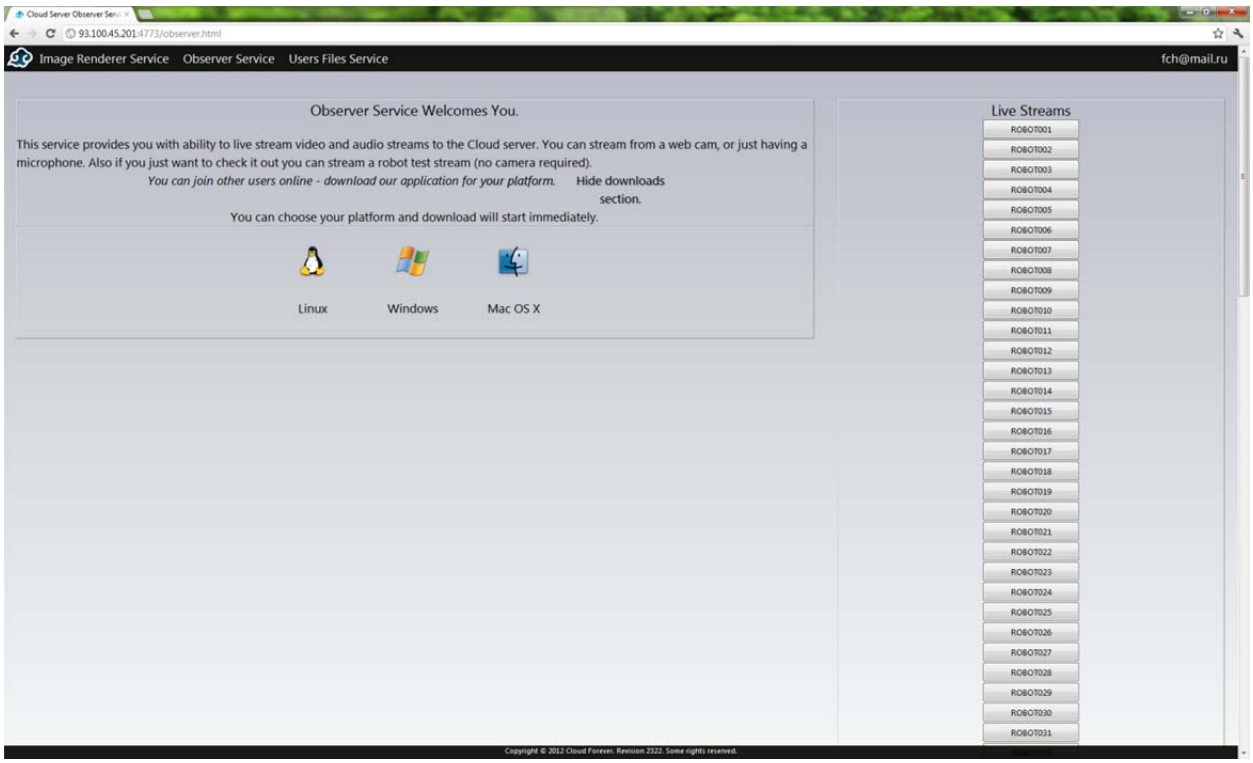

Рисунок 17. Нагрузочное тестирование - 100 роботов (снимок 1)

Рисунок 18. Нагрузочное тестирование - 100 роботов (снимок 2)

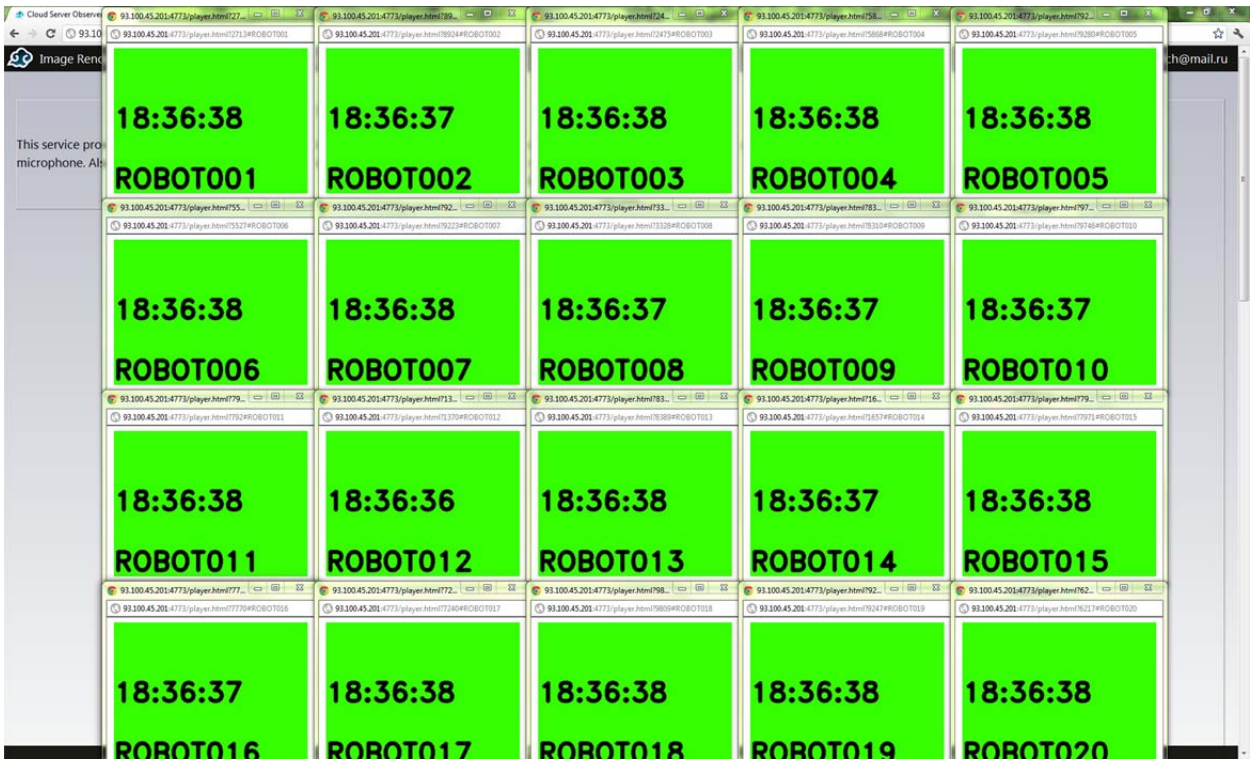

Рисунок 19. Просмотр 20 роботов

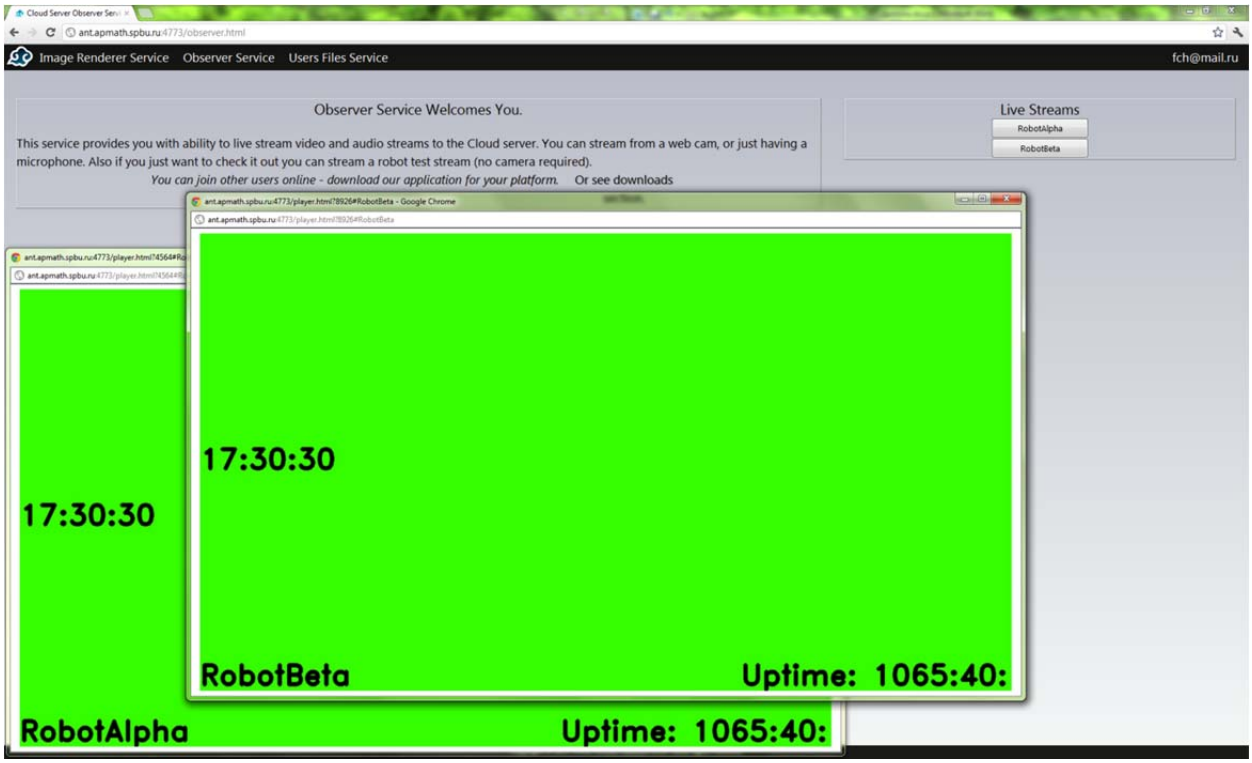

Рисунок 20. Два демонстрационных робота непрерывно вещают аудио/видео-потоки на протяжении 1065 часов (около полутра месяцев)

11. Список литературы

1. 0MQ: The Intelligent Transport Layer
   http://www.zeromq.org/
2. Adaptive Communication Environment (ACE)
   http://www.cs.wustl.edu/~schmidt/ACE.html
3. Adobe AIR Platform
   http://www.adobe.com/ru/products/air.html
4. Adobe Flash Media Server
   http://www.adobe.com/ru/products/flashmediaserver/
5. Adobe Flash Platform
   http://www.adobe.com/products/flash.html
6. Adobe Flash Video (F4V/FLV) File Format Specification
   http://download.macromedia.com/f4v/video_file_format_spec_v10_1.pdf
7. Amazon Elastic Compute Cloud (Amazon EC2)
   http://aws.amazon.com/ec2/
8. Amazon Simple Storage Service (Amazon S3)
   http://aws.amazon.com/s3/
9. Apache Subversion
   http://subversion.apache.org/
10. Apple Mac OS X
    http://www.apple.com/ru/macosx/
11. ARM Microprocessor Architecture
    http://www.arm.com/
12. Boost C++ Libraries
    http://www.boost.org/
13. Boost.Build System
    http://www.boost.org/boost-build2/
14. Bourne-Again Shell (Bash)
    http://www.gnu.org/software/bash/
15. C++ and Beyond 2011: Herb Sutter
    http://channel9.msdn.com/posts/C-and-Beyond-2011-Herb-Sutter-Why-C
16. C++ Renaissance at Microsoft
    http://mariusbancila.ro/blog/2011/06/20/cpp-renaissance-at-microsoft/

17. Cascading Style Sheets (CSS)
http://www.w3.org/TR/2011/REC-CSS2-20110607/
18. Cloud Observer Documentation (PDF)
http://195.19.243.13/documentation/cloud_observer_cpp_docs.pdf
19. Cloud Observer Online Documentation (HTML)
http://195.19.243.13/documentation/html/
20. Cloud Observer Project
http://code.google.com/p/cloudobserver/
21. CMake – Cross Platform Make
http://www.cmake.org/
22. Doxygen Documentation System
http://www.doxygen.org/
23. Dropbox File Hosting
https://www.dropbox.com/
24. FFmpeg Libraries
http://www.ffmpeg.org/
25. Ganglia Monitoring System
http://ganglia.sourceforge.net/
26. Globus – Open Source Grid Software
http://www.globus.org/
27. GNU Compiler Collection (GCC)
http://gcc.gnu.org/
28. Google App Engine
http://appspot.com/
29. Google Drive
https://drive.google.com/start
30. Google: Cluster Computing and MapReduce
http://code.google.com/intl/ru-RU/edu/submissions/mapreduce-minilecture/listing.html
31. HipHop for PHP
https://github.com/facebook/hiphop-php/wiki/
32. HyperText Markup Language (HTML) 5
http://dev.w3.org/html5/spec/
33. Intel x86 Microprocessor Architecture
http://en.wikipedia.org/wiki/X86

34. Java Programming Language
http://www.oracle.com/technetwork/java/index.html
35. JavaScript Programming Language
http://en.wikipedia.org/wiki/JavaScript
36. jQuery Library
http://jquery.com/
37. Microsoft .NET Framework
http://www.microsoft.com/net
38. Microsoft DirectShow API
http://msdn.microsoft.com/en-us/library/windows/desktop/dd375454(v=vs.85).aspx
39. Microsoft DirectSound API
http://msdn.microsoft.com/en-us/library/windows/desktop/bb318665(v=vs.85).aspx
40. Microsoft Language-Integrated Query (LINQ)
http://msdn.microsoft.com/ru-ru/library/bb397926(v=vs.90).aspx
41. Microsoft Silverlight
http://www.microsoft.com/silverlight/
42. Microsoft SQL Server
http://www.microsoft.com/sqlserver/ru/ru/
43. Microsoft Visual Studio
http://www.microsoft.com/visualstudio/ru-ru
44. Microsoft Windows Azure
http://www.windowsazure.com/
45. Microsoft Windows Communication Foundation (WCF)
http://msdn.microsoft.com/ru-ru/library/dd456779.aspx
46. Microsoft Windows
http://windows.microsoft.com/
47. Minimalist GNU for Windows (MinGW)
http://www.mingw.org/
48. Mongoose – Easy to Use Web Server
http://code.google.com/p/mongoose/
49. Mono Project
http://www.mono-project.com/
50. Moore's Law
http://en.wikipedia.org/wiki/Moore%27s_law

51. Motion JPEG (MJPEG)
http://en.wikipedia.org/wiki/Motion_JPEG
52. MPEG Audio Layer III (MP3) Format
http://en.wikipedia.org/wiki/MP3
53. Netwide Assembler (NASM)
http://www.nasm.us/
54. Network Address Translation (NAT)
http://en.wikipedia.org/wiki/Network_address_translation
55. Ohloh Languages Comparison
https://www.ohloh.net/languages?query=&sort=code
56. Opa: Advancing Web Development to the Next Generation
http://opalang.org/
57. OpenAL Library
http://connect.creativelabs.com/openal/
58. OpenCV Libraries
http://opencv.willowgarage.com/
59. OpenSSL Libraries
http://www.openssl.org/
60. Perl Programming Language
http://www.perl.org/
61. PHP Programming Language
http://www.php.net/
62. Platform as a Service (PaaS)
http://en.wikipedia.org/wiki/Platform_as_a_service
63. Premake
http://industriousone.com/premake
64. Python Programming Language
http://www.python.org/
65. Qt Framework
http://qt.nokia.com/products/
66. Red5 Media Server
http://www.red5.org/
67. RESTful Architecture
http://en.wikipedia.org/wiki/Representational_state_transfer

68. Service-Oriented Architecture (SOA)
http://en.wikipedia.org/wiki/Service-oriented_architecture
69. Session Traversal Utilities for NAT (STUN)
http://en.wikipedia.org/wiki/STUN
70. Simple Object Access Protocol (SOAP)
http://en.wikipedia.org/wiki/SOAP
71. Software as a Service (SaaS)
http://en.wikipedia.org/wiki/Software_as_a_service
72. Sparc Microprocessor Architecture
http://www.sparc.org/
73. Swarm Development Group
http://www.swarm.org/
74. UNICORE: Distributed Computing and Data Resources
http://www.unicore.eu/
75. VBScript Programming Language
http://en.wikipedia.org/wiki/VBScript
76. WebKit – Open Source Web Browser Engine
http://www.webkit.org/
77. Wt - C++ Web Toolkit
http://www.webtoolkit.eu/wt
78. Yasm Modular Assembler (YASM)
http://yasm.tortall.net/