САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Математико-механический факультет

Кафедра системного программирования

Кузнецов Кирилл Олегович

Облачная мультиагентная платформа на основе JADE и Google App Engine

Магистерская диссертация

Допущена к защите.

Зав. кафедрой:

д.ф.-м.н., проф. А.Н. Терехов

Научный руководитель:

к.ф.-м.н., Д.Ю. Бугайченко

Рецензент:

к.ф.-м.н., М.К. Грачев

Санкт-Петербург

2013

SAINT-PETERSBURG STATE UNIVERSITY

Mathematics & Mechanics Faculty

Chair of Software Engineering

Kirill Kuznetcov

Cloud multi-agent framework based on JADE and Google App Engine

Master's Thesis

Admitted for defence.

Head of the chair:

Professor A.N. Terekhov

Scientific supervisor:

PhD D.Y. Bugaychenko

Reviewer:

PhD M. K. Grachev

Saint-Petersburg

2013

Содержание

1 Введение

В настоящее время существует множество мультиагентных систем, которые решают задачи в самых разных областях: поиск (как интеллектуальный так и обычная агрегация), логистика, транспорт, моделирование, телекоммуникации [13,14]. Мультиагентные системы представляют собой парадигму распределенных вычислений, которая основана на взаимодействии множества интеллектуальных агентов [1]. Они часто применяются для решения проблем используя децентрализованный подход, когда несколько агентов вносят свой вклад в общее решение кооперируясь друг с другом. Одно из ключевых полезных свойств агентов — это интеллектуальное поведение, которое может быть заложено в каждого из них в соответствии с общим подходом к решению задачи, в рамках которого требуется взаимодействие многих агентов, работающих параллельно на одной или нескольких машинах одновременно.

Многие мультиагентные системы должны динамически адаптироваться к изменяющейся (особенно часто — возрастающей) нагрузке и увеличивающимся объемам данных. С добавлением новых агентов, помимо затрат на их обслуживание, увеличивается также число возможных коммуникаций для уже существовавших ранее агентов. В качестве примера можно рассмотреть систему управления логистикой для перевозок грузов. Есть множество агентов соответствующих транспортным средствам — грузовикам самолетам, баржам - и грузам. В системе постоянно появляются новые грузы, которые нужно доставить оптимальным образом, изменяется состав активных участников логистической цепочки. В случае развития сети доставки или при появлении новых участников, не пользовавшихся системой ранее, необходимо адаптироваться к новым, возрастающим требованиям к ресурсам.

Требуемые машинные мощности, а также хранилище достаточных объемов могут быть представлены вычислительным облаком [3,4]. Системы облачных вычислений предоставляют хорошо масштабируемую инфраструктуру для высокопроизводительных вычислений, которые динамически адаптируются к нуждам пользователя и приложения. На данный момент облака по большей части используются для обеспечения интенсивной вычислительной нагрузки и для предоставления больших объемов для хранения данных. И обе этих цели сочетаются с третьей: потенциально снизить издержки на администрирование и использование приложений, т. к. эти издержки берет на себя облако.

Облачная модель вычислений естественным образом подходит для систем с нерегулярной нагрузкой, например:

- Систем с мультиагентным моделированием - в момент запуска симуляции нужна большая мощность, но большую часть времени ресурсы не используются и простаивают. Таким образом размещение такой системы непосредственно на выделенных физических серверах не оправдано.
- Системы для решения задач, в которых возникает необходимость периодически проводить нагруженный расчет (например, составление расписания).
- Задачи с ярко выраженной сезонностью. Например, система по проработке плана отпуска (летом и в период каникул нагрузка будет больше, чем в холодный сезон) или, например, кооперативные покупки – обычно в период праздников возникает повышенной спрос. Благодаря

гибкому управлению инфраструкторой в облаке можно сильно сэкономить.

- Приложения, для которых ожидается бурное развитие, под это определение, в принципе, подойдет любой стартап.

Мультиагентные приложения могут быть интегрированы с облачной инфраструктурой, чтобы использовать огромное число процессоров и обрабатывать значительные объемы данных [4]. Такой подход позволит переложить серьёзные вычисления на плечи служебных процессов и элементов хранилища в облаке. Целая система или её наиболее загруженная часть может быть размещена в облаке, тогда как «легковесные» части системы могу выполняться на локальном сервере или на клиентской машине. Это позволит повысить эффективность агентов, а также может сделать их легче и умнее. Это достигается при помощи производительных мощностей облака, которые могут повысить «интеллект» агентов за счет использования более сложных алгоритмов или повышенной точности. При этом, агенты смогут адаптироваться к доступным виртуальным машинам, используют такие свои базовые свойства как автономность, проактивность, способности к переговорам и обучению. Таким образом, программные агенты могут быть размещены в облаке, чтобы улучшить их гибкость и адаптируемость, повысить автономность в использовании ресурсов, предоставлении услуг и выполнении крупномасштабных приложений.

На текущей момент было предпринято несколько попыток создания облачных мультиагентных систем, однако это были отдельные конкретные системы. Существующие мультиагентные платформы на текущий момент не предоставляют возможности запуска мультиагентной системы в облаке. В

этой работе представлена попытка реализовать платформу для выполнения систем на базе JADE – одной из самых известных и используемых в настоящее время мультиагентных платформ — в облаке. А также преимущества такого синтеза показаны на примере простейшей логистической системы, которая запущена в получившейся облачной платформе.

2 Постановка задачи

Задача данной работы состоит в том, чтобы разработать платформу для выполнения агентов на базе JADE, работающую в облаке

1. Изучить внутреннюю архитектуру JADE
2. Проанализировать существующие решения в сфере предоставления облачных сервисов, выбрать наиболее подходящий для переноса JADE.
3. Перенести платформу JADE в облако: мультиагентные системы созданные для выполнения в оригинальном JADE должны работать в облаке
4. Предложить демонстрационную систему и проанализировать её работу в облаке

3 Обзор предметной области

3.1 Мультиагентная система

Мультиагентая система представляет собой совокупность нескольких автономных сущностей, существующих и взаимодействующих в общей среде – программных интеллектуальных агентов [1, 20]. Агент автономно существует в рамках некой внешней среды, он может обмениваться с ней информацией, совершать определенные действия, но при это не может управлять состоянием среды.

Для каждого агента определен некоторый круг целей, которые он старается рациональным образом достичь своими действиями. Также агент характеризуется несколькими ключевыми свойствами:

- Реактивность - способность воспринимать внешнюю среду и реагировать в ответ на изменения в ней.
- Проактивность – способность проявлять инициативу, совершая действия направленные на достижение целей агента.
- Социальность – способность конструктивно общаться с другими сущностями, присутствующими во внешней среде

Несмотря на то, что агент сам по себе может существовать и действовать для выполнения поставленной задачи, большая эффективность достигается при работе в группе агентов, которые кооперируются друг с другом для получения информации, делегирования решения определенных задач – мультиагентной системе.

Мультиагентным системам присущи следующие характеристики:

- У каждого конкретного агента недостаточно информации или способностей для решения проблемы и, таким образом, он не имеет полного видения глобальной задачи, которая должна быть выполнена.
- В системе нет глобального контроля.
- Нет централизованного хранения данных.
- Вычисления происходят асинхронно.

Мультиагентные системы используются в задачах, где методы централизованного решения менее эффективны. Это такие задачи, как например: логистика, моделирование, сбор и обработка информации, управление сложными автоматизированными системами и т.д.

За последние годы появилось множество инструментариев, поддерживающих различные агентные архитектуры и предоставляющие библиотеки для протоколов взаимодействия: Jason [10], SPADE [17], JADE[9], Cougaar [12], Jadex [11] и т.д. Были приняты стандарты в области мультиагентных систем, такие как KQML (Knowledge Query and Manipulation Language) [21], который изначально создавался для экспертных систем, позже был принят и в области мультиагентных технологий, и FIPA [6]. Эти технологии создавались для использования в традиционных вычислительных системах и на текущий момент существует не так много исследований на тему интеграции мультиагентных систем и облачных вычислений.

3.2 Стандарты FIPA

В 1996 году в Швейцарии была создана некоммерческая организация FIPA (Foundation for Intelligent Physical Agents) . Целью создания этой организации была разработка стандартов для мультиагентных систем. Эти стандарты специфицируют то, как агенты должны существовать в системе, коммуницировать и взаимодействовать между собой [6]. А также, стандарты коснулись реализации систем, в которых агенты могли бы выполняться и действовать - агентных платформ(АР). Для таких платформ были специфицированы три выделенных служебных роли агентов:

- Система управления агентами (Agent Management System - AMS) – этот агент отвечает за доступ к справочнику всех агентных идентификаторов (AID). Каждому AID в справочники сопоставляется адрес агента в системе, через который будет возможно сообщение по транспортному протоколу. Все агенты платформы регистрируются в справочнике через AMS при появлении в системе, чтобы получить AID.

- Справочный агент(Directory Facilitator - DF) – предоставляет сервис “Желтых страниц” другим агентам в системе. Каждый из агентов должен зарегистрировать свои сервисы в этом справочнике. При обращении к DF для каждого конкретного сервиса клиенту возвращается список агентов, реализующих этот сервис.

- Сервис транспорта сообщений (Message Transport Service - MTS) – отвечает за передачу сообщений между агентами в рамках одной платформы, а также за связь с агентами в других платформах.

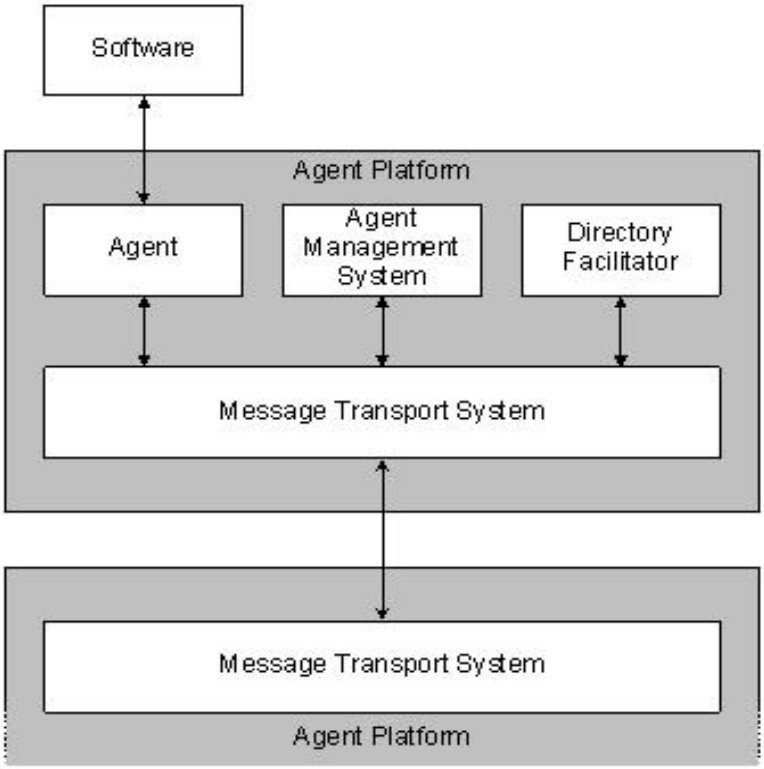

Рис.1: Архитектура FIPA

Также стандарты описывают язык общения между агентами - FIPA Agent Communication Language (ACL), однако только синтаксис и семантику, реализация этого языка не специфицирована. Помимо этого, спецификации FIPA описывают протоколы взаимодействия агентов. Наиболее известен FIPA Contract Net - он применяется для выбора агента-поставщика необходимого сервиса среди нескольких кандидатов.

3.3 **JADE**

JADE (Java Agent Development framework) – это один из самых распространенных инструментов для создания мультиагентых систем, изначально созданный в Telecom Italia [5,9]. JADE написан на Java, поэтому является кросс-платформенным, может работать на беспроводных мобильных устройствах,

поддерживает стандарты FIPA и распространяется бесплатно по лицензии LGPL.

Как платформа для выполнения интеллектуальных агентов JADE:

- Обеспечивает возможность развертывания MAC и берет на себя функции управления жизненным циклом агентов (создание, выполнение, приостановка и завершение). Библиотека Java-классов позволяет реализовать агентов со сколь угодно сложным поведением, а также описать их взаимодействие. В основном, это классы, описывающие агентов на каждом этапе их жизненного цикла, а также их взаимодействие с графическим интерфейсом (если он есть).

- Поддерживает использование различных архитектур агентов и мультиагентных систем – гибкая исходная структура JADE позволяет легко расширять платформу (например - интеграция с Jess [24] и BDI4JADE [25]). JADE не придерживается какой-то конкретной методологии и обладает достаточно высокой гибкостью в использовании.

- Обеспечивает сообщение агентов в системе (механизм транспорта сообщений, языки и онтологии), представление знаний. Для передачи сообщений в JADE используется Agent Communication Channel (ACC). Все взаимодействия между агентами происходят посредством пересылки сообщений языка FIPA ACL.

- Включает в себя инструменты разработки и отладки.

- Помимо агентов и поведений, JADE реализует FIPA-протоколы взаимодействия между агентами, такие как FIPA Contract Net Interaction Protocol и сервис желтых страниц.

3.4 Архитектура JADE

В платформе JADE агенты выполняются в контейнерах, которые могут быть распределены по сети. Эти контейнеры представляют собой отдельные Java-машины. Каждый агент – это поток в процессе, соответствующем его контейнеру. На Java-машине контейнера запущен экземпляр среды выполнения JADE и все сервисы, необходимые для хостинга и выполнения агентов. Существует специально выделенный, так называемый, главный контейнер, который представляет собой точку начальной загрузки в платформе. Это первый загружаемый контейнер и все остальные для присоединения к платформе должны регистрироваться с помощью главного контейнера.

Будучи точкой начальной загрузки, главный контейнер отвечает за:

- Управление таблицей контейнеров, которая хранит все объектные ссылки и транспортные адреса всех контейнерных узлов, составляющих платформу.
- Администрирование глобальной таблицы агентских дескрипторов (Global Agent Descriptor Table - GADT), которая является реестром всех агентов, существующих в платформе, включая их статус и расположение
- Размещение AMS и DF агентов.

Как уже было сказано выше, в системе общение агентов друг с другом, и служебных и рядовых, а также с сервисами платформы осуществляется с помощью сообщений. Для передачи сообщений в JADE существует два протокола: Message Transport Protocol (MTP) и Internal Message transport Protocol

(IMTP). MTP используется для коммуникации между агентами на разных платформах и реализован в соответствии со стандартом FIPA. Передача сообщений осуществляется по HTTP. IMTP – протокол для общения агентов внутри одной платформы JADE. IMTP допускает различные реализации, так изначально в JADE представлены две версии: первая использует Java Remote Method Calling (технологию взаимодействия с объектами, находящимся на другой Java-машине, по сети), вторая реализация создана для мобильных устройств и предназначена для работы с Java ME.

Можно привести типичный пример развертывания определенной мультиагентой системы разработанной с использованием JADE: платформа включает в себя все агенты выполняющиеся на вычислительных мощностях какого-то дата-центра, а контейнеры в платформе соответствуют различным машинам, находящимся в этом дата-центре.

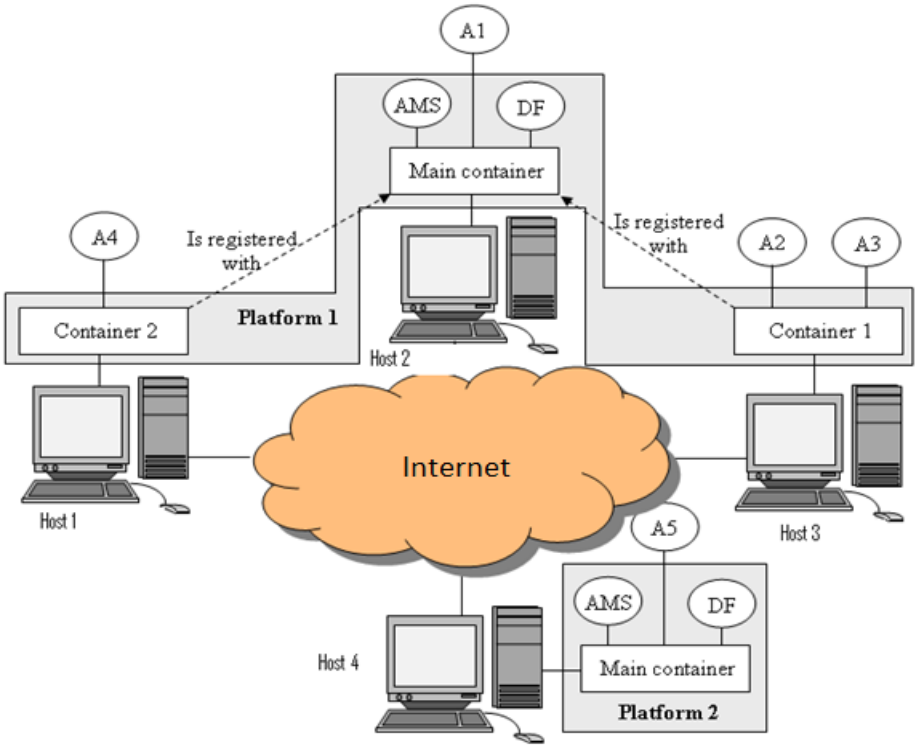

Рис.2: Архитектура платформы JADE

3.5 **Облачные сервисы**

Национальный институт стандартов и технологии (National Institute of Standard and Technology - NIST) дает следующее полное определение:

“Облачные вычисления это модель предоставления повсеместного и удобного сетевого доступа с оплатой за использование по мере необходимости к общему пулу конфигурируемых вычислительных ресурсов (например, сетей, серверов, систем хранения, приложений и сервисов), которые могут быть быстро предоставлены и освобождены с минимальными усилиями по управлению и необходимостью взаимодействия с провайдером услуг.” Облако состоит из группы соединенных между собой и виртуализированных компьютеров, которые выделяются динамически и предоставляются пользователю

Ключевые свойства, которые выделяют облачные вычисления от традиционных подходов, были определены в [22] и обычно содержат следующие:

- Основная инфраструктура и программное обеспечение выделяется предоставляются пользователю как услуга.
- Облачные вычисления представляют гибкую и масштабируемую архитектуру.
- Обеспечивается предоставление сервисов по требованию и гарантируется качество сервисов (Quality of Service - QoS).
- Оплата производится по факту использования вычислительных ресурсов без авансовых обязательств от пользователей облака.

- Облака общедоступны и используются многими клиентами.
- Доступность с любых устройств через интернет.

Более того, согласно NIST, существует три категории способов предоставления облачных сервисов, которые описывают различные степени доступности ресурсов облака:

- *Infrastructure as a Service* (*IaaS*). Пользователю предоставляются возможности использования процессора, хранилища данных, сетевого доступа и других вычислительных ресурсов. Есть возможность устанавливать и запускать программное обеспечение, которые может включать в себя операционную систему и/или приложения. Пользователь не может контролировать аппаратную часть облачной инфраструктуры, однако имеет доступ к управлению операционной средой, хранилищем, установленными программами и, возможно, сетевыми устройствами. В действительности, можно говорить, что пользователю предоставляется выделенная виртуальная машина. Примерами таких коммерческих облачных инфраструктур являются Amazon EC2 [2] и Rackspace [16].

- *Platform as a Service* (*PaaS*). Пользователю предоставляется возможность устанавливать приложения, разработанные им с использованием только языков, компиляторов и других инструментов и технологий, поддерживаемых поставщиков облачных услуг, например, Java или .NET. Потребитель услуг облака не имеет возможности управлять инфраструктурой облака, сетью, серверами, операцион-

ной системой или хранилищем, но может контролировать установленные программы и в некоторых случаях конфигурацию среды хостинга приложения. Фактически, в эту группу входят автоматически управляемые хостинг-ресурсы, которые при этом обладают хорошей масштабируемостью. Примерами таких сервисов могут служить Google App Engine [23], Windows Azure [18], Force.com [7].

- S*oftware as a Service* (*SaaS*). У потребителя облачных сервисов этого типа есть возможность использовать предоставленные провайдером приложения. Такие приложения работают в инфраструктуре облака и доступны с различных устройств через интерфейсы тонких клиентов, таких как веб-браузер или приложения для современных смартфонов (например, Dropbox, Shazam, e-mail клиенты). У пользователя нет доступа к управлению инфраструктурой, сетью, серверами, операционной системой, хранилищем данных или даже свободно конфигурировать приложение. Однако, настройка отдельных параметров часто бывают доступны доля изменения. В эту группу облачных сервисов входят законченные продукты, которые не допускают больших возможностей по своей настройке, например это могут быть почтовые службы или порталы.

Облачные вычисления являются результатом недавнего продвижения нескольких технологий [4] как со стороны аппаратного обеспечения (виртуализация и многоядерные архитектуры), так и со стороны программного обеспечения – это кластерные вычисления, грид-вычисления, веб-сервисы, сервис-

но-ориентированные архитектуры (Service Oriented Architectures - SOA), автономные вычисления и хранилища данных большого объема. В частности, для организации вычислительного облака виртуализация является ключевым элементом, который отделяет функциональность и реализацию системы от физических ресурсов. Инфраструктура облака может быть разбита на несколько виртуальных машин, которые динамически конфигурируются в соответствии с требованиями пользователя и должны выполнять независимые приложения одновременно. Виртуализация отделяет приложения от железа и пользователей друг от друга, давая им ощущение того, что вычислительная инфраструктура полностью соответствуют их приложениям за счет того, что поддерживает заданное качество услуг (QoS). Также, виртуализация используется для того чтобы изолировать приложения. Это позволяет избежать того, что отказ в работе одного приложения вызовет отказ в работе другого. Наконец, виртуализация – это способ повысить безопасность и конфиденциальность одновременно работающих в облаке приложений.

3.6 Анализ поставщиков облачных услуг

В настоящее время услуги облачных сервисов представлены многими компаниями, это и такие тяжеловесы индустрии программного обеспечения, как Microsoft с их Windows Azure [18], и интернет-гиганты, такие как Google, представившие в 2008 году Google App Engine, и Amazon. Amazon в 2005 году первыми начали предоставлять облачные сервисы с помощью Amazon Elastic Compute Cloud (Amazon EC2). Сейчас EC2 входит в платформу Amazon Web Services. Помимо крупных компаний на рынке также присутствуют поставщики облачных сервисов поменьше: Rackspace, GoGrid [8], платформа Force.com компании Salesforce [7] и другие.

Облачные провайдеры в достаточной степени отличаются друг от друга - предоставляют различные сервисы в облаке, модели оплаты, уровни поддержки, надежности, гибкости использования, а также варьируются другие характеристики [3,15]. Поэтому при выборе потенциального поставщика облачных услуг для переноса мультиагентной платформы в облако необходимо четко представлять себе, какие требования должны быть предъявлены используемому сервису, какие он представляет какие возможности, какими свойствами обладает и как может быть сконфигурирован для нужд интеграции.

Ниже будет представлен анализ наиболее популярных облачных сервисов – это уже упоминавшиеся выше: Windows Azure, Amazon Web Services, Google App Engine, Rackspace, GoGrid и Force.com. Все сервисы так или иначе поддерживают работу с Java. Хотя стоит отметить, что в Windows Azure поддержка Java достаточно ограничена [18, 27]. Для сравнения были выбраны следующие характеристики:

- Легкость масштабирования – насколько просто, удобно и безболезненно можно изменять использование ресурсов облака в соответствии с требованиями агентной платформы.
- Уровень поддержки– важно, что бы сервис развивался, активно выходили новые версии, исправляющие ошибки и расширяющие возможности, в соответствии с развитием используемых в облаке технологий; доступная информация, техническая документация, размер интернет сообщества.
- Доступность (uptime) – процент времени бесперебойной работы.

- Объем выделяемые вычислительные ресурсов - процессорные мощности, количество виртуальной памяти, выделяемое дисковое пространство а также квоты на обращение пользователей к облаку.

- Ценовая политика – у каждого провайдера облачных услуг своя модель оплаты за использование, необходимо подобрать ту, которая наиболее подходит для работы мультиагентных систем.

- Удобство обновления системы –насколько удобно обновлять запущенное в облаке приложение.

Amazon Web Services, Rackspace и GoGrid – предоставляют доступ к своим ресурсам как IaaS. Что фактически означает, что разработчику предоставлена выделенная виртуальная машина, на которой клиент может делать практически все угодно, как и на локальной машине. При этом у сервисов PaaS – Windows Azure, Google App Engine и Force.com - есть неоспоримое и очень важное преимущество – облако берет на себя организацию масштабирования. В частности, Google App Engine делает это автоматически, в зависимости от динамики обращений к серверу, наиболее безболезненно для пользователя облачного сервиса. Также PaaS-облака значительно упрощают обновление размещенного в облаке приложения, в то время как IaaS-облака никаким образом не заботиться об этом [3, 15, 19].

Если рассмотреть доступные ресурсы облака:

- Force.com – 2000 объектов в базе максимум, до 120 мб на пользователя, не более 5000 обращения от пользователя в день, ограничения на обращения к веб-страницам и прочие.

- Rackspace – неограниченная масштабируемость

- GoGrid – ресурсы не ограничены, однако для изменения использования необходимо осуществить перенос с текущего сервера на новый, с соответствующими вычислительными ресурсами, например, с больше памятью.
- Google App Engine – неограниченная масштабируемость
- AWS – ограничения на использование дискового пространства
- Windows Azure – ограничения на хранимые данные – 64 мегабайта на единицу (blob)

Говоря о доступности сервисов, все провайдеры заявляют о доступности 99,95-99,99% времени [15].

Если рассмотреть ценовую политику, то тут важно сказать, что Google App Engine единственный предоставляет бесплатный доступ к сервису, не ограниченный по времени, только по используемым ресурсам - это очень важно, так как разработчику необходим доступ к облаку, как минимум, для тестирования приложения. Force.com предоставляет бесплатный пробный доступ к ресурсам на 30 дней, однако такой вариант тоже мешает разработке. Остальные провайдеры предоставляют свои услуги исключительно за деньги.

И, наконец, упоминая о поддержке системы Google App Engine имеет наиболее широкую информационную поддержку – сайт с полным описанием возможностей и документацией[1], официальное сообщество[2], где есть обучающие примеры и другая полезная информация, не представленная в до-

[1] https://developers.google.com/
[2] http://appengine.community.com

кументации, интернациональный блог разработчиков[3]. Остальные поставщики сервисов ограничиваются только официальными сайтами.

Таким образом, подводя итог, можно сказать что Google App Engine является безусловным лидером по большинству выбранных критериев оценки. И его использование для интеграции с JADE обосновано.

[3] http://googleappengine.blogspot.com

4 Особенности реализации

4.1 Ограничения Google App Engine

Google App Engine предоставляет собой PaaS, т.е. фактически имеет свою среду выполнения Java. И, поскольку это не обычная Java-машина, на приложения размещаемые в облаке GAE накладываются определенные ограничения.

Одно из ограничений - это отсутствие возможности использовать TCP-сокеты, все сообщение по сети в Google App Engine возможно только по HTTP. Как было сказано в обзоре JADE, для передачи сообщений внутри платформы используется одна из реализаций IMTP: Java RMI или, так называемая, LEAP для мобильных устройств. Технология RMI использует TCP-сокеты для осуществления коммуникации с Java-машиной на другом хосте. LEAP реализация выполняет транспорт сообщений при помощи проприетарного протокола, который в свою очередь работает поверх TCP. Таким образом, возник конфликт в интеграции JADE и Google App Engine.

Другое важное ограничение – невозможность явно создавать потоки, есть возможности использовать их только для обработки входящих запросов или задач в так называемых очередях (task queues). Время жизни потока, созданного для обработки входящего запроса ограничено 30 секунд, а в случае задачи – это 10 минут. В обзоре JADE упоминалось, что каждый агент – это отдельный поток в процессе Java-машины, где выполняется контейнер платформы, который отвечает за этого агента. Поэтому возможность создавать и уничтожать потоки – критична для работы платформы.

Помимо этих ограничений существует еще одно – доступ Java-приложений в среде Google App Engine разрешен не для всех классов стан-

дартной библиотеки Java. Все поддерживаемые классы перечислены в специальном списке[4].

4.2 Интеграция JADE

Исходя из описанных в предыдущем пункте проблем естественным образом очертился круг задач, которые необходимо было решить для переноса JADE в облако Google App Engine: необходимо было адаптировать работу RMI, отказавшись от использования TCP-сокетов, найти способ создавать потоки в системе и, наконец, убрать или заменить классы, не поддерживаемые Google App Engine.

JADE – чрезвычайно сложная система, изменения отдельных частей могут повлечь ошибки в работе других. По этой причине при внесении изменений желательно максимально отделить части, которые должны быть переделаны, от остальной системы. Поэтому для всех классов, использование которых в облаке было невозможно, был проведен тщательный рефакторинг. По возможности, были выделены интерфейсы, а в некоторых случаях надклассы. В дальнейшем, другой частной реализацией стали классы, созданные для работы в облаке. Таким образом, была частично перепроектирована классовая иерархия JADE. Также, был добавлен менеджер, задачей которого стал выбор конкретной реализации интерфейса, в зависимости от того, где необходимо было запустить систему – в облаке или на локальной машине.

Помимо стандартного варианта размещения приложения в Google App Engine так же возможно использование так называемых бэкэндов (backends). Использование бэкэндов позволяет получить доступ к большим объемам

[4] https://developers.google.com/appengine/docs/java/jrewhitelist?hl=ru

памяти, большей вычислительной мощности, и что самое главное, допускают использование потоков. Собственно, конфигурация бэкэнда определяет набор виртуальных машин, для размещения экземпляров клиентского приложения, описывая требуемый класс мощности и число экземпляров, которые нужно запустить. У такого способа есть свои недостатки – необходимо большее вмешательство разработчика для масштабирования приложения, сложности связанные с отладкой (это будет рассмотрено подробнее). Однако, поскольку это единственный способ создания потоков в Google App Engine, было принято решение использовать механизм бэкэндов для портирования JADE в облако.

Для устранения проблем с IMTP на первой стадии работы было принято решение для начала добиться запуска JADE в режиме “один контейнер на один экземпляр JADE”, а затем перейти к использованию HTTP для транспорта сообщений вместо RMI и TCP. Стоит отметить, что, как выяснилось в процессе работы над адаптированием протокола для работы в Google App Engine, для локальной передачи сообщений в рамках одного контейнера тоже происходит с использованием текущей реализации IMTP – RMI, от которой было решено отказаться, в связи с чем пришлось осуществить переработку вертикальной (т.е. в рамках одного узла протокола) цепочки обработки сообщений и команд, отделив необходимую часть от Java RMI.

Для совместимости с Google App Engine был осуществлен рефакторинг платформы. Из структуры были аккуратно удалены части, связанные с RMI и LEAP реализациями IMTP, поскольку по большей части их все равно не возможно было использовать. Так же была удалена поддержка инструментов отладки – поскольку на текущий момент их невозможно было бы использо-

вать в облаке. Обратная интеграция этих инструментов может стать одной из возможных дальнейших задач развития платформы – например, можно настроить сообщение агентов отладчиков находящихся на локальном компьютере с платформой, работающей в GAE, или же реализовать специальные веб-интерфейсы для работы прямо в облаке. Поскольку JADE использует GUI, созданный с помощью Java Swing, который не поддерживается GAE, эта часть была полностью исключена из платформы, тем более что не было никакой необходимости её сохранять.

Таким образом, в процессе осуществления описанных выше действий из JADE было удалено большинство классов, не входящих в “белый список” Google App Engine. Однако некоторые конфликты все же остались, в этих случаях классы получилось заменить на поддерживаемые аналоги, в некоторых случае это были классы из той же иерархии, но, например на ступень выше. Так, например, были сделано для классов логгирования в системе.

4.3 Проблемы

В процессе работы над интеграцией JADE и Google App Engine существовали две основные проблемы, замедлявшие процесс. Первая из них – отсутствие возможности полноценной отладки приложения в среде разработки. Набор инструментов разработки для GAE (Software Development Kit - SDK) включают в себя тестовый сервер приложений, однако он не поддерживает запуск backend-экземпляров. Таким образом вся отладка осуществлялась при помощи работы с логами в приложении. Каждая итерация отладки занимала длительное время, поскольку требовалось полностью разместить приложение в облаке, а результат такой итерации был гораздо менее информативным чем в обыкновенном отладчике, так как не было возможности в процес-

се работы проследить интересующие особенности поведения или состояние данных в программе.

Другой проблемой JADE была скудная документированность внутреннего устройства достаточно сложной платформы и IMTP. Платформа и протокол спроектированы и использованием нескольких уровней абстракции и единственным источником информации о внутренней структуре служил сам код.

4.4 Демонстрационная система

Одна из наиболее типичных задач, которые решают мультиагентные системы – это задачи управления логистикой транспортных перевозок. Как было сказано в начале, перенос в облако наиболее актуален для систем с нерегулярной нагрузкой. Интенсивность перевозок зависит от числа заказов (может отличаться днем и ночью, в выходные и будние дни), нагруженности транспортных путей и других факторов. Помимо этого для логистических систем вполне возможно разрастание – если система работает хорошо, показывает свою эффективность в решении поставленной задачи, большее число клиентов захочет её использовать, что, в свою очередь, увеличит число агентов.

Для проверки работоспособности получившейся в результате платформы необходимо было использовать тестовую МАС. Все вышесказанное позволило в качестве тестовой предложить простейшую систему для управления логистикой. В качестве базовой была взята демонстрационная система курса по мультиагентным системам университета Порто [26].

В системе существует три вида агентов:

- Агенты-операторы (Operator) – эти агенты отвечают за выдачу заданий по транспортировке. Генерация задач происходит случайным образом. Число операторов в системе не обязано быть постоянным, могут создаваться новы агенты и завершать свою работу старые.

- Транспортные агенты (Vehicle) – эти агенты отвечают за доставку грузов. Транспортные агенты публикует свое состояние в так называемый топик (topic). Топики в JADE – это способы передачи сообщение без посылки конкретному адресату. У топика есть агенты, которые публикуют туда сообщения, и агенты-подписчики, которые их получают. Таким образом, все операторы знают, какие транспортные агенты в данный момент доступны для получения новых задач-заказов.

- Был добавлен агент, отвечающий за генерацию агентов-операторов и перевозчиков.

Процесс происходит следующим образом: агент оператор создает случайно сгенерированную задачу. И затем начинаются переговоры по алгоритму Contract Net: агент оператор посылает запрос на предложение (call for proposal) с данными задачи, транспортные агенты, получившие запрос, предлагают свою “цену” перевозки – в данном случае “цена” отражает только время, за которое задача будет выполнена, и объем топлива, которое будет потрачено. Пример формирования такой “цены” приведен на рис.3. Оператор выбирает того транспортного агента, чье предложение оказалось для него наиболее выгодным и посылает ему подтверждение, остальным агентам приходит ответ с отказом. Затем победитель принимает задачу, посылая соответствующее сообщение оператору и после того, как задача выпол-

нена, сигнализирует об этом агенту, выдавшему задание. На рис. 4 представлена диаграмма взаимодействия.

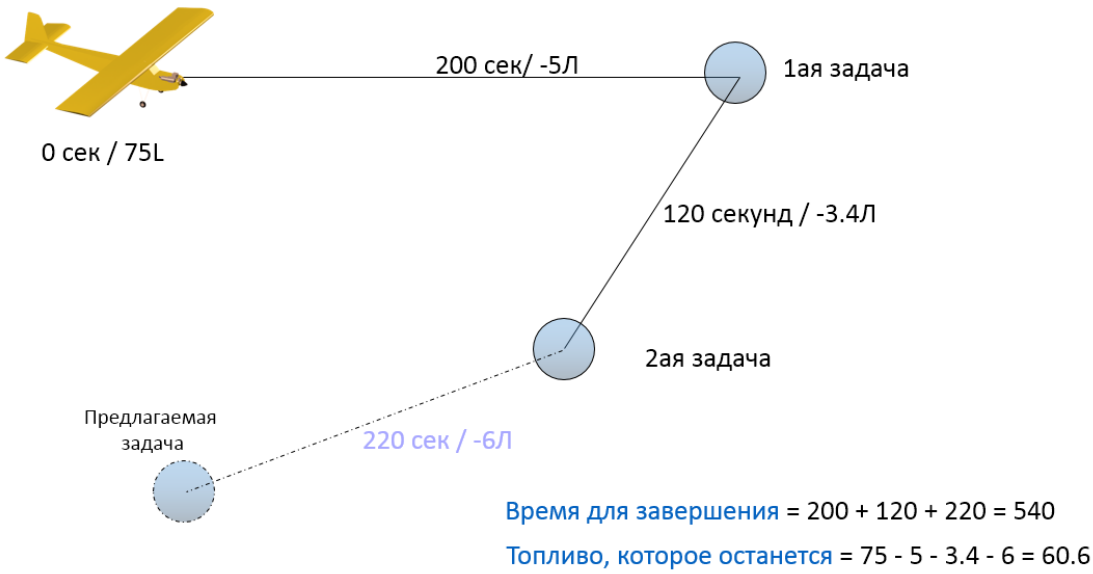

Рис. 3: Формирование “цены” выполнения задачи.

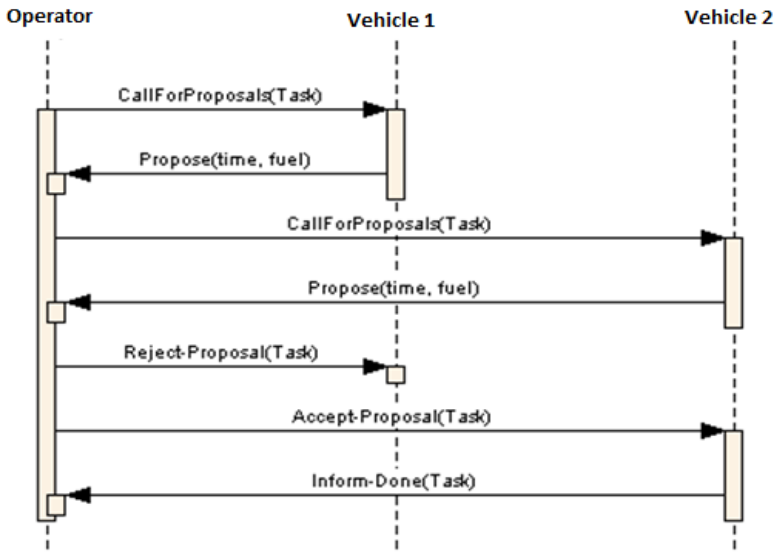

Рис. 4: Взаимодействие агентов в демонстрационной системе

Для запуска системы на облачной платформе система была адапитрована. Поскольку базовая система изначально создавалась для настольных компьютеров, в ней присутствует стандартный для Java GUI, созданный на Swing, а также средства 2d и 3d визуализации. Однако нужно было предложить демонстрационную систему, работающую в Google App Engine, поэтому Swing GUI, в силу того, что он не нужен и, очевидно, не поддерживается GAE был полностью удален из системы. Ручной ввод координат пользователем был заменен на случайно сгенерированный (координаты и частота – в заданных пределах). Также, ручное создание и удаление агентов пользователем легло на плечи выделенного агента, о котором уже шла речь выше, а сами операторы с заданной вероятностью “умирают”. Помимо этого вместо классов библиотек 2d и 3d геометрии, необходимых для позиционирования на карте, были использованы самописные аналоги.

Агенты-перевозчики были немного доработаны и теперь с некоторой вероятностью временно выходят из строя или совсем выходят из системы – в данном случае это отражение того, что реальные транспортные средства, участвующие в грузоперевозке могу становиться недоступными – например, в пересменок, обеденный перерыв у водителя, или будучи на техническом обслуживании. Или же их деятельность может иным способом выводить их за рамки ответственности системы. Также была добавлена возможность передавать задания из очереди для транспортных агентов, которые вышли из строя. Для этого они тоже получают обновление состояния других перевозчиков через топик. Для выбора агента-заместителя опять же используется Contract Net.

5 Заключение

В рамках этой были выполнены практически полностью все поставленные цели и достигнуты следующие результаты (код доступен на Google Code[5]):

- Исследована архитектура JADE.
- Проанализированы наиболее популярные существующие облачные сервисы и обоснован выбор Google App Engine для осуществления интеграции с JADE.
- Изучено API Google App Engine и выявлены потенциальные проблемы.
- JADE перенесено в облако GAE, однако в данный момент работоспособен только вариант с размещением одного контейнера в платформе.
- На базе существующей MAC реализована демонстрационная система и запущена на получившейся платформе.

В настоящий момент уже ведется работа по организации сообщения по IMTP через HTTP. Помимо доработки этой части планируется повысить уровень взаимодействия разработчика с платформой в облаке путем создания полноценного Web GUI либо же организации связи с существующим GUI на локальной машине.

[5] https://code.google.com/p/jade-to-appengine-port/

Список литературы

[1] Д.Ю. Бугайченко, И. П. Соловьев. Абстрактная архитектура интеллектуального агента и методы ее реализации / / Системное программирование, 2005

[2] Amazon Web Services http://aws.amazon.com/

[3] Höfer, C.N. and Karagiannis, G. *Cloud computing services: taxonomy and comparison.* Journal of Internet Services and Applications, 2 (2). pp. 81-94. ISSN1867-4828, 2011

[4] Domenico Talia. Cloud Computing and Software Agents: Towards Cloud Intelligent Services // *volume 741 of CEUR Workshop Proceedings, 2011*

[5] F. Bellifemine, A. Poggi, G.Rimassa. Developing Multi-agent Systems with JADE // Intelligent Agents VII Agent Theories Architectures and Languages, 2001

[6] FIPA http://fipa.org/specs

[7] Force.com http://salesforce.com

[8] GoGrid http://gogrid.com

[9] JADE http://jade.tilab.com/

[10] JADEX http://jadex-agents.informatik.uni-hamburg.de/

[11] Jason http://jason.sourceforge.net/

[12] Cougar http://www.cougaar.org/

[13] A. Birukou, E. Blanzieri, P. Giorgini. A multi-agent system that facilitates scientific publications search / / Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems, ACM, 2006

[14] *A.H. Keyhanipour, M. Piroozmand, B. Moshiri, Lucas*. A multi-layer/multi-agent architecture for meta-search engines / / (AIML-05), Cairo, Egypt , 2005

[15] T.Harris, Cloud Computing Services – A comparison, 2010

[16] Rackspace http://rackspace.com

[17] SPADE https://pypi.python.org/pypi/SPADE

[18] Windows Azure. http://www.windowsazure.com/en-us/

[19] Google App Engine, Amazon Web Services and the Cloud. http://consultingblogs.emc.com/jaddy/archive/2010/04/09/google-app-engineamazon-web-services-and-the-cloud.aspx

[20] Wooldridge M.J. An Introduction to Multiagent Systems. Wiley, 2009.

[21] KQML http://www.csee.umbc.edu/csee/research/kqml/

[22] K. P. Sycara, "Multiagent systems," AI Magazine, vol. 19, no. 2, 1998.

[23] Google App Engine. https://developers.google.com/appengine/

[24] *H. L. Cardoso,* Integrating Jess and Jade, 2007

http://jade.tilab.com/doc/tutorials/jade-jess/jade_jess.html

[25] BDI4JADE http://www.inf.ufrgs.br/~ingridnunes/bdi4jade/

[26] http://paginas.fe.up.pt/~eol/AAMAS_07_08/aamas0708.html

[27] http://hpcru.wordpress.com/2012/04/23/