УДК 004.75

Р. Ю. ЛОПАТКИН[1], Н. В. ЛЫСАК[2], С. А. ПЕТРОВ[3], В. А. ИВАЩЕНКО[1]

1. Інститут прикладної фізики НАН України, м. Суми
2. Вінницький національний технічний університет, м. Вінниця
3. Сумський державний університет, м. Суми

ПРОТОТИП МУЛЬТИ АГЕНТНОЙ РАСПРЕДЕЛЕННОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

Аннотация. В данной статье предложена архитектура мультиагентной вычислительной сети, использующей ресурсы персональной вычислительной техники (персональных компьютеров, ноутбуков, планшетов, смартфонов). Особенностью данной мультиагентной сети является отсутствие центрального узла управления, что позволяет повысить ее отказоустойчивость и отсутствие монополизации ресурсов компьютера являющегося ее элементом. Вычислительная сеть построена по принципу сервис-ориентированного подхода, в котором основным элементом является агент предоставляющий сети определенный сервис. Это позволяет гибко интегрировать новый функционал в систему за счет добовления нового сервиса путем имплементации новых агентов и описание их поведения. Предложенная структура мультиагентной вычислительной сети и описаны основные агенты и протоколы их взаимойдствия друг с другом. Разработан унифицированный интерфейс представления задач для данной вычислительной сети.
Ключевые слова: Мультиагентная система, распределенные вычисления, сервисно-ориентированная архитектура
Аннотація. У данній статті запропонована архітектура мультиагентної обчислювальної мережі, яка використовує ресурси персональної обчислювальної техніки (персональних комп’ютерів, ноутбуків, планшетів, смартфонів), Особливістю мережі є відсутність центрального вузлу керування задачами, що значно підвищує її відмово стійкість і окрім цього відсутня монополізація обчислювальних ресурсів відповідного комп’ютеру, що є елементом цієї мережі. Мультиагентна обчислювальна мережа побудована за принципом сервіс-орієнтованого проектування, в якій основним елементом є агент що надає мережі певний сервіс Даний підхід дозволяє легко розбудовувати мережу шляхом додавання в неї інших агентів та імплементування його взаємодії з іншими агентами. Також запропоновано загальний шаблон представлення обчислювальних задач.
Ключові слова: Мультиагентна система, розподілені обчислення, сервіс-орієнтована архітектура.
Abstract. In the current paper proposed architecture of multiagent computational network that uses resources of personal computing devices (personal computers, notebooks, tablets, and smartphones). A feature of this multi-agent network is the absence of a central control unit, which allows it to increase fault tolerance and lack of monopolizing computer resources by its element. The computer network is built on the service-oriented approach principle the main element is the agent who offers a service to network. This gives a flexibility to integrate new features into the system by add new service through the implementation of new agents and a description of their behavior. The proposed structure of the multi-computer network, and describes the main agents and their protocols. We design the unified interface presentation tasks for this computer network.
Keywords: Multi-agent system, distributed computations, service-based modeling.

Введение

В последнее время активно развиваются облачные и грид-технологии, в том числе для удовлетворения потребностей ученых в вычислительных ресурсах [1]. Однако в последнее время наблюдается тенденция удешевления вычислительных ресурсов, громадные ресурсы персональной вычислительной техники также потенциально могут быть задействованы для этих целей [2]. К такой персональной вычислительной технике, относятся настольные персональные компьютеры (далее - ПК), ноутбуки, планшетные компьютеры, мобильные телефоны.

Вычислительные ресурсы можно приобретать как сервис и при этом работа с этими ресурсами проста и не требует каких-то специальных действий или умений. При этом нет необходимости покупать дорогостоящее серверное оборудование, содержать квалифицированные кадры для его обслуживания, выделять отдельно помещение со специальными системами кондиционирования, электропитания, оповещения о недопустимых событиях (например, резкое повышение температуры из-за выхода из строя системы кондиционирования). Ресурсы можно покупать даже для вычисления любой отдельной задачи.

С другой стороны, если с подобной легкостью можно было бы объединять для вычислений существующие ресурсы ПК обычных пользователей, то это позволило бы привлечь громадные дополнительные вычислительные ресурсы.

Для оценки этих ресурсов был проведен мониторинг одного из сегментов локальной сети Института прикладной физики НАН Украины, в который входит одиннадцать компьютеров. На каждом компьютере был автоматически запущен агент мониторинга, который каждые 10 секунд передавал сервису мониторинга данные о загрузке процессора и памяти. Чтобы набрать достаточно данных для статистики мы наблюдали за исследуемой локальной сетью одну неделю, чтобы охватить рабочие и выходные дни.

В результате эксперимента было выявлено, что уже имеющиеся мощности ПК загружены в среднем примерно на 1%. Соответственно неполную загрузку можно расценивать как потерю ценных ресурсочасов. Согласно прогнозу аналитической компании Forrester Research, в 2008 году количество работающих ПК в мире превысило отметку в 1 миллиард. В 2015 году аналитики прогнозируют появление уже 2 миллиардов только самих ПК [6].

Очень важно, что использование персональных компьютеров для создания вычислительной сети автоматически решает вопросы специализированного обслуживания, они не сконцентрированы в пределах одного помещения и не требуют отбора тепла, замена и ремонт стоят значительно дешевле. Но остается не решенным вопрос создания программного обеспечения, не требующих значительных усилий и специальной подготовики для настройки, объединить группу ПК в одну вычислительную сеть.

Актуальность

Например, Институт прикладной физики НАН Украины имеет кластер из 10 двухпроцессорных четырехядерных нодов, что позволяет одновременно решать 80 независимых задач. Но на балансе самого института находится около 150 персональных компьютеров общей производительностью превышающей возможности вычислительного кластера. На основе данных мониторинга можно с уверенностью утверждать, что на базе персональных компьютеров возможно построение целой инфраструктуры для решения достаточно больших вычислительных задач как научного, так и производственного плана. Но к такой системе с другой стороны выдвигаются дополнительные требования. К ним можно отнести адаптацию и сохранение работоспособности в условиях динамического изменения доступных ресурсов: во-первых, система должна минимально влиять на удобство работы владельца машины, во-вторых, любой персональный компьютер в сети может быть выключен в любой момент и это не должно привести к краху системы.

Таким образом, задача утилизации вычислительных мощностей персональных компьютеров является крайне актуальной. Для её решения необходима разработка гибких, сервис-ориентированных, самонастраивающихся, отказоустойчивых программных систем, которые позволили бы осуществлять менеджмент ресурсов персональной компьютерной техники с целью их использования в качестве вычислительных инструментов: в том числе и для решения больших и сложных задач.

Такие попытки были сделаны достаточно давно, однако в основном нацелены на решение какой-то конкретной задачи [3, 4]. Общей особенностью подобных систем является то, что участники, которые передали в общее пользование свой персональный компьютер, не решают свои задачи. Они скачивают и устанавливают себе на компьютер программный модуль (одинаковый для всех), который получает от главного сервера определенную порцию данных для расчета и по окончании вычислений возвращает серверу результат. Таким образом, подобные системы нельзя назвать многопользовательскими и они не пригодны для решения рассматриваемой в работе проблемы.

Описание системы

Объектом нашего исследования является многопользовательская вычислительная система, которая способна разворачиваться на базе локальных и глобальных гетерогенных (неоднородных) компьютерных сетей [2]. Под неоднородной сетью понимается сеть произвольной топологии с различными каналами связи, состоящая из компьютеров различной конфигурации и мощности, на которых установлены различные операционные системы.

Такая вычислительная сеть предназначена для предоставления сервисов по обработке и хранению данных многих пользователей. Для проектирования системы был выбран агентный подход и сервис-ориентированную архитектуру. Поэтому исследуемая система далее именуется агентной вычислительной системой (АВС).

Данная архитектура обусловлена необходимостью обеспечения:

- **масштабируемости системы** - компоненты системы взаимодействуют друг с другом удаленно, без необходимости знать месторасположение друг друга в системе, а исходя из их возможностей (доступных ресурсов, например);

- отказоустойчивости системы - система должна сохранять работоспособность даже при выходе из лада или отключения любого ее компонента;

- адаптивность - система может решать сколько копий задачи запускать. Это оказывает значительное влияние на скорость вычислений при различных условиях в зависимости от вероятности потери задачи (например, из-за отключения от сети машины, на которой считалась эта задача). Система должна уметь подбирать оптимальное значение этого параметра.

Архитектура системы

Основная особенность архитектуры разрабатываемой нами системы состоит в том, что ней **нет планировщика задач как отдельно выделенного компонента**. Процессом вычисления задач занимается сообщество агентов, которое совместно балансирует нагрузку на аппаратные мощности. За вычисление каждой задачи отвечает отдельный агент именуемый вычислителем. Это мобильный агент, способный перемещаться между машинами системы и перемещать за собой код своей задачи. Цель этого агента - провести задачу из состояния "запланирована" в состояние "посчитана" за минимально возможное время ($t_{calc} \rightarrow min$). Причем этот агент отвечает не только за то, чтобы найти машину, на которой в определенный момент есть необходимое количество доступных вычислительных ресурсов, но и сам управляет процессом вычисления его задачи, а не делегирует эту роль каким-то другим компонентам системы. Это возможно благодаря тому, что все задачи, которые отправляются в систему должны соответствовать заданному нами стандарту.

Таким образом, агенты путем переговоров делят доступные ресурсы и используют их для вычисления задач.

Подробная схема переходов между возможными состояниями этого агента изображена на рис. 1.

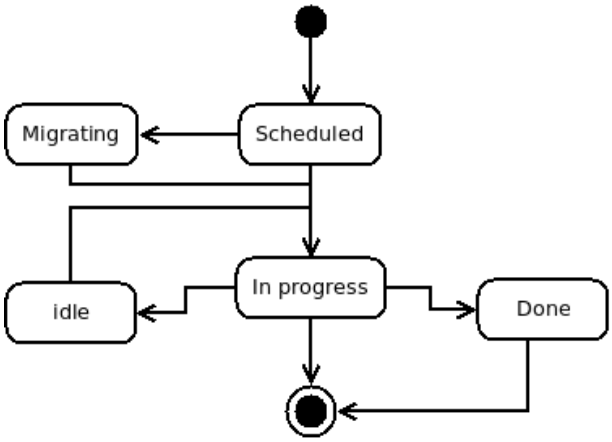

Рисунок 1 – Жизненный цикл Агента-вычислителя

Первое состояние вычислителя - задача и ее входные данные созданы и она готова к вычислениям, далее он переходит в состояние вычисления задачи в случае наличия необходимых ресурсов или в состояние поиска необходимых ресурсов. Как только ресурсы будут найдены, агент перейдет в состояние вычисления задачи. Во время вычисления этот агент может переходить в состояние ожидания (в случае нехватки вычислительных ресурсов). Вычислитель переходит в состояние «готово», когда есть валидный результат вычисления задачи, а затем переходит в финальное состояние, после чего уничтожается. Он может также перейти в финальное состояние из состояния вычисления задачи, если произошла ошибка во время вычислений или истек таймаут вычисления задачи.

Для обеспечения футкционирования такой системы разработана много уровневая архитектура системы, которая представлена на рисунке 2.

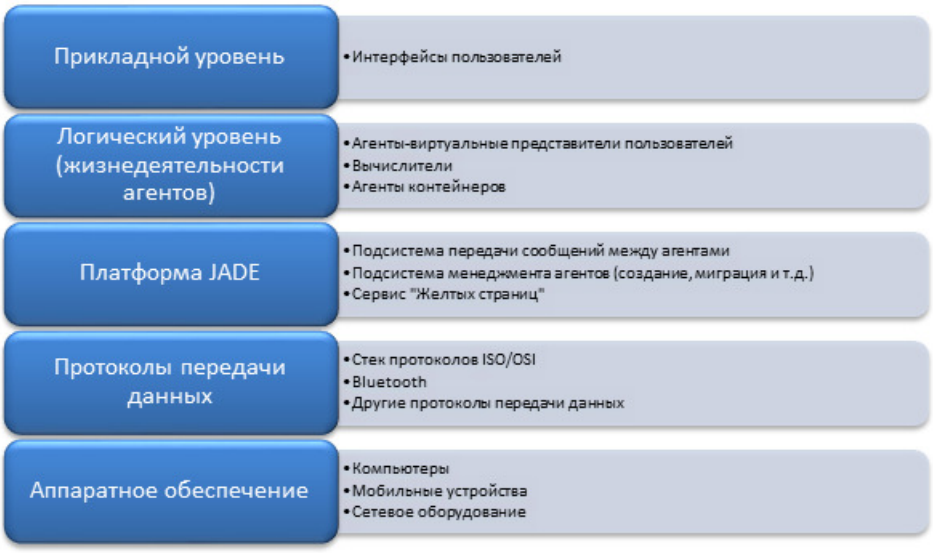

Рисунок 2 – Архитектура вычислительной системы

Как видно из рисунка, она представляет собой иерархическую пятиуровневую систему. Самый базовый уровень – Аппаратный, который состоит из компьютеров и связывающего их сетевого оборудования. Второй уровень – Транспортный, обеспечивающий логическую связь между компьютерами. Третий уровень служит базой для построения агентной вычислительной сети. Им может быть любая платформа, которая дает возможность работать с такими сущностями, как агенты и обеспечивать общение между ними. Примером такой платформы является Jade [5]. Следующий уровень системы содержит логику работы агентов, обеспечивает основные сервисы, такие как решение задач, хранение данных, интерфейсы пользователя, авторизацию и т.п. Разработка основных алгоритмов работы агентов этого уровня как раз и является целью данного исследования. Последний, пятый уровень, является сервисным с точки зрения потребителя (пользователя) и дает возможность получить услуги по расчету необходимых задач.

Техническая реализация системы

Нами был разработан действующий прототип системы.

Система состоит из контейнеров, в которых “живут” агенты. На каждой машине запущено ровно по одному контейнеру. Контейнер представляет собой отдельный процесс. В нашем случае менеджментом контейнеров и агентов полностью занимается агентная платформа jade. Т.е. она выступает как слой, на котором базируется наша система. Агент технически реализован как отдельный поток, к нему нет прямого доступа, но на него могут влиять другие агенты (но только путем отправки ему сообщений) и окружающий его мир - например, настройки, ограничения или доступные свободные ресурсы машины, на которой он находится в данный момент. Составной частью каждого агента есть разработанные нами специальные компоненты, инкапсулирующие в себе и предоставляющие агенту логику взаимодействия с системой.

Такими компонентами являются: **AgentsManager** - менеджер агентов, **PlatformManager** - менеджер платформы. **ServicesManager** - менеджер сервисов. Менеджер агентов предоставляет агенту возможность созавать или удалять других агентов (но удалять он может только созданных им агентов). Менеджер сервисов позволяет агенту регистрировать в системе свои сервисы, причем агент может регистрировать и дерегистрировать свои сервисы “на лету”.

Стоит заметить, что агент может выполнять все действия, предоставляемые менеджерами и без их помощи, напрямую отправляя сообщения агентам, ответственным за менеджмент агентной платформы. Но выделение этих логических компонентов в архитектуре системы упрощает ее разработку и поддержку. Тем более если учитывать, что эти менеджеры доступны всей иерархии агентов.

Агенты взаимодействуют друг с другом исключительно посредством отправки сообщений друг другу. Они обмениваются сообщениями посредством протокола MTP (message transport protocol), который передает данные поверх популярных протоколов передачи данных, таких как tcp/ip, bluetooth и другие. Все сообщения агентов - сообщения стандарта FIPA-ACL (agent communication language). За передачу сообщений между агентами отвечает сервис передачи сообщений (MTS или message transport service), работающий по стандарту FIPA MTS [7]. Сообщения, которыми обмениваются агенты, оформлены согласно их онтологиям. Т.е. представлены в виде структурированных данных, которые можно валидировать и извлекать знания из них (ontology and content languages).

Стоит отметить, что участником вычислительной сети может стать клиент, не обязательно использующий фреймверк jade. Главное условие - реализация стандартов, по которым взаимодействуют компонеты системы (агенты).

Для миграции агента необходимо сохранить его текущее состояние и данные, и создать агента такого же типа на другой машине с таким же состоянием и данными. Поэтому технически миграция агента - это сериализация экземпляра его Java-класса на одной машине, перемещение и десериализация на другой.

Иерархия агентов

Для реализации логического и прикладного уровня были созданных классы агентов, иерархия которых представлена на рис. 3.

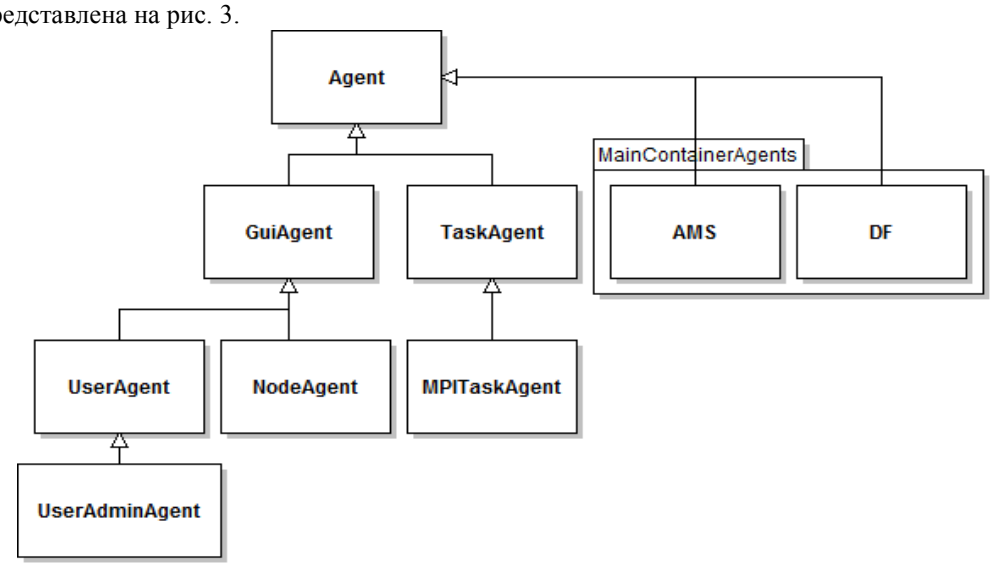

Рисунок 3 – Иерархия агентов

В предложенной системе присутствуют три ключевых вида агентов:

− **Представители пользователей в системе (UserAgent или сокращенно UA)** – агенты, которые предоставляют пользователям интерфейсы для работы с системой. Он является виртуальным представителем пользователя в системе. Может мигрировать в другой контейнер при выключении машины.

− **Вычислители задач(TaskAgent или сокращенно TA)** – агенты, которые производят вычисления задач пользователей. Такой агент создается под каждую новую задачу. На протяжении своего жизненного цикла этот агент находит машину, пригодную для расчета задачи, затем мигрирует (естественно только по согласию NodeAgent-а) и производит расчет задачи и потом ожидает определенный период времени пользователя чтобы отдать ему результаты вычислений, затем самоуничтожается. Этот агент привязан к UserAgent-у конкретного пользователя.

− **Менеджеры нодов (NodeAgent или сокращенно NA)** – агенты, которые управляют ресурсами компьютеров, на которых они запущены. На каждой машине обязательно существует ровно один экземпляр такого типа агентов. Менеджеры нодов предоставляют Вычислителям задач и Представителям пользователей возможность использовать ресурсы своей машины. Причем обнаружение необходимых ресурсов происходит через специальные сервисы менеджеров нодов. Также менеджер нода отвечает за сбор и передачу по требованию статистики о доступном аппаратном обеспечении компьютера и его загруженности. Также предоставляет информационные сервисы о количестве TaskAgent-ов в своем контейнере.

Если говорить о вычислителе задач, то его задачу можно представить в виде вектора: $t = <C,HDD,RAM>$, где C - количество тактов процессора, HDD - объем винчестера, RAM - объем оперативной памяти, необходимых для вычисления задачи. Компьютер можно представить в виде вектора: $pc = <C,HDD,RAM>$ - мощность процессора, доступные объемы постоянной и оперативной памяти. Стоит отметить, что на текущий момент система предоставляет только вертикальное масштабирование: если TaskAgent-у не хватает ресурсов, то может мигрировать на машину, на которой хватает. Но может быть ситуация, что в системе просто нет машины с достаточным количеством ресурсов. Такую ситуацию можно решить только с помощью горизонтального масштабирования. Например, в случае нехватки памяти - использовать вместе память нескольких машин. Отсюда вытекает идея, что для горизонтального масштабирования TaskAgent должен создавать N копий себя и эти копии должны взаимодействовать друг с другом для решения одной общей задачи. Взаимодействие здесь означает, например, синхронизацию распределенной памяти - в случае масштабирования по памяти.

Менеджеры нодов не имеют возможности мигрировать, а Представители и Вычислители - мобильные агенты которые могут мигрировать (согласно своей внутренней логике на более подходящую с их точки зрения машину) - на машину с достаточным количеством ресурсов для выполнения своих заданий.

Всю систему можно представить в виде:

$S = (T_a, U_a, C_a, A_a, T, C)$ - множества: вычислителей, представителей пользователей, менеджеров нодов, вспомогательных агентов, задач и компьютеров соответственно.

Каждый агент реализован в виде отдельного класса языка java.

Класс Agent – базовый для всех агентов системы класс. Содержит в себе готовый функционал по созданию, удалению, миграции агентов. UIAgent – агент с графическим интерфейсом.

Также в системе присутствуют вспомогательные типы агентов.

UserAdminAgent – агент администратора системы. Предоставляет возможность конфигурировать систему, администрировать пользователей системы, а также предоставляет сервис авторизации в системе.

DebugAgent – агент для отладки работы системы.

AMS и **DF** – стандартные агент системы, автоматически запускаются при запуске главного контейнера. AMS (система управления агентами) обеспечивает службу имен (например, гарантирует, что каждый агент на платформе обладает уникальным именем) и осуществляет управление на платформе (к примеру, можно создавать и ликвидировать агентов на удаленном контейнере по запросу AMS). DF – (координатор каталог) обеспечивает сервис «желтых страниц», с помощью которого агенты могут искать друг друга по описанию предоставляемых сервисов.

Интерфейс пользователя

В разработанном прототипе реализованы все интерфейсы пользователя, через которые он может отправлять свои задачи в вычислительную сеть и получать из неё результаты расчетов. Работа происходит в несколько этапов. Сначала пользователь пишет программный код для решения своей задачи согласно предоставленному шаблону или использует уже готовые библиотеки, содержащие код решения для его задачи. Затем пользователь загружает входные данные и отправляет через своего Представителя программный код в систему. Представитель создает Вычислителя под данную задачу, и он уже действует самостоятельно: ищет подходящий Контейнер, рассчитывает задачу и по окончании вычислений переходит в состояние ожидания, когда пользователь запросит результаты вычислений. По истечении определенного времени (выставляется администратором АВС) Вычислитель самоуничтожается.

Настройки системы

Пользователь, как участник вычислительной сети, может задавать ограничение по ресурсам, которые он отдает системе. В частности это ограничения по количеству используемой памяти, по загрузке процессора и по количеству используемых ядер процессора. Для этого у пользователя существует специальная форма в его интерфейсе.

Эти настройки сохраняются в системных настройках машины и ими руководствуются все агенты живущие в ней постоянно или мигрировавшие в нее для достижения своих целей. Например, Node Agent определяет количество TA, которые могут одновременно считать свои задачи на его машине, исходя из максимального количества ядер, которые разрешил использовать пользователь, TA также загружает ядро процессора не больше, чем ему разрешил это делать пользователь.

Авторизация пользователя и интерфейс для работы с системой

Для каждого реального пользователя в системе автоматически создаётся его виртуальный представитель - User Agent (UA). Этот агент живет в системе не зависимо от того авторизован пользователь или нет.

Когда пользователь запускает систему на своей машине, в ней автоматически создается Node Agent, который показывает ему форму для авторизации. Далее пользователь вводит логин и пароль и Node Agent (NA) использует эти данные, чтобы авторизовать пользователя. Для этого NA в сервисе "желтых страниц" находит всех агентов, предоставляющих сервис авторизации и передает введенные в этой форме данные для авторизации пользователя. В случае если какой-то из UA принял авторизационные данные, он мигрирует на машину пользователя и отображает пользователю интерфейс для работы с системой, иначе - будет выдано сообщение об ошибке. Именно через этот интерфейс пользователь получает доступ для работы с системой: отправляет задачи в систему для вычислений, просматривает их статус, сохраняет результаты вычислений на жесткий диск. Пользователь видит список своих задач и может скачивать результаты. Скачивание происходит следующим образом: пользователь отправляет запрос своему UA на скачивание задачи через интерфейс пользователя, UA направляет запрос соответствующему TA, а TA, получив такой запрос, мигрирует на компьютер пользователя и сохраняет результаты на жесткий диск.

Посчитанная задача существует в системе определенное время, после которого автоматически уничтожается не зависимо от того забрал пользователь результаты вычислений или нет.

Интерфейс администратора системы

Администратор начинает развертывание системы путем запуска главного контейнера системы, к которому потом подключаются контейнеры, созданные пользователями.

В своем интерфейсе администратор системы создает всех пользователей системы, задает им права. При создании пользователей в системе для каждого пользователя создается UserAgent - виртуальный представитель пользователя, который затем мигрирует на машину пользователя (в случае успешной авторизации) и предоставляет пользователю интерфейс работы с системой.

Кроме создания системы, администратор видит загруженность всей системы, а также распределение агентов по системе. Также администратор может задавать глобальные настройки системы такие как лимит времени на вычисление одной задачи, лимит времени до самоуничтожения задачи после того, как она будет посчитана. Он может применять новые настройки только для новых задач или для новых и уже существующих.

Шаблон задачи

Каждая задача, которая отправляется в систему, должна соответствовать специальному шаблону. Это сделано для того, чтобы TaskAgent мог унифицированным образом управлять вычислением своей задачи и его код не приходилось модифицировать под каждую отельную задачу. Кроме того, использование шаблона решает другие важные задачи, такие как контроль нагрузки на машину, на которой осуществляются вычисления, инициализация задачи для вычислений, сбор и сохранение результатов вычислений.

Чтобы удовлетворить все перечисленные требования, задача должна быть реализована в виде отдельного класса, который реализует заданный интерфейс (Task) и соответственно методы этого интерфейса: *oneStep()* - в этом методе должна содержаться вся логика, выполняемая на одной итерации вычислений, *getResult()* - возвращает строку, содержащую результат вычислений, *actionBefore()* - в этом методе необходимо выполнять все действия по инициализации задачи перед вычислениями, *actionAfter()* - после.

Когда TaskAgent, имеющий в своем распоряжении задачу, оформленную по описанному шаблону получает необходимые вычислительные ресурсы, он производит все действия необходимые для вычисления задачи начиная от инициализации и оканчивая сбором результатов вычислений и отправкой их пользователю. Если для получения необходимых вычислительных ресурсов TaskAgent-у необходимо мигрировать на другую машину, то он мигрирует вместе со своей задачей. При этом агент перемещает

скомпилированный код своей задачи в виде байт массива. После миграции TA записывает код своей задачи на жетский диск.

Сервисы агентов

Так как система сервис-ориентированная, в ней агенты предоставляют друг другу сервисы. Агенты узнают о том, какой агент предоставляет необходимый им сервис посредством глобального для всей системы сервиса желтых страниц. Изначально всем агентам известно каким образом получить доступ к сервису желтых страниц (за это ответственна агентная платформа). Агенты могут регистрировать свои сервисы как при создании, так и регистрировать/дерегистрировать сервисы во время своей жизни. Как уже говорилось ранее, за это отвечает разработанный нами менеджер сервисов агента.

Система состоит из трех основных типов агентов, каждый из которых предоставляет свое множество сервисов. К списку основных сервисов агентов системы можно отнести следующие сервисы.

Сервисы TA: **готов считать** - агент регистрирует себя в этом сервисе, чтобы его могли извещать другие агенты о появлении доступных ресурсов.

Сервисы NA: **готов предоставить ресурсы для вычислений** - сервис предоставляет информацию о доступных вычислительных ресурсах и в момент, когда все доступные ресурсы машины заняты TaskAgent-ами хотя бы по одному параметру (например, все ядра процессора отданы под вычисления), NA дерегистрирует себя как представителя этого сервиса пока какой-то из TA не закончит вычисления и соответственно появятся доступные ресурсы.

Сервисы UA: **сервис авторизации пользователей** - поиск UA для авторизации пользователей в системе производится только через этот сервис.

Таким образом, благодаря сервис-ориентированной архитектуре, агенты могут с легкостью, например, обнаруживать доступные ресурсы системы и запрашивать их под свои потребности и происходит это без необходимости знать где конкретно находятся эти ресурсы, что способствует масштабируемости системы.

Переговоры агентов за использование ресурсов

Как было сказано выше, TA должен управлять процессом вычисления его задачи. Для этого он должен договариваться с NA о вычислениях на их машинах. TA выбирает для вычислений машину подходящую по ресурсам для вычисления его задачи. Инициаторами переговоров могуть быть как TA, так и NA. Это сделано для снижения нагрузки на систему, т.к. можно было бы реализовать этот функционал путем периодического опроса всех NA о появлении свободных ресурсов всеми TaskAgent-ами, но при этом была бы громадная нагрузка на систему.

Алгоритмы Вычислителя при создании

Когда создается TA, он сразу совершает поиск агентов, предоставляющих сервис “**готов предоставить ресурсы для вычислений**”. См. рис. 4.

Рисунок 4 – Поиск вычислителем свободных ресурсов при его создании

Если есть агенты, предоставляющие необходимые ресурсы, то TA сразу же запрашивает необходимые ему ресурсы мигрирует на машину одного из них (наиболее приоритетного) для вычислений. Иначе,

ТА остается в на той машине, на которой он был создан и пассивно ожидает участия в аукционах проводимых NA за право использования ресурсов.

Критерий выбора Task-агентов на аукционе

Когда у NA появляются свободные ресурсы (что происходит когда какой-то из TaskAgent-ов на его машине оканчивает вычисления), то он должен известить всех агентов, предоставляющих сервис "готов считать" об этом. Но естественно, что этих ресурсов может не хватить на всех агентов, которым они необходимы, поэтому NA должен выбрать кому достануться эти ресурсы. Для этого NA объявляет аукцион для выбора ТА, которым будет предоставлено право посчитать задачу в его контейнере.

Каждый Task-агент присылает организатору аукциона вектор, описывающий его задачу.

$T = <T_{create}, CPU_{est}, RAM_{est}>$ У организатора аукциона есть эталонный вектор T $T' = <T'_{create}, CPU'_{est}, RAM'_{est}>$. Он перемножает все присланные ему векторы (скалярное произведение) на свой эталонный вектор и выбирает победителем аукциона того агента, для которого получилось максимальное произведение. Эталонный вектор организатора аукциона можно варьировать для обеспечения победы выгодных ему(организатору) агентов, что может обеспечить достижение определенной задачи, например, максимально загружать машину NA.

Схема переговоров ТА с NA за ресурсы показана на рис 5.

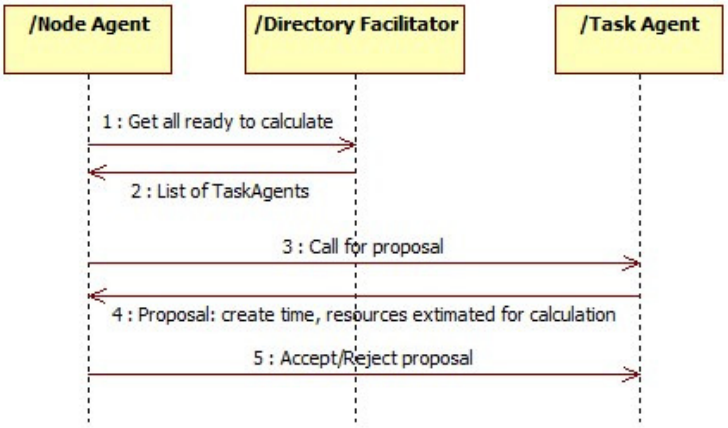

Рисунок 5 – Переговоры агентов за ресурсы

Выводы

Была разработана архитектура системы для построения многопользовательских вычислительных сетей и в соответствии этой архитектуре была осуществлена ее техническая реализация. За основу была взята платформа Jade, которая одновременно обеспечивает функционал компонентов мультиагентных систем и взаимодействие ее участников (агентов) в рамках сервис-ориентированного подхода.

В дальнейшем планируется

1. Добавить возможность не только вертикального, но и горизонтального масштабирования. Для этого необходимо будет распараллеливать задачи вручную или автоматически. Для автоматического распараллеливания можно использовать статистический анализ кода.

2. Автоматизировать создание шаблона, чтобы пользователь не набирал весь код класса, реализующий задачу вручную, а мог сосредоточиться на самой задаче и вводил только код, связанный непосредственно с логикой вычислений.

Список литературы

5. Uwe Schwiegelshohna, Rosa M. Badiab, Marian Bubak Perspectives on grid computing //Future Generation Computer Systems – 2010. – Volume 26, Issue 8. – P. 1104–1115.

6. Азаров О.Д. / Комп'ютерні мережі: навчальний посібник / [Азаров О.Д., Захарченко С.М., Кодун О.В. та ін] - Вінниця: ВНТУ, 2013. -371 с. ISBN 978-996-641-543-4.

7. SETI@home - Search for ExtraTerrestrial Intelligence at home [Электронный ресурс].Режим доступа: http://setiathome.berkeley.edu/ - оглавление с экрана.

8. Rosetta@home - Protein Folding [Электронный ресурс]. Режим доступа:http://boinc.bakerlab.org/rosetta/ - оглавление с экрана.

9. Jade - Java Agent DEvelopment Framework [Электронный ресурс]. Режим доступа: http://jade.tilab.com/ - оглавление с экрана.

10. Worldwide PC Adoption Forecast, 2007 To 2015 [Электронный ресурс]. Режим доступа: https://www.forrester.com/go?objectid=RES42496 - оглавление с экрана.

11. FIPA Agent Message Transport Service Specification (SC00067) [Электронный ресурс]. Режим доступа: http://www.fipa.org/specs/fipa00067/SC00067F.html.

Стаття надійшла:15.10.15.

Відомості про авторів

Лопаткін Роман Юрійович - к.ф.-м.н., доцент, завідуючий Науково-дослідним центром навчально-наукових приладів ІПФ НАН України, вул. Петропавлівська, 58, м. Суми, Україна, тел. (0542) 604538.

Лисак Наталія Володимирівна – к.т.н., доцент кафедри менеджменту та безпеки інформаційних систем Вінницького національного технічного університету.

Петров Сергій Олександрович – к.т.н., старший викладач кафедри комп’ютерних наук Сумського державного університету.

Іващенко Віталій Анатолійович – молодший науковий співробітник сектору телекомунікацій та грід-технологій іпф НАН України, вул. Петропавлівська, 58, м. суми, україна, тел. (0542) 604538.