инженерия программного обеспечения

УДК 004.891.3

Косякин Антон Николаевич

CIRROCUMULUS* – СИСТЕМА УПРАВЛЕНИЯ IT-ИНФРАСТРУКТУРОЙ

Аннотация

С ростом сложности компьютерных систем повышаются затраты на их поддержку и администрирование. Хотя это правило очевидно и давно всем известно, до определённого момента оно остаётся незамеченным. Современные программные комплексы, а в особенности высоконагруженные веб-проекты и другие распределённые компьютерные системы содержат большое число логических компонентов и физических узлов. Системному администратору, поддерживающему такой проект, необходимо отслеживать все происходящие в нём события, оперативно на них реагировать, зачастую в режиме 24/7. На каком-то этапе жизненного цикла проекта человек перестает справляться, поэтому появляется необходимость в автоматизации этих работ. В статье проводится краткий обзор подходов к решению указанной задачи и рассматривается мультиагентная система Cirrocumulus, предлагающая новый подход к администрированию сложных распределённых компьютерных систем.

Ключевые слова: управление IT-инфраструктурой, экспертные системы, мультиагентные системы, облачные технологии.

ВВЕДЕНИЕ

Когда администратор поддерживает систему, имеющую большое количество компонент и физических узлов, главной сложностью является число происходящих в ней событий, а также количество метрик, так или иначе характеризующих производительность и работоспособность инфраструктуры в целом. Без специализированных средств автоматизации человек может оказаться не способным держать в голове все нюансы работы поддерживаемой им системы и оперативно реагировать на возникающие исключительные ситуации. В качестве ключевых задач, которые должен решать подобный инструментарий, можно выделить наблюдение (мониторинг) за инфраструктурой и управление её отдельными компонентами.

Проект Cirrocmulus был разработан для автоматизации поддержки распределённых компьютерных систем и IT-инфраструктур. Он основан на мультиагентной архитектуре и представляет собой набор агентов, запускаемых на отдельных узлах. Каждый из них поддерживает набор определённых операций и настроен на сбор конкретных фактов о работоспособности обслуживаемого им модуля или узла системы. В зависимости от используемых алгоритмов, агент может обмениваться данными с другими агентами и на основе консолидированной в его памяти

* *Cirrocumulus* – перисто-кучевые облака. Более тонкие облака, состоящие из очень мелких волн, хлопьев, ряби.

информации принимать решения о своих дальнейших действиях. Таким образом, Cirrocumulus позволяет решать задачи мониторинга и управления системой или IT-инфраструктурой.

Текущие существующие решения по наблюдению за системой (такие как Zabbix, Nagios, Munin) предлагают администратору на все подконтрольные физические узлы устанавливать специализированные сервисы-агенты, которые постоянно собирают заранее запрограммированную информацию о работоспособности узла и отсылают её на центральный сервер. Далее, используя удобную панель управления, человек может централизованно отслеживать состояние и работоспособность всех компонентов, смотреть историю изменения различных параметров и на основе полученных данных предпринимать какие-либо действия.

В дополнении к описанным выше существуют системы по автоматическому развёртыванию и настройке отдельных физических узлов. Опять же, на каждый узел инфраструктуры предлагается устанавливать специализированный агент-сервис, принимающий команды от центрального сервера. Он может заниматься установкой необходимого ПО, настройкой уже имеющегося (путём, допустим, исправления конфигурационных файлов) и выполнять какие-либо разовые операции на узле. Самыми популярными представителями подобных систем являются Chef и Puppet.

Комбинируя указанные типы систем, можно автоматизировать большое количество задач по поддержке инфраструктуры: в случае отклонения каких-либо заданных параметров от нормы, система мониторинга может отправлять уведомление администратору. Он, впоследствии, будет отдавать соответствующие команды для перезапуска вышедших или выходящих из строя программных компонент или даже физических узлов. Или же система мониторинга может автоматически отправлять простые команды на необходимые узлы. Например, в случае небольшого веб-проекта, где СУБД запущена на отдельном сервере и настроена репликация, система мониторинга может отслеживать доступность основного сервера БД и, в случае его сбоя, автоматически переконфигурировать приложение на работу с работоспособной репликой. На файл-сервере можно также отслеживать результаты самодиагностики жёстких дисков и в случае отклонения от нормы отсылать уведомление администратору, параллельно подключая резервный диск.

На практике, к сожалению, данный подход имеет ряд недостатков. Прежде всего, стоит отметить ограничения на объем обрабатываемых в каждый момент времени данных и взаимосвязи между ними.

В некоторых системах возникающие события могут быть взаимосвязаны друг с другом. Допустим, в результате отказа одного из узлов размещённая на нём СУБД перестанет отвечать на входящие запросы и, как следствие, откажут зависящие от этой базы данных другие компоненты системы. Для отслеживания подобных ситуаций системы мониторинга дают возможность настроить набор триггеров, срабатывающих в случае, если некоторые параметры контролируемого узла достигают определённых значений. Далее эти триггеры можно скомбинировать в более сложные, используя логические операторы «И» и «ИЛИ». При их срабатывании – вызывать внешние программы.

В описанном выше примере сложные триггеры, не имея возможности оперировать полной картиной происходящего в системе, будут срабатывать лишь как индикаторы последствий возникшей проблемы, а не информировать администратора о её первоначальных причинах. Подобного поведения можно было бы избежать, если бы описанные в подсистеме мониторинга триггеры могли динамически подстраиваться под происходящие события: в какие-то определённые моменты, опираясь на полную информацию о состоянии системы, отключаться или игнорировать, а затем – включаться обратно.

Другим примером неполноты обрабатываемых данных являются параметры сработавшего триггера. Допустим, администратор хочет отслеживать загрузку центрального процессора на всех узлах системы и при пре-

вышении определённого порога – каким-либо образом получать уведомления. Фактически, триггер представляет собой булеву функцию, возвращающую значения «истина» или «ложь» от конечного набора параметров, характеризующих состояние системы в данный момент. Поэтому, в случае его срабатывания, мы можем знать только лишь об этом одном факте: «загрузка процессора превысила допустимый порог». Данный факт не содержит в себе информации о том, какое в момент срабатывания было значение загрузки центрального процессора. Подобную информацию необходимо явно запрашивать у системы мониторинга из внешней программы, обрабатывающей возникшую исключительную ситуацию.

С одной стороны, описанные выше проблемы имеют решение за конечное число дополнительных шагов, и в некоторых случаях можно смириться с возросшей сложностью настройки системы мониторинга. Но, с другой стороны, не стоит забывать, что большинство систем мониторинга являются централизованными. Поэтому располагаемые на узлах агенты-сервисы являются пассивными, а все действия происходят на одном единственном узле, создавая на него дополнительную нагрузку и порождая также увеличение времени реакции. При обслуживании инфраструктуры с большим числом узлов, а также тесными взаимосвязями между компонентами, количество генерируемых событий в моменты какого-либо сбоя может скачкообразно расти, и нагрузка на центральный сервер спровоцирует его отказ, оставив администратора вообще без какой-либо информации о происходящем.

Использование мультиагентной архитектуры позволяет проекту Cirrocumulus решить описанные выше проблемы. Основная особенность такого подхода – отсутствие централизованности. Каждый агент, согласно заложенным в него алгоритмам, сам занимается мониторингом своего (или соседних) узла, реагирует на отклонения. Тем самым в случаях серьезных сбоев инфраструктуры нагрузка на систему мониторинга распределяется между несколькими агентами (часть агентов может собирать первичную информацию, консолидировать её и перенаправлять дальше, но уже в меньшем объёме). Даже если это и приведёт к их отказу, в целом система скорее всего останется работоспособной. Другой важной характеристикой мултиагентного подхода является проактивность. Агент (в рамках заложенных алгоритмов) может собирать информацию о собственном узле, окружающих агентах и на основе обобщенной картины принимать решения о своих дальнейших действиях. Вместе с этим, во время выполнения агентом каких-либо действий, всегда имеется полный доступ к собранным об окружающем «мире» фактам, поэтому последовательность и список операций могут корректироваться во время работы. Подобное поведение архитектурно не заложено в обычные системы мониторинга и поэтому сложно реализуемо на их основе.

1. CIRROCUMULUS

Проект Cirrocumulus, помимо использования мультиагентной архитектуры, имеет ещё одну отличительную особенность – он позволяет интегрироваться с системами мониторинга и управления узлами добавлением между ними модуля принятия решений на основе текущего (полного) состояния контролируемой системы. Как уже было сказано ранее, Cirrocumulus представляет собой набор агентов, запускаемых на отдельных узлах. Имея собранные факты о работоспособности обслуживаемого модуля системы, в зависимости от используемых алгоритмов, каждый агент может дополнять свои знания о «мире», общаясь с другими агентами. Получив более полное представление о происходящих событиях, он может принимать решения о своих дальнейших действиях.

Модуль принятия решений Cirrocumulus'а можно рассматривать как некую экспертную систему. При внедрении системный администратор, исходя из собственного опыта и особенностей автоматизируемой системы, составляет набор метрик с требуемым уровнем детализации характеризующих каждый модуль или компонент. Затем фиксируется (и реализуется в программном

коде) список допустимых действий, которые можно осуществлять в системе, и составляется база знаний: описывается реакция (оперируя реализованными ранее действиями) Cirrocumulus'а на возникновение конкретных ситуаций в ходе работы системы. Конкретные ситуации, в свою очередь, определяются комбинацией значений ранее собранных метрик. Таким образом, уровень автоматизации администрирования подконтрольной компьютерной системы значительно повышается по сравнению с использованием традиционного мониторинга. При правильной организации процесса также появляется возможность вести журнал срабатывания модуля принятия решений и впоследствии анализировать его для поиска узких мест в архитектуре и реализации подконтрольной инфраструктуры.

Разработка подобной интеллектуальной системы с нуля сопряжена с определёнными сложностями. Прежде всего, необходимо решить низкоуровневые задачи. К ним относятся способ коммуникации между агентами, формат обмениваемых сообщений. Также важно зафиксировать способы адресации, возможные типы коммуникативных актов (команд). При этом, исходя из области применения Cirrocumulus, её основыми пользователями будут простые системные администраторы. Поэтому, решая инфраструктурные задачи и подбирая инструменты, необходимо сохранить простоту подключения к команде разработчиков других людей, ранее не принимавших участия в подобных проектах. К счастью, в начале 2000-х годов Институт инженеров по электротехнике и электронике (IEEE) разработал набор стандартов FIPA (Foundation for Intelligent Physical Agents). Он описывает базовые принципы взаимодействия агентов друг с другом, включая в себя структуру и способы сериализации сообщений, типы коммуникативных актов и даже способы расширения протокола для поддержки специфичных возможностей. Именно на этих стандартах и основывается Cirrocumulus. Конечно, полноценная реализация FIPA достаточно трудоёмка, ведь стандарт включает в себя несколько десятков спецификаций и покрывает множество общих случаев. На текущем этапе проекту требуется лишь небольшое подмножество стандарта, поэтому строгое следование ему не является одной из приоритетных задач.

1.1. МУЛЬТИАГЕНТНОЕ ПРОГРАММИРОВАНИЕ

На данный момент, мультиагентное программирование набирает популярность как интересный подход к построению автономных компьютерных систем. Уже разработано большое количество платформ и инструментов, предоставляющих среду выполнения для мультиагентных приложений, а также наборы низкоуровневых библиотек для разработчиков конечных агентов. Преследуя попытки предоставить стандартизированные и кроссплатформенные окружения, большинство подобных систем разрабатываются на языке программирования Java. Наверное, самой известной из них является разработка компании Telecom Italia под названием JADE. JADE – фреймворк с открытым исходным кодом, также следующий стандартам FIPA и предоставляющий разработчику основу для создания мультиагентных приложений, решая за него низкоуровневые задачи типа коммуникаций между агентами.

Однако стоит признать, что язык Java позволяет реализовать только некоторые аспекты агентно-ориентированного программирования, в то время как другие аспекты, такие как принятие решений, потребуют использования внешних инструментов. Со стороны этот факт может показаться несущественным, но на практике, чем проще система и чем меньше знаний необходимо для начала её использования, тем продуктивнее становится труд программистов.

Давайте вспомним термин «агент» – программный элемент, будучи помещённым в среду, реагирует на изменения этой среды и вырабатывает последовательность действий, пытаясь достичь определённой цели. Агент также может взаимодействовать с другими агентами, если подобное взаимодействие помогает в достижении поставленной цели. С точки зрения разработчика, данное определение подразумевает под собой три необ-

ходимых условия, которым должна удовлетворять среда выполнения (и язык) для построения мультиагентных систем:

1. *Возможность описания и реализации реактивного поведения агента.* В большинстве случаев решается использованием конечного автомата.

2. *Возможность описания и реализации модуля принятия решений* на основе таких подходов к построению искусственного интеллекта, как экспертные системы, системы обработки правил и т. д.

3. *Возможность взаимодействия с другими агентами,* например обмен сообщениями.

Императивные и объектно-ориентированные возможности Java ограничиваются только реализацией конечных автоматов и коммуникаций, не предоставляя никакой возможности описать интеллектуальную составляющую агента. Этот факт обуславливается отсутствием таких логических конструкций языка, как предикаты или декларативные правила.

Хорошим кандидатом для построения подобных систем является разработанный компанией Ericsson язык Erlang, благодаря возможности сопоставления с образцом (pattern matching) на уровне самого языка. Идея сопоставления с образцом заключается в том, что при написании программного кода условия входа в функцию или блок операторов описываются в виде набора конкретных значений некоторых переменных. В случае если указанные переменные совпадают с образцом, происходит присвоение всем переменным указанных значений, а затем совершается вход в блок операторов. Рассмотрим несколько простых примеров.

```
is_valid_signal(Signal) ->
  case Signal of
       signal, _What, _From, _To} ->
            true;
       {signal, _What, _To} ->
            true;
       _Else ->
            false
    end.
```

В данном случае, если переменная *Signal* представляет собой структуру, состоящую из терма signal и двух или трёх других полей, то сигнал считается правильным. Во всех других случаях – нет.

```
fact(N) when N>0 ->
    N * fact(N-1);
fact(0) ->
    1.
```

Здесь описывается функция *fact*, которая при вызове её с параметром 0 возвращает 1, а при вызове с любым другим положительным числом в качестве параметра, вычисляет его факториал. Конечно, подобные конструкции можно реализовать на Java или другом императивном языке программирования. Но скорее всего это повлечёт за собой либо существенное усложнение синтаксиса, либо введение фазы препроцессинга исходного кода для его последующего преобразования в компилируемый вариант. Пошаговое выполнение программы в целях отладки при таком подходе в большинстве случаев становится невозможной. Ярким примером использования языка Erlang для построения мультиагентных систем является система eXAT, разработанная в Университете Катании (Италия). Однако функциональная природа и относительная непопулярность самого Erlang увеличивают сложность вхождения в подобную систему для многих разработчиков. Именно поэтому для реализации Cirrocumulus был выбран разработанный в 1995-м году японским программистом Юкихиро Мацумото язык Ruby. Данный язык является кроссплатформенным, полностью динамическим и объектно-ориентированным. Объектно-ориентированный подход и особенности синтаксиса Ruby позволяют, несмотря на его императивность, без дополнительной трансляции реализовать на нём предметно-ориентированный язык программирования (DSL), подходящий к использованию для построения модуля принятия решений. Рассмотрим небольшой фрагмент исходного код (листинг 1).

В данном примере задаётся правило под названием *build_virtual_disk*, срабатывающее если ему на вход будет передана структура вида *:virtual_disk, :NUMBER, :actual_state, :created,* где по аналогии с Erlang строки, состоящие из строчных букв,

являются термами, а остальные – переменными. Однако, как видно из кода, автоматического присвоения значений переменным не происходит и осуществляется вручную. Но это является разумной платой за простоту синтаксиса и возможность использования привычной парадигмы программирования.

Обобщая сказанное выше, видим, что в целом архитектура Cirrocumulus представляется таким образом:

1) на каждом узле контролируемой системы запускается отдельная копия агента;

2) в зависимости от типа узла, агент поддерживает конкретную *онтологию:* собирает заранее фиксированный набор фактов об этом узле и позволяет выполнять над ним лишь определённые действия;

3) для каждой онтологии может быть описан набор правил, принимающих на вход комбинацию значений фактов (предикат от состояния узла) и выполняющих определённый набор действий в случае срабатывания;

4) агенты имеют возможность общаться друг с другом, обмениваясь собранными фактами и запрашивая выполнение каких-либо действий.

1.2. ОНТОЛОГИИ

Взаимодействие нескольких агентов осуществляется не только использованием одинаковых типов сообщений, но также и разделением между ними общих представлений об «окружающем мире». Другими словами, они должны использовать одну *онтологию*. Именно по этой причине в стандартах FIPA указано, что название онтологии является обязательной частью каждого передаваемого сообщения. Для этой цели Cirrocumulus позволяет разработчику описывать онтологии, используя концепции объектно-ориентированного программирования (основная слабая сторона языка Erlang). Строго говоря, никаких ограничений на структуру класса онтологии не накладывается. Единственное условие – необходимость реализации метода, вызываемого ядром системы при получении входящего сообщения, адресованного данному агенту (и данной онтологии). Сообщение, в зависимости от типа коммуникативного акта, может содержать какой-либо факт, переданный другим агентом, либо запрос на выполнение определённой операции. В случае если был получен новый факт, он добавляется в базу знаний агента, и запускается процесс поиска правил, необходимых к выполнению. Все такие правила добавляются в специальную очередь выполнения и обрабатываются последовательно в фоновом потоке, тем самым не блокируя работу агента по приёму новых запросов. Если же исходное сообщение содержало запрос на выполнение определённого действия, управление передаётся в соответствующий модуль системы, и по завершении – происходит уведомление о статусе выполнения запрошенной операции.

Именно онтологии и являются той комбинацией систем мониторинга и управления, о которых было рассказано ранее. Они содержат информацию о всех собираемых метриках подконтрольного узла, способы сбо-

Листинг 1

```
rule 'build_virtual_disk', [ [:virtual_disk, :NUMBER, :actual_state,
                                        :created] ] do |engine, params|

    disk_number = params[:NUMBER]

    Log4r::Logger['cirrocumulus'].info "Building Virtual Disk #{disk_number}"

    disk = VdsDisk.find_by_number(disk_number)

    engine.ontology.build_disk(disk)

  end
```

ра этих метрик и в дополнение – набор действий, которые возможно осуществлять над этим узлом. Хранимая же в оперативной памяти база фактов является основой экспертной системы, позволяющей автоматизировать сложные операции по администрированию и поддержке.

В качестве примера можно рассмотреть работу хостинг-провайдера, предоставляющего услуги аренды виртуальных серверов. Подобная онтология будет оперировать такими понятиями, как *виртуальный сервер*, *виртуальный диск, хост-сервер (сервер, на котором выполняется запуск виртуальных серверов)*. Типичными действиями для такой онтологии будут:

– создание нового виртуального сервера;

– запуск и останов виртуального сервера;

– создание и подключение виртуальных жестких дисков к серверу;

– изменения конфигурации виртуальных серверов (увеличение выделенных ресурсов, изменение настроек сети и т. д.).

Отслеживаемыми событиями в таком случае являются:

– внезапный останов виртуального сервера и последующий его запуск как решение проблемы;

– выход из строя хост-сервера и перезапуск всех расположенных на нём виртуальных серверов в другом месте;

– миграция виртуальных серверов для обеспечения равномерной загрузки хост-серверов.

1.3. СЛОЖНЫЕ ВЗАИМОДЕЙСТВИЯ

В зависимости от требований, выполняемые агентом действия могут быть достаточно сложными и требовать для реализации конечные автоматы, состоящие из большого числа состояний и переходов. Возможны ситуации, когда подобные случаи невозможно полноценно реализовать на этапе разработки, до запуска системы в эксплуатацию. Широко известно, что декомпозиция на более мелкие подзадачи помогает разработчику в подобных ситуациях и также способствует переиспользованию отдельных частей кода в других мультиагентных приложениях. Для таких случаев Cirrocumulus вводит понятие *саг* – последовательности действий, растянутых во времени и включающих в себя коммуникации с другими агентами. По своей структуре и реализации сага похожа на онтологию – от разработчика требуется реализовать метод обработки входящего сообщения, являющегося частью текущей коммуникации. Помимо этого, также добавляется понятие таймаута – алгоритм может запросить от ядра системы уведомить его по истечении определённого промежутка времени. Подобный механизм используется при опросе доступных агентов о возможности совершения ими какого-либо действия. Агенты, не ответившие в течение определённого периода времени, исключаются из дальнейшей коммуникации.

Саги запускаются основной онтологией как реакция на входящий запрос и выполняются в фоновом режиме, не блокируя агента для принятия новых команд. При этом, метод определения адресата сообщения, благодаря поддержке стандартов FIPA, остаётся очень простым: в каждом сообщении указывается имя агента, название используемой онтологии и идентификатор коммуникации, который сопоставляется (совпадает) с идентификатором запущенной саги.

2. ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ

Первоначальная цель разработки системы Cirrocumulus – реализация API услуги аренды виртуальных серверов в облаке. Пользователям предоставляется возможность аренды вычислительной мощности и некоторого объёма дискового пространства для хранения данных по требованию.

Разработанная система выполняет две основные задачи:

1. Предоставление высокоуровневого API для создания и запуска виртуальных серверов, а также для выделения требуемого объёма дискового пространства в хранилище данных (создание и подключение к серверам виртуальных дисков).

2. Мониторинг, обслуживание и поддержание работоспособности инфраструктуры данного облака и всех его компонент.

На низком уровне инфраструктура облака представляет собой набор серверов трёх типов:

1. *Управляющий сервер,* на котором запущены основные сервисы: DHCP и TFTP для организации локальной сети, шина обмена данными между агентами и непосредственно управляющий агент.

2. *Система хранения данных.* На таких серверах располагаются виртуальные диски (данные виртуальных машин). В рабочем прототипе для обеспечения балансировки нагрузки и сохранности данных, используется два таких сервера. Доступ к данным осуществляется по протоколу ATA Over Ethernet (AoE) в изолированной локальной сети.

3. *Хост-узлы,* на которых происходит непосредственный запуск виртуальных машин. Данные сервера не имеют жестких дисков, и загрузка ОС происходит по локальной сети (используя сервис TFTP на управляющем сервере). В прототипе используется набор из трёх подобных серверов.

Для каждого из описанных типов серверов была выделена собственная онтология.

Cirrocumulus-storage

Основное понятие данной онтологии – виртуальный жёсткий диск. Агент позволяет создать диск заданного объёма и анонсировать его хост-узлам для дальнейшего использования. Там виртуальные диски, анонсированные с нескольких серверов, объединяются в RAID-массив (зеркало). Тем самым не только повышается сохранность пользовательских данных, но и немного увеличивается скорость доступа к ним.

Помимо входящих запросов на выделение дискового пространства, данная онтология также занимается мониторингом дисковой подсистемы. Каждый сервер хранения данных содержит по 6 жёстких дисков, объединённых в общий массив. Каждые несколько минут агент производит мониторинг их основных параметров: текущая температура, показатели самодиагностики, общее время работы (в часах) каждого диска. На основании собранных метрик агент делает предположение о надёжности дисковой подсистемы и посылает системному администратору уведомление с предложением заменить выходящий из строя жёсткий диск.

Cirrocumulus-xen

Название этой онтологии основывается на названии гипервизора, выбранного для реализации проекта: Xen. Соответственно, такие агенты занимаются управлением данным гипервизором и запуском виртуальных машин. На вход в онтологию поступает конфигурация создаваемого или запускаемого сервера: количество ядер процессора, объём оперативной памяти, количество сетевых карт, список подключенных виртуальных жёстких дисков. В случае если на сервере есть свободные ресурсы в нужном количестве, происходит запуск сервера и его дальнейший мониторинг. Агент постоянно следит за состоянием запущенных виртуальных машин. В случае если любая из них выключилась (не было запроса на выключение), в шину обмена данных направляется уведомление о данном факте.

Также агент постоянно следит за работоспособностью виртуальных дисков: копии каждого из них анонсируются с нескольких серверов хранения данных и собираются в RAID-массив. Необходимо проверять состояние каждой копии и при сбое любой из них (например, выключение сервера хранения данных) бить тревогу и запускать процесс синхронизации данных после устранения подобной ошибки (повторного «появления» анонсированной копии). Если виртуальный диск становится полностью не доступен, для предотвращения нарушения целостности пользовательских данных соответствующий виртуальный сервер принудительно выключается.

Cirrocumulus-cloud

Центральная онтология, реализованная в проекте. Такой агент имеет непосредственный доступ к базе данных, в которой хранится информация обо всех существующих виртуальных серверах, виртуальных дисках и их состоянии: некоторые серверы могут быть выключены пользователем, некоторые должны быть запущены. На основании этих данных агент заполняет свою базу знаний фактами о желаемых статусах виртуальных машин. Затем, происходит периодических опрос соседних агентов (cirrocumulus-xen) о

фактическом статусе соответствующих машин. В случае если они не совпадают, запускаются процессы корректировки состояния: либо принимается решение о запуске выключенного сервера или, наоборот, о принудительном останове соответствующей виртуальной машины. Также происходит обработка поступающих сообщений о незапланированном выключении серверов.

Если процесс остановки виртуальной машины или изменения её конфигурации обычно выполняется за один шаг, то процесс запуска представляет собой полноценный конечный автомат:

– у соседних агентов выясняется, не запущен ли уже где-то искомый сервер;

– в случае отрицательного ответа происходит опрос свободных ресурсов на каждом из хост-узлов;

– в зависимости от политики распределения ресурсов (равномерное распределение ресурсов по хост-узлам или последовательное заполнение каждого из них) выбирается узел, на котором будет происходить запуск; таким образом заодно решается задача введения в кластер новых хост-узлов: при старте новых серверов свободные узлы будут задействованы в первую очередь;

– соответствующему агенту отправляется запрос на подключение необходимых виртуальных дисков; этот агент, в свою очередь, коммуницируя с агентами онтологии cirrocumulus-storage, подключает к своему узлу диски и собирает их в RAID-массив;

– происходит непосредственно запуск виртуального сервера с заданной конфигурацией.

В случае сбоя на каком-либо из шагов, агент отсылает в шину обмена данными (ответ направляется непосредственно отправителю запроса) информацию о причинах неудачи.

Стоит отметить, что, несмотря на распределённую мультиагентную инфраструктуру, в описываемой системе присутствует центральный узел (онтология cirrocumulus-cloud и сервисы, запущенные на этом узле), реализующий ключевые алгоритмы. Однако на практике сбой этого узла не приводит к фатальному нарушению работоспособности системы: на некоторое время нарушаются некоторые функции мониторинга текущего состояния и игнорируются все входящие запросы на изменение этого состояния. После восстановления работоспособности центрального сервера вся инфраструктура без дополнительных действий продолжает свою работу в полном объёме.

Для снижения времени восстановления после глобального сбоя в проекте Cirrocumulus предусмотрена функция восстановления состояния агента. Каждый агент имеет собственную локальную базу данных, в которой хранит свою текущую базу знаний: список виртуальных жёстких дисков, которые необходимо анонсировать, список запущенных виртуальных серверов и т. д. В случае перезапуска после возникшего сбоя, агент старается восстановить своё предыдущее состояние, не дожидаясь запросов от управляющего агента. Тем самым, кратковременные выходы отдельных агентов из строя зачастую могут вообще не влиять на работоспособность всей инфраструктуры облака.

ЗАКЛЮЧЕНИЕ

В целом описанная система довольно функциональна, сохраняя разумную простоту построения конечных агентов и, тем самым, пригодна для автоматизации администрирования разного рода распределённых компьютерных систем. На данный момент проект является открытой разработкой, и его исходный код доступен любому желающему. Сама система успешно внедрена в несколько компаний, занимающихся предоставлением разнообразных хостинг-услуг, а также просто обладающих сложной внутренней IT-инфраструктурой. И хотя текущая реализация Cirrocumulus является пилотной версией, проект успешно себя зарекомендовал в каждом из внедрений. Также в процессе эксплуатации обнаружились более интересные функциональные требования, которые будут реализованы в последующих версиях системы. Например, если в базе знаний агентов описать зависимости между возможными событиями в инфраструктуре, исполь-

зуя математический аппарат теории вероятностей, можно определять вероятную причину сбоя и использовать это знание для определения дальнейших действий агента. В качестве примера можно привести тот факт, что сбой жёстких дисков сервера, на котором располагается СУБД, может привести к замедлению или полной остановке её работы. Сильный наплыв посетителей на web-сайт может спровоцировать большое количество запросов к базе данных и также замедлению её работы, что, в свою очередь, является причиной увеличения времени обработки запросов от пользователей. Используя теорию Байесовских сетей, агент может сделать предположение о причинах увеличения времени отклика системы на запросы и быстрее предпринять какие-либо действия, не дожидаясь явного проявления проблемы.

Литература

1. Zabbix – ultimate open source availability and performance monitoring solution / http://www.zabbix.com/product.php (дата обращения 25.02.12).

2. Chef – an open-source systems integration framework built specifically for automating the cloud / http://wiki.opscode.com/display/chef/Architecture+Introduction (дата обращения 25.02.12).

3. Java Agent DEvelopment Framework / http://jade.tilab.com/ (дата обращения 25.02.12).

4. *Corrado Santoro.* eXAT: Software Agents in Erlan / 11th Erlang User Conference – Stockholm, Nov. 10, 2005.

5. Foundation for Intelligent Physical Agents. FIPA ACL Message Structure Specification. No. SC00061G, 2002.

6. Foundation for Intelligent Physical Agents. FIPA Communicative Act Library Specification. No. SC00037J, 2002.

Abstract

With growth of computer systems, their administration and support costs also raise. It is an obvious and well-known rule, but it remains unnoticed until a certain moment.

Modern software systems, in particular high-loaded web projects or other distributed systems, contain large number of logical and physical components. System administrator who supports such system, has to trace all occurring events, quickly react on them and sometimes do this in 24/7 mode. At some stage of the project a human becomes unable to manage such a complexity and need of some kind of automation arises. This article provides a brief overview of the approaches to solving this problem and describes multi-agent system Cirrocumulus, which offers a new approach to the administration of complex distributed computer systems.

Keywords: IT-infrastructure management, expert systems, multi-agent systems, cloud technologies.

Косякин Антон Николаевич,
аспирант кафедры системного программирования математико-механического факультета СПбГУ,
архитектор в компании Artec BPO,
anton@tinuviel.ru