УДК 519.687.1

ДИНАМИЧЕСКОЕ УПРАВЛЕНИЕ РЕСУРСАМИ ВИРТУАЛЬНЫХ ИНСТРУМЕНТОВ НА ВЫЧИСЛИТЕЛЬНОМ КЛАСТЕРЕ

А. А. Московский[1], **А. Ю. Первин**[1], **B. J. Walker**[2]

Технология виртуальных машин (ВМ) обеспечивает значительную гибкость в задачах распределения ресурсов. Как правило, нагрузка на приложения меняется с течением времени и, как следствие, меняются и потребности этого приложения в наращивании или высвобождении ресурсов этого приложения. Именно такие возможности предлагают ВМ. Разработана система для автоматического динамического управления аппаратными ресурсами приложений, работающих на вычислительном кластере. В задачи среды входит обеспечение надлежащего уровня сервиса приложения в допустимом интервале путем изменения доступных приложению ресурсов. Для эффективного управления ресурсами могут быть использованы профили приложений, собранные средствами нагрузочного тестирования. Разработанное программное обеспечение позволяет развертывать и управлять сервисами внутри виртуальных машин, которые могут быть запущены на нескольких компьютерах (узлах вычислительного кластера) одновременно. Разработаны и протестированы следующие приложения: вычислительный сервис и веб-сервис. Составлены профили этих приложений и изучены зависимости между производительностью приложений и ресурсами. Представлены промежуточные результаты исследования, направленного на изучение вопросов динамического управления ресурсами с использованием теории оптимального управления и методов оптимизации. Статья подготовлена по материалам доклада авторов на международной научной конференции “Параллельные вычислительные технологии” (ПаВТ-2008; http://agora.guru.ru/pavt).

Ключевые слова: виртуальные инструменты, вычислительные кластеры, управление ресурсами, методы оптимизации, вычислительный сервис, веб-сервис.

1. Введение. Цель настоящего исследования состоит в разработке методов и средств управления приложениями, работающими на вычислительном кластере в виртуальной среде. При работе на традиционных кластерах количество вычислительных узлов, используемых тем или иным приложением, служит в качестве основной и естественной метрики потребления ресурсов. При использовании виртуальных машин (ВМ) также можно задействовать эту метрику для распределения ресурсов. Однако в дополнение к этому технология ВМ предлагает целый ряд новых возможностей по управлению ресурсами, которые сложно или невозможно реализовать при использовании традиционных компьютеров. В частности, виртуальную машину `Xen` [1] можно:

1) приостановить и сохранить ее состояние в памяти, понизив нагрузку на процессор для последующего запуска других приложений;

2) остановить и сохранить ее состояние на диске, предоставляя таким образом возможность использовать ресурсы другими, более приоритетными с точки зрения пользователя виртуальными машинами;

3) переместить приложение с одного физического компьютера на другой;

4) запустить приложение с некоторым числом процессоров, а затем добавлять или отнимать процессоры во время работы ВМ с учетом потребностей других ВМ;

5) запустить приложение с некоторой долей процессора и затем увеличивать или уменьшать эту долю исходя из потребностей приложений, работающих внутри ВМ;

6) запустить приложение с некоторым объемом оперативной памяти и затем, по мере необходимости, изменять этот объем;

7) запустить приложение с ограниченной сетевой пропускной способностью и динамически изменять этот параметр в зависимости от потребностей приложений ВМ.

[1] Институт программных систем РАН, 152020, Ярославская обл., Переславский район, местечко “Ботик”, Россия; e-mail: moskov@phys069b-2.chem.msu.ru; ArtemPervin@gmail.com

[2] Hewlett–Packard Laboratories, 1501, Page Mill Road, Palo Alto, CA 94301, United States; e-mail: bruce.walker@hp.com

Столь значительная гибкость настроек ВМ способствует оптимальному использованию ресурсов, поскольку можно выделять приложению ровно столько ресурсов, сколько ему требуется, повышая тем самым КПД аппаратных средств. Консолидация ВМ и автоматическое перераспределение ресурсов, основанное на загруженности приложений и их приоритетах, предоставляет ИТ-администраторам возможность обеспечивать корректную работу большего числа сервисов в рамках имеющейся инфраструктуры.

Использование технологий виртуализации вычислительных ресурсов актуально в том числе и для грид-среды, в которой ВМ позволяют значительно упростить задачу автоматизации распределения ресурсов и управления конфигурацией узлов грида [2, 3]. Однако на сегодняшний день многие преимущества ВМ до сих пор используются не в полной мере. Так, например, сейчас сравнительно мало систем, применяющих ВМ для эффективного потребления простаивающих мощностей компьютеров [4 – 6]. Нам представляется перспективной концепция предоставления части аппаратных ресурсов компьютеров различным сервисам с помощью ВМ. Для грид-систем, служащих вычислительной площадкой одновременно для многих приложений, критически важно иметь формализованные средства для автоматического распределения ресурсов. То же самое верно и для крупных центров обработки данных (ЦОД): время и внимание системного администратора может стоить дорого, а медленная реакция на события — привести к катастрофе.

Проекты в этом направлении активно ведутся как в коммерческих, так и в академических организациях. Среди них можно выделить инициативы `Amazon EC3` и `3Tera's Applogic` — широко известные коммерческие платформы, предназначенные для размещения сетевых сервисов на ВМ. Проект `Cluster-on-Demand` [7] предоставляет средства для создания виртуальных кластеров из ВМ. В проекте `Virtual Workspaces` ВМ используются в грид-среде для изоляции приложений от аппаратного окружения с помощью промежуточного программного обеспечения `Globus Toolkit` [8]. `SoftUDC` [9] — это платформа для "коммунальных вычислений", в которой виртуализируются такие ресурсы, как процессор, дисковая память и сетевая пропускная способность. При распределении ресурсов в виртуальном окружении иногда используются экономические модели. Так, например, в системе `Shirako` [10] предложен механизм, позволяющий приложениям и самой среде заключать контракты на аренду ресурсов, в то время как в проекте `Tycoon` [11] используется модель аукциона для распределения ресурсов.

Представляется возможным сделать следующий шаг вперед: системы автоматического управления ресурсами могут учитывать, какую ценность представляют выделенные приложению процессорные мощности, память или другие ресурсы при текущей пользовательской нагрузке. Имея эту информацию, можно точно определить, каким образом следует наращивать ресурсы, доступные приложению, и когда можно затребовать эти ресурсы обратно.

В ходе настоящего исследования разработано программное обеспечение, которое позволяет экспериментировать с различными схемами распределения ресурсов. При этом во внимание принимаются следующие предположения:

— любое приложение имеет набор параметров, которые однозначно определяют качество предоставляемого приложением сервиса с точки зрения конечных пользователей (например, время отклика для веб-сайта); при этом параметры могут быть измерены во время работы приложения;

— с помощью технологии ВМ аппаратные ресурсы могут назначаться приложениям динамически с достаточно высокой степенью точности и, как следствие, эти ресурсы могут рассматриваться как непрерывные величины.

С этими предположениями задача управления качеством сервиса приложения может рассматриваться как задача непрерывного оптимального управления. Такой подход значительно отличает наше исследование от аналогичных работ в области управления качеством сервиса [12 – 14]. В нашем подходе приложения рассматриваются как черные ящики, а среда времени исполнения может использовать мощные методы теории оптимального управления для реализации схем эффективного распределения ресурсов.

Для того чтобы автоматизировать процесс принятия решения о целесообразности тех или иных ресурсов для приложения, предлагается использовать абстракцию *уровень сервиса*. Среда времени исполнения может использовать информацию с датчиков, описывающих текущее состояние приложения, наряду с *профилем производительности* этого приложения для выработки алгоритма поддержки требуемого уровня сервиса приложения. В ситуации дефицита ресурсов система может отнимать ресурсы у менее важных с точки зрения пользователей приложений. При этом профили производительности могут быть составлены либо на испытательном стенде до запуска сервиса в эксплуатацию, либо непосредственно в процессе работы сервиса.

В рамках настоящего исследования ставится задача реализовать различные типы виртуальных инструментов (virtual appliances) и способы управления ими в условиях меняющейся нагрузки на эти инстру-

менты. Термин `Virtual Appliances` пока не имеет устоявшегося перевода на русский язык. В литературе встречается обозначение “шаблоны виртуальных машин”, однако, на наш взгляд, такой перевод недостаточно точно отражает оригинальный смысл этого термина. Мы используем термин “виртуальные инструменты”, чтобы подчеркнуть автономность пары “приложение” и “операционная система”. Виртуальным инструментом может быть как готовое к использованию приложение (веб-сайт), так и промежуточная компонента системы (СУБД).

Для исследования поставленных задач прежде всего необходима инфраструктура для развертывания, мониторинга и распределения ресурсов виртуальных инструментов. В следующем разделе описывается архитектура реализованной среды Виртуальные Сервисы, служащей в качестве такой инфраструктуры.

Затем необходимы модели или профили производительности приложений. Эта информация будет не только описывать оптимальный состав аппаратных ресурсов для заданной нагрузки, но также характеризовать эффект от добавления или изъятия тех или иных ресурсов у приложения. В разделе 4 описывается концепция профилей производительности. Наконец, необходим специальный регулятор, который бы использовал профили производительности и информацию времени исполнения для перераспределения ресурсов с целью оптимизации их использования.

Рис. 1. Компоненты системы Виртуальные Сервисы

2. Архитектура среды Виртуальные Сервисы. Для решения поставленных задач была реализована простая система, которая позволяет развертывать *виртуальные сервисы* — параллельные виртуальные инструменты. Виртуальные инструменты представляют собой либо сетевые сервисы (например веб-сайт), либо высокопроизводительные параллельные приложения, работающие внутри одной или нескольких виртуальных машин. Другими примерами таких инструментов могут быть законченные решения в виде сервисов сетевых игр, средств обработки данных и т.п. Для управления виртуальными машинами используется `Xen` — монитор виртуальных машин с открытым исходным кодом, однако принципы, заложенные в описываемой системе, могут быть перенесены на любую другую аналогичную по функциональности платформу. На рис. 1 представлены основные компоненты системы и схема их взаимодействия.

Типичная эксплуатация системы подразумевает наличие *Поставщика Сервиса*, который запускает

сервисы, и Пользователей, подключающихся к сервису через так называемою *точку входа* — пары IP-адреса и сетевого порта. Среда предоставляет, в частности, такие возможности:

— запуск и останов сервиса; запуск сервиса подразумевает запуск одной или нескольких виртуальных машин с соответствующим виртуальным диском (образом файловой системы), установку правил маршрутизации сетевого трафика и создание виртуальной сети из виртуальных машин, предназначенных для этого сервиса;

— выделение и высвобождение ресурсов сервиса; запросы на ресурсы могут быть инициированы вручную Поставщиком Сервиса или программно при помощи специальной компоненты — *планировщиком сервиса*;

— перенаправление сетевого трафика; эта функция необходима для того, чтобы обеспечить доступ пользователям к приложениям в виртуальной среде, а также для балансировки нагрузки.

Гибкость в управлении ресурсами ВМ играет ключевую роль в нашем исследовании. В то же время существенное значение имеет возможность использования разнообразных алгоритмов для управления всей системой в целом. В связи с этим применяется двухслойный механизм распределения ресурсов для разделения системного и сервисного слоев. На сервисном слое встраиваемый планировщик сервиса, учитывая данные мониторинга вычислительного узла, принимает решения о потребности в дополнительных или, наоборот, исключении неиспользуемых ресурсов для этого сервиса. На верхнем слое системный планировщик при поиске оптимального распределения ресурсов между сервисами учитывает договоренности между Поставщиками Сервисов и Администратором, выраженные в приоритетах сервисов. Кроме того, системный планировщик может использовать профиль производительности приложения в случае, если ему требуется оценить различные варианты распределения ресурсов. Оба планировщика способны взаимодействовать между собой и использовать необходимые им данные мониторинга. Система автоматически поддерживает объем ресурсов, доступный приложению и необходимый ему для обеспечения заданного уровня сервиса. В случае выхода из строя физического компьютера, на котором находится ВМ, среда автоматически создаст аналогичную ВМ на одном из свободных компьютеров. Для отслеживания таких ситуаций используются методики, применяемые в решениях “высокой доступности”.

3. Разработанные инструменты.

WebMapServer. Это приложение [15] позволяет запрашивать различную информацию по географическим картам. В тестах использовались данные по округу Итаска, штата Миннесота, США, полученные через Геологическую службу США. Отображаемые пользователем страницы содержат сгенерированные по запросу фрагменты карты в формате GIF.

X-Com. Вычислительный сервис основан на программном обеспечении `X-Com` [16], которое представляет собой систему метакомпьютинга, разработанную в МГУ им. М. В. Ломоносова. Система X-Com чем-то напоминает систему распределенных вычислений `Condor` [17], однако реализация X-Com значительно более компактна, менее требовательна к ресурсам и проще в установке и эксплуатации. Кроме того, система X-Com может работать в самых различных окружениях: вычислительные кластеры, федерации кластеров, грид-среды, совокупности гетерогенных процессоров, очереди задач и т.д.

Виртуальный кластер. Этот сервис позволяет запускать требуемое количество ВМ с поддержкой сетевого подключения между ними. В результате запуска этого сервиса создается набор виртуальных узлов — *виртуальный кластер*. Сетевая поддержка реализуется с помощью механизма bridging, который позволяет включать ВМ в виртуальный кластер абсолютно прозрачным для пользователя образом. В результате пользователи могут взаимодействовать с узлами виртуального кластера без какой-либо дополнительной настройки таким образом, как если бы это был обычный компьютер.

По результатам проведенных вычислительных экспериментов в виртуальном кластере было показано, что в таком окружении можно работать с полноценными MPI-приложениями, использующими несколько виртуальных узлов кластера одновременно. Кроме того, проведены успешные эксперименты по запуску параллельных программ, созданных с помощью средства быстрой разработки параллельных приложений `OpenTS` [18].

4. Профиль производительности. Профиль производительности приложения иллюстрирует зависимость между объемом ресурсов, предоставленных этому приложению, генерируемой на это приложение пользовательской активностью и уровнем сервиса, который обеспечивает это приложение пользователям. Объем ресурсов может быть выражен в абсолютных (например, 1 Гбайт оперативной памяти) или в относительных величинах (например, 53 % процессора). Пользовательская активность определяется для каждого приложения отдельно. Так, например, для веб-сайта пользовательская активность выражается количеством пользователей в секунду, обращающихся к этому сайту (частота запросов). Наконец, уровень сервиса может быть измерен как разница между желаемым (целевым) состоянием сервиса и его текущим

состоянием. Концепция уровня сервиса подробно рассматривается в следующем разделе

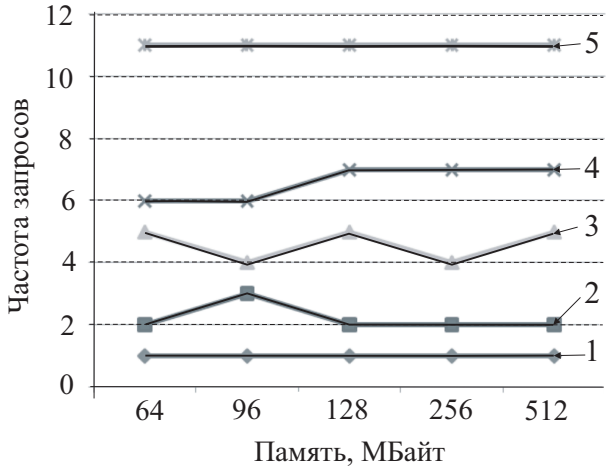

Рис. 2. Нагрузка на сервис WebMapServer при различных объемах памяти и долях процессора. Обозначения кривых CPU : 1) 10 %, 2) 30 %, 3) 50 %, 4) 70 %, 5) 100 %

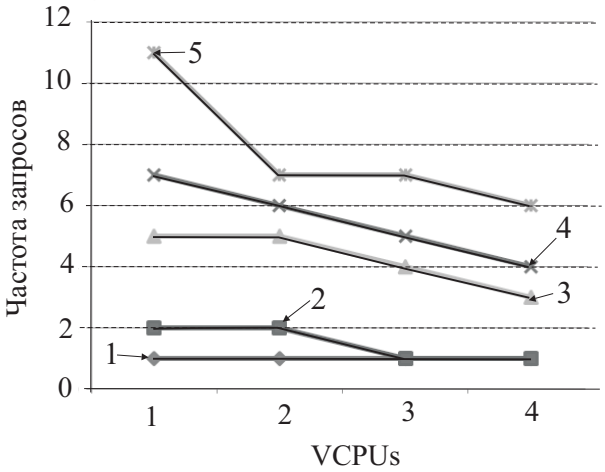

Рис. 3. Нагрузка на сервис WebMapServer при различном количестве VCPU и долей процессора. Обозначения кривых CPU : 1) 10 %, 2) 30 %, 3) 50 %, 4) 70 %, 5) 100 %

Эта зависимость может быть выражена в табличной форме. В этом случае в таблице описываются некоторые типичные сценарии использования приложения с разными уровнями пользовательской активности. С помощью методов интерполяции и экстраполяции могут быть получены значения, не вошедшие в таблицу. Данные для этой таблицы могут быть собраны с помощью утилит нагрузочного тестирования, например, `httperf` [19]. Полагаем, что при достаточном объеме данных в таблицах такой подход может дать хорошие результаты в задаче распределения ресурсов.

В ходе исследования были выполнены замеры производительности с различными уровнями пользовательской нагрузки и объемом ресурсов, выделенных сервису. В настоящий момент можно варьировать следующие параметры ВМ: объем оперативной памяти, число виртуальных процессоров, используемых ВМ (VCPU), и долю процессорного времени, определяющую максимальное значение (в процентах) процессорного времени физического компьютера, которое может занимать ВМ. Пользовательская нагрузка в каждом тестовом запуске задавалась таким образом, чтобы максимизировать использование ресурсов, предоставленных сервису, и в тоже время минимизировать число сетевых ошибок (таких, как, например, закрытие соединения по таймауту). Мы называем такие уровни нагрузки *точками перегиба*: это такая максимальная пользовательская нагрузка, которую сервис в состоянии обработать корректно.

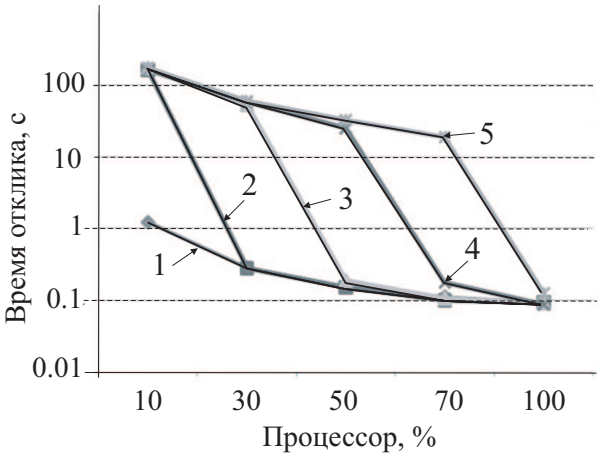

Рис. 4. Время отклика WebMapServer и доли процессора при различных уровнях нагрузки. Кривые, обозначающие частоту запросов: 1) 1, 2) 2, 3) 5, 4) 7, 5) 10

Профилировка производительности WebMapServer показала, что это приложение слабо чувствительно к объему оперативной памяти: было отмечено незначительное отклонение максимальной частоты запросов (рис. 2). Увеличение числа виртуальных процессоров не увеличило, а наоборот — снизило производительность (рис. 3). Такое поведение, очевидно, вызвано тем, что приложение является однопоточным и, как следствие, не использует дополнительные процессоры. В то же время параметр, отвечающий за долю физического процессора, предоставляемую ВМ, оказывал наибольшее влияние на производительность приложения. Зависимость между максимальной частотой запросов и долей процессора оказалась почти линейной.

Как можно видеть на рис. 4, при нагрузке в 10 запросов в секунду (кривая 5), уровень сервиса весьма чувствителен к доле процессорного времени: необходимо по меньшей мере 70 % процессорного времени

для обработки запросов за разумное время (около 1 секунды). В то же время нагрузка в один запрос в секунду может быть корректно обработана и при 10 % процессорного времени.

Профили производительности могут использоваться для оценки различных вариантов размещения ресурсов без непосредственного воздействия на производительность приложений, работающих в системе.

5. Соглашения об уровне сервиса. Рассмотрим ситуацию, когда владелец веб-сайта желает поддерживать среднее время отклика своего сайта ниже некоторого порогового значения (например, менее одной секунды). В случае если этот веб-сайт испытывает высокую пользовательскую нагрузку, могут потребоваться дополнительные вычислительные ресурсы для поддержания уровня сервиса в требуемом интервале. Такой уровень сервиса называется целевым. В описанном сценарии вполне естественно обратиться к помощи автоматических средств, поскольку ручное вмешательство может быть слишком медленным и порождать ошибки.

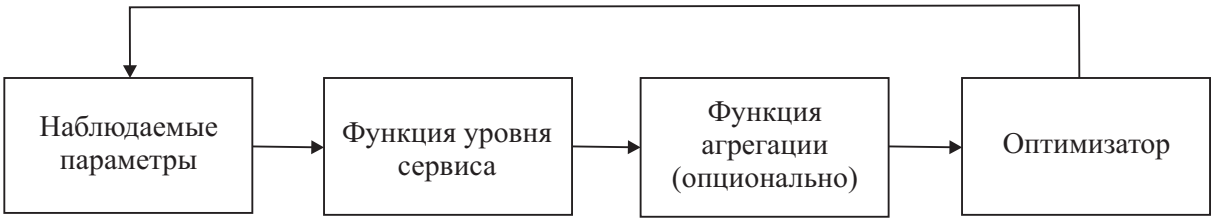

Рис. 5. Схема управления

Одна из возможных схем управления уровнем сервиса приведена на рис. 5. Это классическая схема управления с обратной связью. Для того чтобы повторно использовать алгоритмы, заложенные в *Оптимизаторе*, работающем на системном уровне, необходимо иметь средства для преобразования *наблюдаемых параметров* приложения (таких, как, например, время отклика) в абстрактное значение уровня сервиса [20]. Именно для этой цели вводится *функция уровня сервиса*.

Идея функции уровня сервиса состоит в следующем. Функция принимает наблюдаемые параметры приложения в качестве входного аргумента и возвращает значение уровня сервиса в интервале от 0 до 100 процентов. Затем эти значения передаются Оптимизатору, который, в свою очередь, отыскивает оптимальное распределение ресурсов между одним или несколькими виртуальными инструментами. С этой целью уровни сервисов максимизируются путем вариации объема ресурсов приложения.

Функция уровня сервиса определяется уникальным образом для каждого типа виртуального инструмента, поскольку наблюдаемые параметры и их целевые уровни инструментов различаются. Так, например, для инструмента веб-сайт наблюдаемые параметры должны включать (по меньшей мере) текущее время отклика. В то же время для вычислительного сервиса с предельным сроком, к которому должен быть выполнен расчет (дедлайн), наблюдаемые параметры описывают среднюю скорость счета. В этом случае оптимальной будет такая скорость вычислений, при которой расчет завершится к назначенному сроку и при этом не будут использованы лишние ресурсы.

Поскольку используемые функции уровня сервиса непрерывны, можно применять методы непрерывной оптимизации в Оптимизаторе. В общем случае функция уровня сервиса имеет вид “переключателя” и записывается в форме:

$$s(x) = \frac{wa(b-x)}{\sqrt{1+\big(a(b-x)\big)^2}} + w, \tag{1}$$

где x — наблюдаемые параметры (например, время отклика), s — уровень сервиса, a, b и w — параметры функции, позволяющие адаптировать ее под конкретное приложение.

Предлагается использовать значение 50 % в качестве целевого уровня сервиса для любого инструмента в нашей системе. В этой точке соглашения об уровне сервиса будут строго соблюдаться. Значение 50 выбрано потому, что функция в окрестности этой точки изменяется максимально быстро и, следовательно, здесь оптимизационные алгоритмы будут наиболее эффективны.

Изменяя параметры функции уровня сервиса, можно задавать мягкие и жесткие требования к уровню сервиса. Рассмотрим функцию 1 с параметрами $a = 10$, $b = 1$ и $w = 0.5$. Эти параметры используются для виртуального инструмента веб-сайт. Эта функция равна почти 100 % при x меньше 1 секунды и быстро уменьшается до нуля при времени отклика больше 1.5 секунды (рис. 6).

Параметры кривой должны быть тщательным образом подобраны так, чтобы соответствовать характеристикам производительности приложения и его целевому уровню сервиса. Концепция уровня сервиса

Рис. 6. Кривая уровня сервиса для веб-сайта

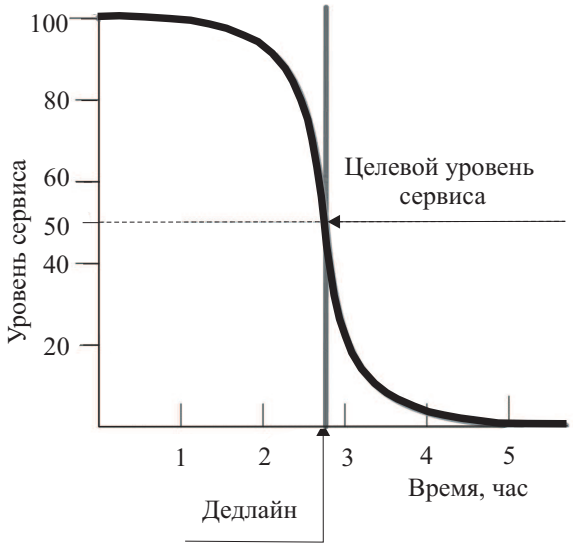

Рис. 7. Кривая уровня сервиса для вычислительного сервиса

может быть естественным образом обобщена для случая многих наблюдаемых параметров, когда кроме времени отклика отслеживается, например, процент сетевых ошибок.

Очевидно, что этот подход применим не только для веб-сайтов и может быть естественным образом расширен на другие типы приложений. В частности, можно использовать его для очередей задач. Как было отмечено выше, Поставщик Сервиса может указать дедлайн, к которому задачи в очереди должны быть рассчитаны (рис. 7). Измерив примерную вычислительную сложность задач, можно посчитать текущую скорость расчета и найти оценочное время завершения. Если системе удается решить задачу в срок, то уровень сервиса этого инструмента равен 50 %. В противном случае он будет снижаться до 0 %, т.е. когда результаты вычислений будут уже не нужны (например, прогноз погоды на завтрашний день, полученный три месяца спустя).

В реальной жизни система должна быть способна работать с несколькими приложениями одновременно. Агрегация n уровней сервиса в один может быть выполнена с помощью взвешенного произведения

$$\sigma(x_1, \ldots, x_n) = \prod_{i=1}^{n} w_i S_i(x_i), \tag{2}$$

где w_i — относительные веса (приоритеты) приложений, S_i — уровни сервисов приложений, а σ — общий уровень сервиса системы. Таким образом, максимизируя σ, система должна форсировать завершение сервиса расчета прогноза погоды завтрашнего дня к полудню сегодняшнего, ценой, возможно, некоторого неудобства утренних посетителей веб-сайта. Ряд методов оптимального управления может быть использован для максимизации σ, например, динамическое программирование.

6. Промежуточная реализация алгоритма оптимизации. На сегодняшний день используется простая одномерная условная оптимизация для поиска оптимального объема ресурсов, необходимого приложению. Рассмотрим пример с вычислительным сервисом. Алгоритм, описанный ниже, пытается найти минимальную долю процессора, при которой будут выполняться соглашения об уровне сервиса, на основе метода двоичного поиска.

На первом этапе находятся границы интервала возможных долей процессора, среди которых будет производиться поиск. Процедура поиска начинается с некоторой грубой оценки объема требуемых приложению ресурсов. Если текущий уровень сервиса приложения ниже целевого (случай I), то значение оценки будет выше текущего объема доступных приложению ресурсов. В противном случае алгоритм будет снижать долю процессора (случай II). Далее этот шаг повторяется с увеличивающимся значением оценки, причем на каждом шаге оценка будет увеличиваться вдвое до тех пор, пока текущий уровень сервиса приложения не достигнет некоторого значения, близкого к целевому уровню сервиса (выше целевого уровня сервиса в случае I и ниже — в случае II).

На втором этапе алгоритм отыскивает в интервале, образованном двумя последними значениями, полученными на предыдущем этапе, величину оптимальной доли процессора для этого приложения. Для

ускорения процесса используется метод половинного деления. На этом шаге доля процессора в распоряжении приложения последовательно увеличивается (или уменьшается) и замеряется новый уровень сервиса приложения. В случае если приложению требуется более одного процессора, автоматически производится запуск дополнительных ВМ с этим приложением.

Этот алгоритм периодически запускается через равные интервалы времени. Таким образом, уровень сервиса приложения поддерживается в заданном интервале автоматически. Существующая реализация, безусловно, является весьма упрощенной, но тем не менее все же применимой к различным инструментам. Так, например, этот алгоритм использовался для динамического распределения ресурсов вычислительного сервиса X-Com в процессе его работы. Инструмент X-Com был запущен с некоторым предустановленным дедлайном, к которому требовалось завершить расчет, и с объемом ресурсов, заведомо недостаточным для выполнения поставленной задачи. Однако, используя описанный выше алгоритм, система увеличила объем ресурсов инструмента так, чтобы все задачи были посчитаны точно к сроку. Эксперименты показали отклонение фактического времени завершения от запланированного лишь на 0.5 %. Например, задача, рассчитываемая в течение около 40 минут, была закончена лишь на 10 – 15 секунд позже заданного срока.

7. Выводы. В ходе нашего исследования была разработана платформа для управления приложениями, работающими в виртуальных машинах Xen. С помощью компонент системы можно запускать и останавливать сервисы, динамически увеличивать или уменьшать объем доступных сервисам ресурсов. Представлена простая модель автоматизации управления уровнем сервиса приложений, использующая методы условной оптимизации. Однако, что более важно, на базе платформы был протестирован ряд приложений с целью составления профилей производительности для последующей разработки более точных и эффективных моделей управления ресурсами виртуальных инструментов. Используемый подход не привязан исключительно к Виртуальным Сервисам и может быть использован в других средах, таких, как Virtual Workspaces или Cluster-on-Demand после некоторых доработок.

Можно возразить, что изменение доли физического процессора, предоставленного ВМ в процессе работы приложения, или полный отказ ВМ может нанести ущерб производительности вычислительных приложений, работающих в таком окружении. Действительно, многие существующие на сегодняшний день MPI-приложения пострадают, поскольку они были спроектированы для работы в однородных вычислительных системах и не смогут корректно работать даже в случае аварийного завершения одного параллельного процесса. Однако более совершенные параллельные приложения, учитывающие неоднородность и нестабильность вычислительной среды, смогут работать в таких условиях. В качестве примера можно упомянуть разработку MapReduce [21] — это универсальная среда общего назначения, которая могла бы справиться с дополнительной сложностью технологий виртуализации. Учитывая те огромные усилия, которые прилагают участники ИТ-сообщества для улучшения высокоуровневых средств параллельного программирования, можно ожидать, что в будущем такого рода приложения получат широкое распространение.

СПИСОК ЛИТЕРАТУРЫ

1. Xen hypervisor (http://www.xen.org/).
2. *Keahey K., Foster I., Freeman F., Zhang X., Galron D.* Virtual workspaces in the grid // Proc. of the 11th Euro-Par Conf. Lisbon, 2005 (http://workspace.globus.org/papers/VW_EuroPar05.pdf).
3. *Yousef L., Wolski R., Gorda B., Krintz C.* Paravirtualization for HPC systems // Proc. Workshop on Xen in HPC Cluster and Grid Computing Environments. Sorrento, 2006, pp. 474–486 (http://dx.doi.org/10.1007/11942634_49).
4. *Novaes R.C., Roisenberg P., Sheer R., Northfleet C., Jornado J.H., Cirne W.* Non-dedicated distributed environment: a solution for safe and continuous exploitation of idle cycles // Proc. Workshop on Adaptive Grid Middleware. New Orleans, 2003.
5. *Абрамов С., Московский А., Первин А., Коряка Ф.* Развертывание испытательного полигона для Grid-приложений в Переславле-Залесском // Распределенные вычисления и грид-технологии в науке и образовании. Дубна, 2006.
6. *Andersen R., Vinter B.* Harvesting idle Windows CPU cycles for grid computing // Int. Conf. on Grid Computing and Application. Las-Vegas, 2006. pp. 121–126.
7. *Moore J., Irwin D., Grit L., Sprenkle S., Chase J.* Managing mixed-use cluster with Cluster-on-Demand. Durham: Duke University Press, 2002.
8. *Sotomayor B.* A resource management model for VM based virtual workspaces. Chicago: University of Chicago, 2007.
9. *Kallahalla M., Uysal M., Swaminathan R., Lowell D.E., Wray M., Christian T., Edwards N., Dalton C.I., Gittler F.* SoftUDC: a software-based data center for utility computing. Los Alamitos: IEEE Computer Society Press, 2004.
10. *Fu Y., Chase J., Chun B., Schwab S., Vahdat A.* SHARP: An architecture for secure resource peering // ACM SIGOPS Operating Systems Review. **37**, N 5. 133-148.

11. *Lai K., Rasmusson L., Adar E., Sorkin S., Zhang L., Huberman B.* Tycoon: an implementation of a distributed market-based resource allocation system. Palo Alto: HP Labs, 2004.
12. *Moroni S., Jofre A., Figueroa N., Sahai A., Chen Y., Iyer S.* A game-theoretic framework for Optimal SLA/Contract creation. Palo Alto: HP Labs, 2007.
13. *Bennani M., Menasce D.* Resource allocation for autonomic data centers using analytic performance models // Proc. of the Second Int. Conf. on Autonomic Computing. Washington: IEEE Computer Society Press, 2005. pp. 229–240.
14. *Menasce D., Bennani M.* Autonomic virtualized environment // Int. Conf. on Autonomic and Autonomous Systems. Washington: IEEE Computer Society Press, 2006.
15. MapServer (http://mapserver.gis.umn.edu/).
16. *Воеводин Вл., Филамофитский М.* Суперкомпьютер на выходные // Открытые системы. 2003. № 5. 43–48.
17. *Thain D., Livny M.* Distributed computing in practice: The Condor Experience. Concurrency and Computation // Practice and Experience. 2004. **17**, N 2–4. 323–356.
18. *Абрамов С., Адамович А., Инюхин А., Московский А., Роганов В., Шевчук Ю., Шевчук Е.* Т-система с открытой архитектурой // Суперкомпьютерные системы и приложения. Минск: ОИПИ НАН Беларуси, 2004. 18–22.
19. Httperf homepage (http://www.hpl.hp.com/research/linux/httperf/).
20. *Chen Y., Iyer S., Liu X., Milojicic D., Sahai A.* SLA decomposition: translating service level objectives to system level threshold. Palo Alto: HP Labs, 2007.
21. *Dean J., Ghemawat S.* MapReduce: simplifed data processing on large clusters // Proc. of the 6th Symposium on Operating System Design and Implementation. San Francisco, 2004 (http://labs.google.com/papers/mapreduce-osdi04.pdf).

Поступила в редакцию
18.03.2008