А. А. Московский, А. Ю. Первин, B. J. Walker

Динамическое управление виртуальными инструментами на вычислительном кластере

Научный руководитель: д.ф.-м.н. С. В. Знаменский

Аннотация. Разработано программное обеспечение для развертывания и управления приложениями, работающих внутри виртуальных машин. Созданы и протестированы следующие приложения: вычислительный сервис и веб-сервис. Составлены профили этих приложений и изучены зависимости между производительностью приложений и ресурсами. Представлены промежуточные результаты активного исследования, направленного на изучение вопросов управления аппаратными ресурсами с использованием математической теории.

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)[1] 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;

Представлено по тематике: *Методы оптимизации и теория управления, Программное обеспечение для суперЭВМ* .

[1]Виртуальная машина — программная среда, эмулирующая работу реального (физического) компьютера

(2) остановить и сохранить ее состояние на диск, предоставляя, таким образом, возможность использовать ресурсы другими, более приоритетным с точки зрения пользователя, виртуальными машинами;
(3) переместить с одного физического компьютера на другой;
(4) запустить с некоторым числом процессоров, а затем добавлять или убирать процессоры во время работы ВМ, с учетом потребностей и приоритетов других ВМ;
(5) запустить с некоторой долей процессора и затем увеличивать или уменьшать ее, исходя из потребностей приложений, работающих внутри ВМ;
(6) запустить с некоторым объемом оперативной памяти и затем, по мере необходимости изменять его;
(7) запустить с ограниченной сетевой пропускной способностью и динамически изменять этот параметр в зависимости от потребностей приложений ВМ.

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, где 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 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, различные 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 обсуждаемое исследование от аналогичных работ в области управления качеством сервиса [12–14]. В этой работе приложения рассматриваются как черные ящики, и среда времени исполнения может использовать мощные методы теории оптимального управления для реализации схем эффективного распределения ресурсов.

Для того, чтобы автоматизировать процесс принятия решения относительно 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 исследования ставится задача реализовать различные типы виртуальных инструментов (virtual appliances[2]) и способы управления ими в условиях меняющейся нагрузки на эти инструменты. 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

[2]Термин virtual appliance пока не имеет устоявшегося перевода на русский язык. В литературе встречается обозначение «шаблон виртуальной машины», однако, на наш взгляд, такой перевод недостаточно точно отражает оригинальный смысл этого термина. Мы используем термин «виртуальный инструмент», 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).

состав аппаратных ресурсов для заданной нагрузки, но также характеризировать 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.

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, либо вычислительные 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 системе, могут быть перенесены на любую другую аналогичную по функциональности платформу. На рис. 1 представлены основные компоненты системы и схема их взаимодействия.

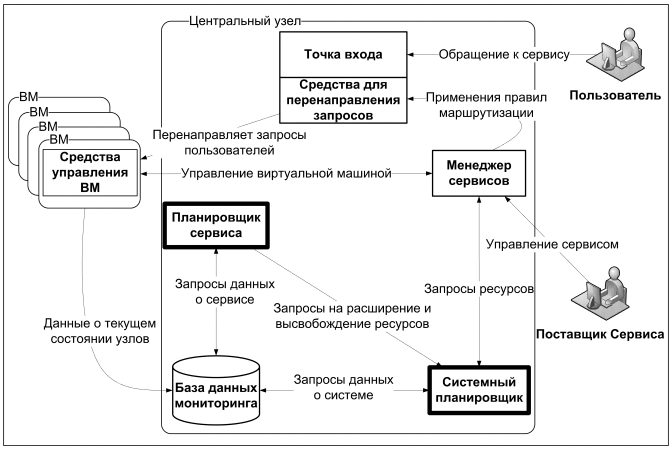

Рис. 1. Компоненты системы Виртуальные Сервисы

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 функции:

- 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 обеспечения доступа пользователей к приложениям в виртуальной среде. Эта функция также используется для балансировки нагрузки.

Гибкость в управлении ресурсами ВМ играет ключевую роль в этом исследовании. В то же время существенное значение имеет возможность использовать разнообразные алгоритмы 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

В настоящий момент система поддерживает три виртуальных инструмента. Для каждого инструмента подготовлен соответствующий образ ВМ и средства поддержки (конфигурационные файлы, планировщики и т.д.)

- **WebMapServer**
  Это приложение [15] позволяет запрашивать различную информацию по географическим картам. В тестах использовались данные по округу Итаска, штата Миннесота, 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`. X-Com [16] — эта система метакомпьютинга, разработанная в МГУ им. М. В. Ломоносова. X-Com чем-то напоминает систему распределенных вычислений `Condor` [17], однако, реализация X-Com значительно более компактна, менее требовательна к ресурсам, 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 число ВМ с поддержкой сетевого подключения между ними. В результате запуска этого сервиса создается набор виртуальных узлов, формирующих *виртуальный кластер*. Сетевая поддержка реализуется с помощью механизма bridging. Он 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, так, 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 (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 обсуждается в следующем разделе.

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 то же время, минимизировать число сетевых ошибок (таких, как, например, закрытие соединение по таймауту). Мы называем такие уровни нагрузки *точками перегиба*. Это такая максимальная пользовательская нагрузка, которую сервис в состоянии обработать корректно.

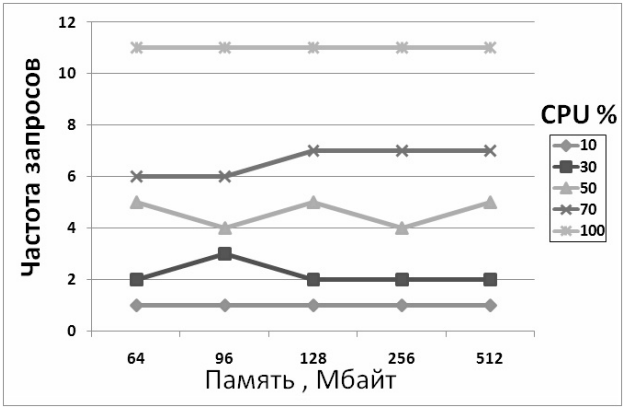

Рис. 2. Нагрузка на сервис WebMapServer, при различных объемах памяти и долях процессора

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 (самая правая кривая), уровень сервиса весьма чувствителен к доле процессорного времени: необходимо, по меньшей мере, 70% процессорного времени для обработки запросов за разумное время (около

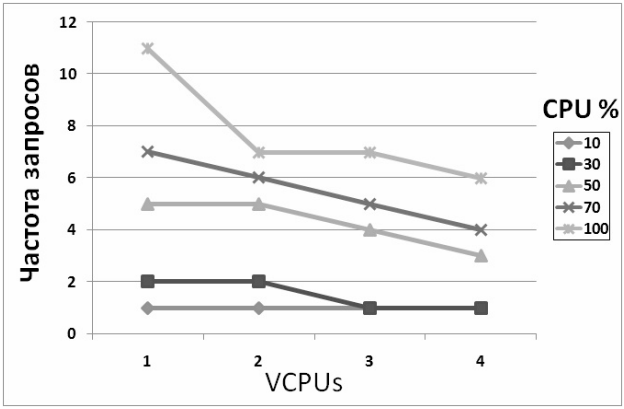

Рис. 3. Нагрузка на сервис WebMapServer, при различном количестве VCPU и долей процессора

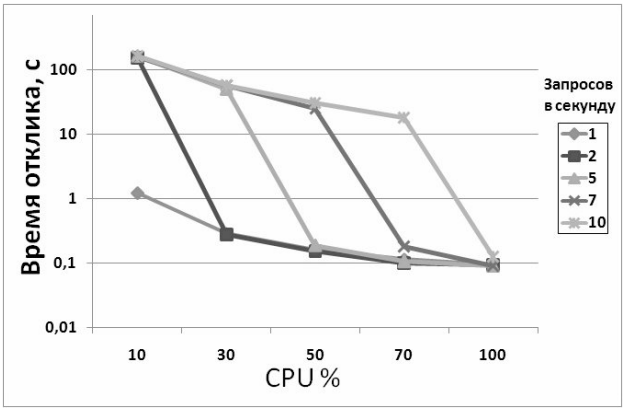

Рис. 4. Время отклика WebMapServer и доли процессора при различных уровнях нагрузки

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 1 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.

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*.

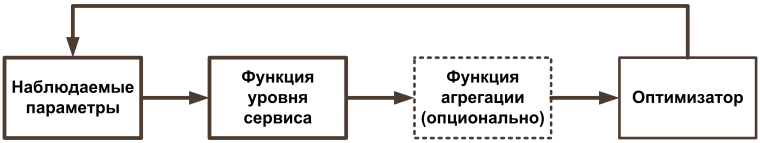

Рис. 5. Схема управления

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 приложений 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. Таким

образом, для инструмента *веб-сайт*, уровень сервиса которого определяется как ограниченное снизу некоторой величиной среднее время отклика, 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:

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

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

Мы предлагаем использовать значение 50% в качестве целевого уровня сервиса для любого инструмента. 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 равным 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 (см. рис. 6). Параметры кривой должны быть тщательным образом подобраны так, чтобы соответствовать характеристикам производительности приложения и его целевому уровню сервиса. Концепция уровня сервиса может быть естественным образом обобщена для случая многих наблюдаемых параметров, когда кроме времени отклика отслеживается, например, процент сетевых ошибок.

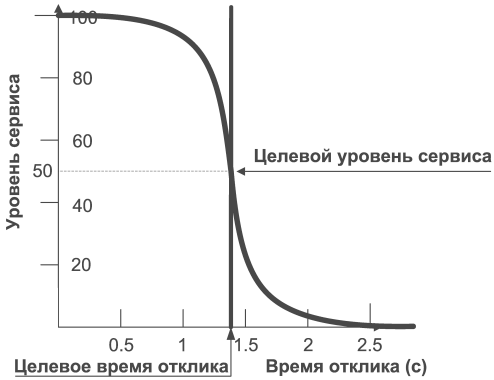

Рис. 6. Кривая уровня сервиса для веб-сайта

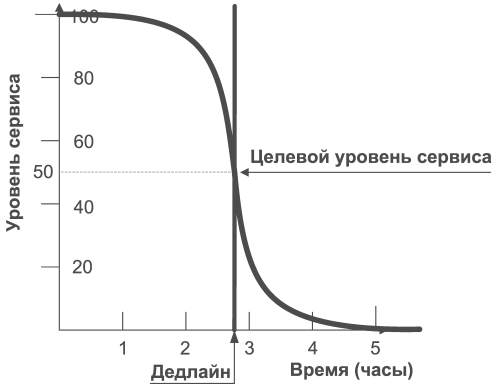

Рис. 7. Кривая уровня сервиса для вычислительного сервиса

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 раньше, Поставщик Сервиса может указать дедлайн, к которому задачи в очереди должны быть рассчитаны (см. рис. 7). Измерив примерную вычислительную сложность задач, можно посчитать текущую скорость расчета и найти оценочное время завершения. Если системе удается решить задачу в срок, тогда уровень сервиса этого инструмента равен 50%. В противном случае он будет

снижаться до 0%, то есть, когда результаты вычислений будут уже не нужны (например, прогноз погоды на завтрашний день, полученный три месяца спустя).

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

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

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, используя метод двоичного поиска.

На первом этапе находятся границы интервала возможных долей процессора, среди которых будет производиться поиск. Поиск начинается с некоторой грубой оценки, относительно объема требуемых приложению ресурсов. Если текущий уровень сервиса приложения ниже целевого (случай I), тогда значение оценки будет выше текущего объема доступных приложению ресурсов. В противном случае, этот параметр будет снижаться (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, тем не менее, все же применима 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 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 40 минут, была закончена лишь на 10–15 секунд позже заданного срока.

7. Выводы

В ходе 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.

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 такие приложения получат широкое распространение.

Благодарности

Данная работа была частично поддержана и выполнялась в рамках следующих проектов:

- Проект 07-07-12038 офи Российского Фонда Фундаментальных Исследований «Метапрограммирование на основе шаблонных классов C++ как средство создания высокопроизводительных распределённых приложений».
- Программа Союзного государства «Разработка и использование программно-аппаратных средств Грид-технологий перспективных высокопроизводительных (суперкомпьютерных) вычислительных систем семейства «СКИФ», шифр «СКИФ-ГРИД».
- Программа фундаментальных научных исследований ОНИТ РАН «Оптимизация вычислительных архитектур под конкретные классы задач, информационная безопасность сетевых технологий», проекты:
  1.3 «Разработка и реализация языков T# и T++ и соответствующих им средств для эффективной поддержки высокопроизводительного параллельного счета»;
  2.3 «Сравнительное исследование параметров и характеристик перспективных аппаратно-программных архитектур кластерных мультипроцессорных систем»;

- Программа фундаментальных исследований Президиума РАН «Разработка фундаментальных основ создания научной распределенной информационно-вычислительной среды на основе технологий GRID».
  Проект 4.2 «Функционально-ориентированные суперструктуры как эффективное средство для построения высокопроизводительных распределенных приложений и сервисов».
  Проект 4.5 «Разработка технологий интеграции разнородных, географически распределенных данных в GRID-системах».
- Программа Президиума РАН «Поддержка инноваций и разработок», проект ИПС РАН «Технология и система параллельного программирования OpenTS»;
- Государственный контракт № 02.514.11.4034 с Федеральным агентством по науке и инновациям по теме: «Эффективные методы создания высокопроизводительных параллельных программ на языке C++ и его диалекте T++ для многоядерных ЭВМ, SMP, кластерных и распределенных систем»;
- Программа Союзного государства (шифр «ТРИАДА») «Развитие и внедрение в государствах-участниках Союзного государства наукоемких компьютерных технологий на базе мультипроцессорных вычислительных систем», проекты:
  «Разработка средств параллельной обработки изображений дистанционного зондирования Земли с динамическим распределением нагрузки. Разработка прототипа распределенного архива изображений с единым каталогом информации»;
  «Создание прототипа системы эффективного сервера приложений для ВМВС»;
  «Разработка принципов эффективного отказоустойчивого счета T++-приложений на априорно неустойчивых (мета-)кластерных конфигурациях и их программная реализация».
- Совместный проект Института программных систем и Hewlett-Packard Laboratories [22] «Виртуальные сервисы».

Список литературы

[1] Xen hypervisor, Эл. ресурс: http://www.xen.org/. ↑1

[2] Keahey K., Foster I., Freeman F., Zhang X., Galron D. Virtual workspaces in the grid. — Lisbon: Proc. of the 11th Euro-Par Conf., 2005, Эл. ресурс: http://workspace.globus.org/papers/VW_EuroPar05.pdf. ↑1

[3] Youseff L., Wolski R., Gorda B., Krintz C. Paravirtualization for HPC Systems. — Sorrento: Workshop on Xen in HPC Cluster and Grid Computing Environments, 2006, 474-486 с. ↑1

[4] Novaes R., Roisenberg P., Scheer R., Northfleet C., Jornada J. H., Cirne W. Non-Dedicated Distributed Environment: A Solution for Safe and Continuous Exploitation of Idle Cycle: Workshop on Adaptive Grid Middleware, 2003. ↑1

[5] Абрамов С., Московский А., Первин А., Коряка Ф. Развертывание испытательного полигона для Grid-приложений в Переславле-Залесском. — Дубна: Распределенные вычисления и грид-технологии в науке и образовании, 2006. ↑

[6] Andersen R., Vinter B. Harvesting idle Windows CPU cycles for grid computing // Int. Conf. on Grid Computing and Application. — Las-Vegas, 2006, 121-126 с. ↑1

[7] Moore J., Irwin D., Grit L., Sprenkle S., Chase J. Managing mixed-use cluster with Cluster-on-Demand. — Durham: Duke University Press, 2002. ↑1

[8] Sotomayor B. A resource management model for VM based virtual workspaces: University of Chicago, 2007. ↑1

[9] Kallahalla M., Uysal M., Swaminathan R., Lowell D. E., Wray M., Christian T., Edwards N., Dalton C., Gittler F. SoftUDC: a software-based data center for utility computing. — Los Alamitos: IEEE Computer Society Press, 2004. ↑1

[10] Fu Y., Chase J., Chun B., Schwab S., Vahdat A. SHARP: an architecture for secure resource peering // ACM SIGOPS Operating Systems Review. — Т. **37**, № 5, 2003, 133-148 с. ↑1

[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. ↑1

[12] Moroni S., Joffre A., Figueroa N., Sahai A., Chen Y., Iyer S. A game-theoretic framework for optimal SLA/contract creation. — Palo Alto: HP Labs, 2007. ↑1

[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, 2006. ↑

[14] Menasce D., Bennani M. Autonomic Virtualized Environment // Int. Conf. on Autonomic and Autonomous Systems. — Washington: IEEE Computer Society Press, 2006. ↑1

[15] MapServer, Эл. ресурс: http://mapserver.gis.umn.edu/. ↑3

[16] Воеводин В., Филамофитский М. Суперкомпьютер на выходные. — Т. **5**: Открытые системы, 2003, 43-48 с. ↑3

[17] Thain D., Livny M. Distributed computing in practice: the Condor experience // Concurrency and Computation: Practice and Experience. — Т. **17**, № 2-4, 2004, 323-356 с. ↑3

[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 с. ↑3

[19] Httperf homepage, Эл. ресурс: http://www.hpl.hp.com/research/linux/httperf/. ↑4

[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. ↑5

[21] Dean J., Ghemawat S. MapReduce: simplified data processing on large clusters // Proc. of the 6th Symposium on Operating System Design and Implementation. — San Francisco, 2004, Эл. ресурс: http://labs.google.com/papers/mapreduce-osdi04.pdf. ↑7

[22] HP Labs, Эл. ресурс:
http://www.hpl.hp.com/. ↑7

Институт Программных Систем РАН

Hewlett-Packard Laboratories

A. A. Moskovsky, A. Y. Pervin, B. J. Walker. *Dynamic Resources Management of Virtual Appliances on a Computational Cluster.* (in Russian.)

Abstract. We have devised an approach for automated, dynamic resource management of applications running on a computational cluster. The following applications have been created and tested: computing oriented and web oriented. Performance-resource dependences of these applications were studied. We present an ongoing research on dynamic resource allocation, involving optimal control methods.

Перевод проверен: к.х.н А. А. Московский