Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования

«Московский физико-технический институт

(государственный университет)»

Факультет управления и прикладной математики

Кафедра теоретической и прикладной информатики

Алгоритм динамического распределения ресурсов в облачной инфраструктуре

Магистерская диссертация

Направление подготовки: 03.04.01 Прикладные математика и физика

Выполнил:
студент 176 группы _____ Редько Игорь Геннадьевич

Научный руководитель:
к.т.н. _____ Мелехова Анна Леонидовна

Москва 2017

Оглавление

Введение 4

Облачные вычисления 4

Преимущества облачных вычислений 6

Многопроцессорные архитектуры 7

Системы с равномерным доступом к памяти 9

Системы с неравномерным доступом к памяти 10

История возникновения NUMA архитектур 12

Особенности виртуализации на NUMA архитектурах 12

Актуальность задачи 15

Проблема размещения виртуальных машин 15

Проблема итеративного перераспределения 17

Существующие алгоритмы балансировки нагрузки 18

Подходы при балансировке 21

Существующие алгоритмы балансировки нагрузки 21

Улучшение размещения виртуальных машин с учётом трафика 21

Взвешенная рандомизированная миграция ВЦПУ 22

Демон оптимизации и уменьшения задержек при работе с памятью 23

Взвешенный циклический алгоритм 25

Ребалансировка нагрузки методом первого подходящего с весами 27

Постановка задачи 27
Методология эксперимента 28
Описание стенда 28
Выбор полезной нагрузки 28
Описание алгоритма 34
Результаты 37
Выводы 42
Список литературы 43

Введение

Облачные вычисления

Облачные вычисления играют важную роль в современной мире информационных технологий и является одним из самых быстрых и интенсивно растущих рынков. Такие крупные компании, например как Oracle и Microsoft, инвестируют огромные суммы в развитие данной технологии, а глава Oracle имеет мнение, что 80% бюджетов IT-компаний уйдут в облачные сервисы и технологии. Термин «облачные вычисления» используют как для приложений, которые предоставляются в виде сервисов по сети, так и для программно-аппаратных комплексов в дата-центрах, которые предоставляют вышеописанные сервисы. Сами приложения, как правило, относят к модели Software as a Service (SaaS). И если в SaaS предоставляются вычислительные ресурсы для приложения (ОЗУ, процессорное время и т.д.), то в более низкоуровневой модели Infrastructure as a Service (IaaS) в качества ресурса выступают виртуальные окружения (виртуальные машины или контейнеры, виртуальное сетевое оборудование).

В рамках данной работы будут использоваться определения, данные Национальным Институтом Стандартов и Технологий [1]:

- Облачные вычисления — информационно-технологическая концепция, подразумевающая обеспечение повсеместного и удобного сетевого доступа по требованию к общему пулу конфигурируемых вычислительных ресурсов (например, сетям

передачи данных, серверам, устройствам хранения данных, приложениям и сервисам — как вместе, так и по отдельности), которые могут быть оперативно предоставлены и освобождены с минимальными эксплуатационными затратами или обращениями к провайдеру.

Стоит отметить несколько принципов, которые отличают облачные вычисления, которые отмечают в своих работах исследователи из Беркли[2]:

- Предоставление «бесконечных» вычислительных ресурсов по требованию, на случай пиковых нагрузок, тем самым устраняя необходимость долгосрочного (для пользователя) планирования резервов.

- Отказ от авансовых обязательств для облачных пользователей. Это позволяет компаниям начать с малого и увеличивать аппаратные ресурсы только тогда, когда есть увеличение их потребностей.

- Возможность платить за использование вычислительных ресурсов на краткосрочной основе по мере необходимости (например, почасовая оплата процессоров или поденная оплата хранилищ) и освобождать их по мере необходимости, тем самым поощряя освобождение машин и хранилищ, когда они больше не нужны.

Преимущества облачных вычислений

Для того чтобы понять, как возникла задача балансировки ресурсов, рассмотрим три сценария, в которых облачный подход к вычислениям (utility computing) более предпочтителен, чем традиционный хостинг[2]. В первом случае востребованность сервиса сильно меняется со временем. Например, при традиционном подходе выделение ресурсов дата-центра, позволяющие справляться с редкими (несколько дней в месяц) пиковыми нагрузками, приводит к неэффективному использованию ресурсов. В качестве альтернативы облачная модель позволяет организации проводить почасовую оплату вычислительных мощностей, что потенциально ведёт к снижению трат, что подтверждено исследователями из Университета Северной Каролины [15]. Во втором рассматриваемом сценарии не известно, какое количество ресурсов будет необходимо. К примеру, Веб-стартап будет нуждаться в технической поддержке ажиотажного спроса при резком росте популярности, но потенциально спрос на сервис может упасть, и тогда часть ресурсов можно будет освободить[14]. Третий сценарий применим к компаниям, занимающимся ресурсоёмкими вычислениями (например, анализом крупных массивов данных), которые могут воспользоваться «ассоциативностью затрат» для ускорения обработки данных. Это возможно за счёт использования 1000 виртуальных машин в течении часа вместо использования одной машины на протяжении 1000 часов.

Таким образом облачные вычисления позволяют заказчикам распределить затраты на вычислительные ресурсы не равномерно, а в соответствии с реальной необходимостью в этих ресурсах (например, использовав 100 машино-часов сегодня и ни одного завтра, платить только

за 100).

Облачные вычисления также позволяют повысить результат консолидации ресурсов[3]. Для традиционных дата-центров оценки средней загруженности колеблются между 5% и 20%. Это связано с тем, что пиковые непродолжительные нагрузки могут быть на порядок выше средней загруженности, и ресурсы, предназначенные для работы при пиковых нагрузках, простаивают при обычных сценариях[8]. Так как ввод в строй новых вычислительных мощностей при традиционных подходах может занимать от нескольких дней до нескольких недель (доставка оборудования, включение его в инфраструктуру, настройка[9]), то использование таких подходов приводит либо к низкой эффективности использования ресурсов, либо к отказам системы в критических сценариях. Благодаря эластичности облачных систем становится возможно динамически распределять вычислительные мощности между сервисами, тем самым потенциально повысить среднюю эффективность серверов.

Для того чтобы предоставлять сервис такого рода, поставщики облаков должны иметь возможность быстро перераспределять нагрузку между физическими серверами. Для этого уже сейчас используют технологии виртуализации и живой миграции виртуальных машин.

Многопроцессорные архитектуры

Под *многопроцессорной системой* будем понимать систему, содержащую несколько процессоров и общую память, видимую для всех процессоров. Системы с распределенной памятью не являются предметом данного исследования. Архитектуры многопроцессорных систем можно классифицировать по нескольким признакам. Приведем наиболее

распространенные классификации.

По наличию нескольких типов процессоров в системе многопроцессорные архитектуры подразделяются на гомогенные и гетерогенные. В гомогенной архитектуре все процессоры одинаковы. В гетерогенной архитектуре присутствуют процессоры разных типов: например, графические процессоры и процессоры общего назначения.

По ролям, которые процессоры играют в многопроцессорной системе, архитектуры делятся на симметричные (Symmetric MultiProcessing или SMP) и ассиметричные (Asymmetric MultiProcessing или AMP). В симметричной архитектуре все процессоры играют одинаковую роль и имеют одинаковый доступ к периферийным устройствам. В асимметричной многопроцессорной архитектуре процессоры играют разные роли и имеют разный доступ к периферийной аппаратуре: например, система может использовать один из процессоров только для выполнения операций ввода-вывода. Соответственно, доступ к устройствам ввода-вывода в такой асимметричной архитектуре будет нужен только одному процессору.

И, наконец, по времени доступа к памяти разными процессорами в системе, многопроцессорные архитектуры подразделяются на системы с однородным доступом к памяти (Uniform Memory Access или UMA) и системы с неоднородным доступом к памяти (Non-Uniform Memory Access или NUMA). В системах с однородным доступом к памяти время запроса к данным из памяти не зависит ни от того, какой именно процессор обращается к памяти, ни от того, где именно в памяти лежат нужные данные. В системах с неоднородным доступом к памяти время доступа к данным из памяти определяется их расположением по отношению к

процессору. Далее рассмотрим UMA и NUMA архитектуры подробнее.

Системы с равномерным доступом к памяти

В UMA системах все процессоры имеют доступ к совместно используемой памяти через общую шину (или другой вид соединения), как показано на рисунке 1. Особенность архитектуры UMA состоит в том, что все процессоры используют единый канал доступа к памяти, а следовательно, время доступа одинаково для всех процессоров. Отметим также, что время доступа не зависит и от расположения информации в памяти.

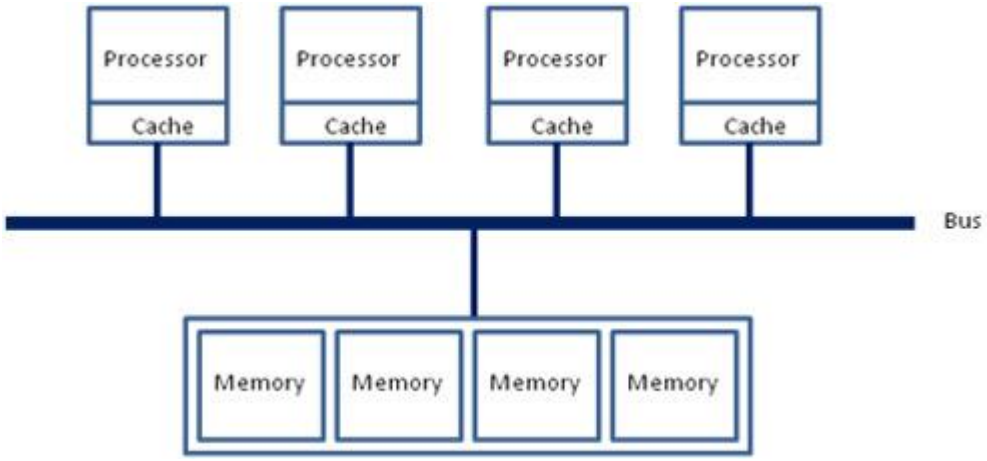

Рис.1. Архитектура UMA систем.

Преимуществом такой архитектуры является то, что балансировку нагрузки на процессоры провести достаточно легко, так как не нужно задумываться о времени доступа к памяти каждого из процессоров — с этой точки зрения все они одинаковы.

Однако системы с UMA архитектурой имеют ряд недостатков, из-за которых не имеют широкого распространения на рынке на сегодняшний день. Во-первых, в каждый момент времени обмен по шине может вести только один из процессоров, то есть процессоры должны соперничать за

доступ к шине. А поскольку фактически процессор обычно намного быстрее памяти, возникает простой процессоров, ожидающих своей очереди для доступа к памяти через шину.

С первого взгляда может показаться, что ситуация, описанная выше, может быть улучшена при наличии у каждого процессора локальной кэш-памяти. Но в этом случае возникает проблема когерентности этих кэшей, которую будет сложно поддерживать, так как при записи в память процессор должен оповестить другие процессоры, содержащие в своих кэшах те же данные, о внесенных изменениях. Это можно сделать или через общую шину, чем увеличить нагрузку на нее еще больше, или через дополнительные каналы связи прямо между процессорами, что потребует изменений в архитектуре системы.

Системы с неравномерным доступом к памяти

NUMA системы пришли как решение проблемы масштабирования систем с однородным доступом к памяти. Принципиальная схема этой многопроцессорной архитектуры представлена на рисунке 2. В архитектуре совместно используемой памяти NUMA для каждого процессора существует отдельный локальный модуль памяти, к которому он может обращаться напрямую, что дает значительное преимущество в работе. В то же время, он может обращаться к модулю памяти любого другого процессора при помощи общей информационной сети.

В данном случае время доступа к памяти варьируется в зависимости от местонахождения данных, к которым осуществляется доступ. Если данные находятся в локальном модуле памяти, доступ осуществляется быстро, в противном случае, для доступа к удаленному модулю требуется большее время.

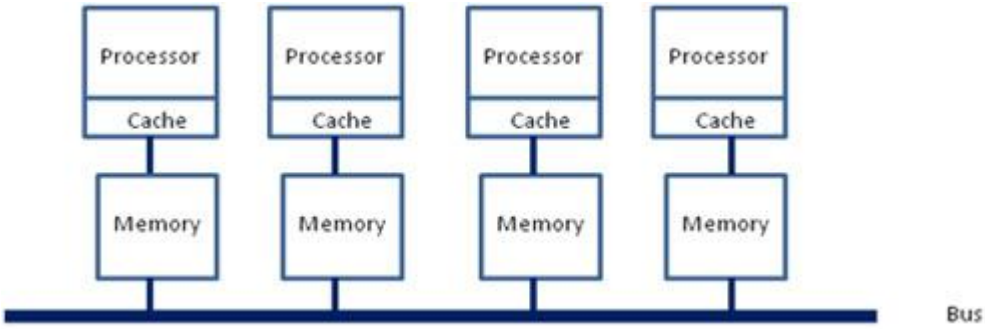

Рис.2. Архитектура NUMA систем.

Преимуществом архитектуры NUMA как иерархической структуры совместно используемой памяти является возможность сокращения среднего времени доступа в результате рационального использования быстрой локальной памяти. NUMA архитектура имеет свои особенности, которые можно и нужно учитывать при оптимизации производительности виртуальных окружений[4].

Также поддерживать когерентность кэшей в NUMA архитектурах проще, чем в UMA архитектурах. Эта простота достигается за счет того, что каждому модулю памяти соответствует один процессор. Если процессор захотел получить доступ к чужой памяти, то он не может сделать это незаметно для процессора, которому эта память принадлежит. Поэтому при изменениях в запрашиваемом куске памяти процессор, которому эта память принадлежит, будет знать, кого оповестить об этих изменениях. Так NUMA архитектура, используемая в системах Intel, поддерживает когерентность кэшей, поэтому ее иногда называют ccNUMA — cache coherent NUMA. Это означает наличие специального аппаратного решения для согласования содержимого кэшей. Конечно, такое общение кешей ухудшает общую производительность системы, но без него программировать систему с непредсказуемым текущим состоянием данных было бы крайне затруднительно.

Недостатком NUMA систем является большое время доступа к чужой памяти, из-за чего невнимание к особенностям архитектуры может привести к ухудшению работы памяти, возможные эффекты были описаны в практической литературе[5], а также подтверждены на практике[6].

NUMA на сегодняшний день более распространена на рынке. UMA архитектура воспринимается скорее как предок NUMA, чем как реальная альтернатива[7]. Это связано не только с ее преимуществами по сравнению с UMA архитектурами, а еще и с тем, что NUMA проще в реализации.

История возникновения NUMA архитектур

NUMA архитектура логически следует из симметричных мультипроцессорных архитектур, которые активно разрабатывались в течение 1990х годов множеством крупных компаний. Впервые идею гибридной архитектуры предложил Стив Воллох, он воплотил ее в системах серии Exemplar. Вариант Воллоха – система, состоящая из восьми SMP-узлов. Фирма Hewlett-Packard купила идею и реализовала на суперкомпьютерах серии SPP. Идею подхватил Сеймур Крей (Seymour R.Cray), которого ещё называют “отцом суперкомпьютеров”, и добавил новый элемент – когерентный кэш, создав так называемую архитектуру cc-NUMA (Cache Coherent Non-Uniform Memory Access), которая расшифровывается как " неоднородный доступ к памяти с обеспечением когерентности кэшей". Он ее реализовал на системах типа Origin.

Особенности виртуализации на NUMA архитектурах

Как уже было объяснено ранее, главная особенность заключается в

разной скорости доступа к своим и чужим относительно процессора модулям памяти. В качестве наглядного доказательства прямого влияния этого фактора приведём исследования Yuxia Cheng в Performance-Monitoring-Based Traffic-Aware Virtual Machine Deployment on NUMA Systems. В этой статье автор измерил производительность NUMA системы для нескольких типов виртуализационной нагрузки на систему.

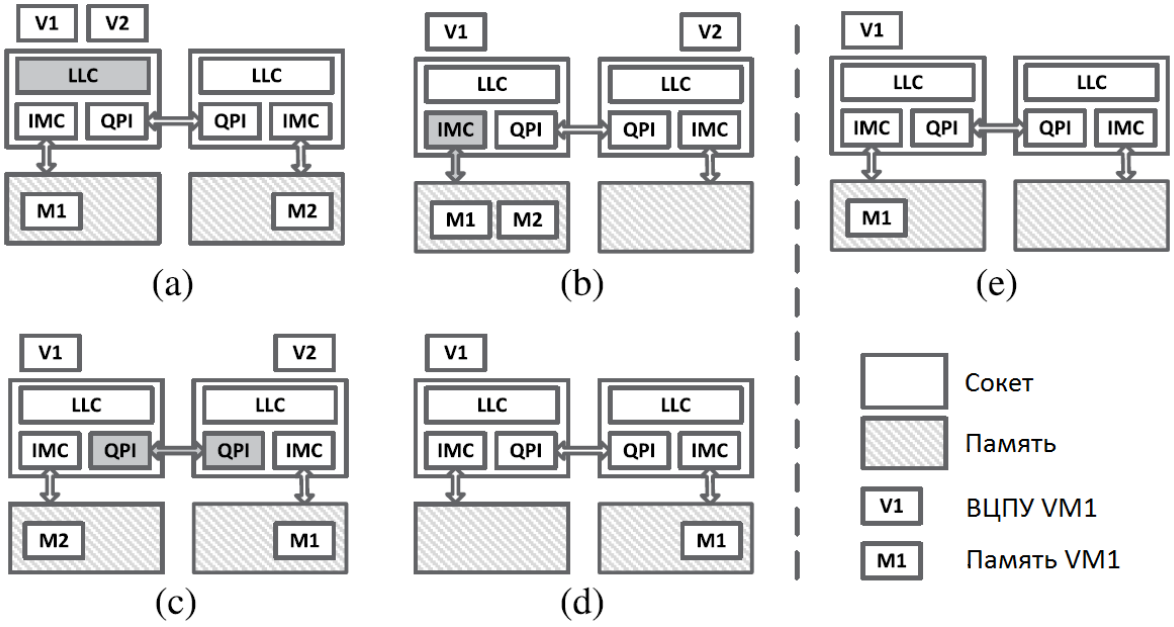

Рис.3. Причины падения производительности ВМ при работе на NUMA системе.

Описание пяти сценариев нагрузки с иллюстрацией представлено на рисунке 3, в скобках указано узкое место:

a) Две ВМ на одном камне, память на разных узлах. (LLC кэш)

b) Одна ВМ и память на чужом узле. (удалённая память)

c) Две ВМ на разных камнях, память “накрест”. (QPI).

d) Две ВМ на разных камнях, память на одном узле. (IMC камня которому принадлежит память)

е) Одна ВМ и память на своем узле. (база)

Актуальность задачи

Проблема размещения виртуальных машин

Работа виртуальных окружений в многоядерных NUMA системах с большим количеством разделяемых ресурсов может приводить к потерям производительности на разных уровнях. В работах других исследователей было показано, что задача оптимального распределения по ядрам принадлежит NPC классу [11], если количество ядер больше или равно трем, так как данная задача сводится к многомерной задаче о назначениях.

Многомерную задачу о назначениях можно математически сформулировать следующим образом[24]:

Даны K множеств $Q_1, Q_2 \ldots Q_K$*, каждое из которых содержит M элементов, и функция стоимости* $C: Q_1 \times Q_2 \times \ldots \times Q_K \rightarrow R$*. Нужно найти такое назначение A, состоящее из* M *подмножеств, каждое из которых содержит один единственный элемент из каждого множества* Q_K*,* $1 < k < K$*. Каждый элемент назначения A* $a_m = \{a_{m1}, a_{m2} \ldots a_{mk}\}$ *имеет стоимость* $c_m = C(a_m)$*, где* $1 \leq m \leq M$ *и* a_{mk} *– элемент, выбранный из множества* Q_k*. При этом каждый элемент* Q_k*,* $1 < k < K$*, должен принадлежать одному единственному подмножеству назначения A, общая стоимость* $\sum_{m=1}^{M} c_m$ *минимальна.*

Для того, чтобы показать, как задача размещения виртуальных машин сводится к многомерной задаче о назначениях, сформулируем

задачу размещения виртуальных машин в математических терминах:

Дано множество S*, которое содержит* N *элементов. Обозначим через* S_K *множество всех* K*-размерных подмножеств* S*. Каждое из этих* K*-размерных подмножеств обозначим через* G_i*, где* $i = 1,2 \dots (\frac{N}{K})$*. Каждое подмножество* G_i *имеет вес* w_i*. Цель состоит в нахождении* $\frac{N}{K}$ *таких подмножеств* $G_{p_1}, G_{p_2} \dots G_{p_{\frac{N}{K}}}$*, что выполнены следующие условия:*

1. $\cup_{i=1}^{\frac{N}{K}} G_{p_i} = S.$
2. $\sum_{i=1}^{\frac{N}{K}} w_{p_i}$ *минимальна.*

В такой формулировке каждый элемент S соответствует виртуальной машине, каждое подмножество G_i соответствует группе виртуальных машин, назначенных на исполнение на одно ядро, а их веса w_i соответствуют суммарной деградации всех виртуальных машин в каждом подмножестве G_i. Первое условие означает, что каждая виртуальная машина распределена на какое-либо ядро. Второе условие, соответственно, отвечает за минимизацию общей деградации всех виртуальных машин.

Теперь покажем, что задача размещения виртуальных машин сводится к многомерной задаче о назначениях. Пусть $S = \cup_{k=1}^{K} Q_k$ и $N = M * K$. Построим подмножества G_i, $1 \leq i \leq (\frac{N}{K})$. Если подмножество G_i содержит в точности один элемент из $Q_k, 1 < k < K$,

то его вес будет равен $w_i = C(a_1, a_2 \dots a_K)$, где C – это функция стоимости в контексте многомерной задачи о назначениях, а Q_k – элемент, выбранный из Q_k. Иначе положим $w_i = +\infty$.

Минимизация суммарного веса в задаче размещения виртуальных машин ведет к минимизации суммарной стоимости в многомерной задаче о назначениях и наоборот. Из этого следует, что задача размещения виртуальных машин является NP-полной при количестве ядер для размещения виртуальных машин $K \geq 3$.

Проблема итеративного перераспределения

Как уже было отмечено, задача распределения виртуальных машин принадлежит классу NPC при количестве ядер для размещения большим трех, но это не самая большая проблема, с которой придется столкнуться.

Отметим еще несколько проблем перераспределения виртуальных машин, которые усложняют задачу балансировки:

- Виртуальные машины могут требовать ресурсы не кратно узлам. Например, 3 виртуальные машины по 4 ГБ нужно разместить на двух узлах по 6 ГБ.

- Нагрузки могут иметь разную интенсивность. Например, виртуальные машины заявлены с одинаковыми размерами памяти, но некоторые из них активно работают с памятью, а некоторые нет. Эта проблема может быть решена с помощью чтения счётчиков производительности.

- Миграция памяти – это дорого во всех смыслах, так как создает дополнительную нагрузку на CPU, LLC, QPI, IMC. Поэтому

перед тем, как динамически перераспределять виртуальные машины по узлам, нужно оценить, перекроет ли выигрыш от перераспределения затраты на миграцию.

- Существуют проблемные ситуации, которые нельзя решить одной миграцией, например, сильно фрагментированные по узлам виртуальные машины.

- Размер и интенсивность нагрузки меняется динамически. Это создает дополнительные трудности в оценке целесообразности перераспределения виртуальных машин.

Существующие алгоритмы балансировки нагрузки

Чтобы улучшить возможности виртуальных машин для предоставления надежных услуг на облачной платформе, предлагается облачная вычислительная система с архитектурой, ведущей и подчиненной систем. Система состоит из облачного контроллера. Облачный контроллер является точкой входа в систему, а также отвечает за управление лежащими в основе виртуализованными ресурсами, такими как серверы, хранилища и сети. Контроллер облака представляет собой набор сервисов, которые сгруппированы по трем категориям: ресурсные службы, службы данных и интерфейсные службы. Он также может обрабатывать перевод протоколов и обеспечивать управление общими системами. Политики брокера. Этот компонент моделирует сервис-брокеров, которые обрабатывают маршрутизацию трафика между пользовательскими базами и центрами обработки данных. Политика маршрутизации по умолчанию направляет трафик в ближайший центр обработки данных с точки зрения задержки сети от исходной базы пользователей. Кроме того, экспериментальная брокерская политика для

максимальной нагрузки для совместного использования нагрузки центра обработки данных с другими центрами обработки данных, когда производительность исходного центра обработки данных ухудшается выше заранее определенного порога.

Решение задачи итеративного перераспределения в общем случае имеет логическое строение, представленное на рисунке 5.

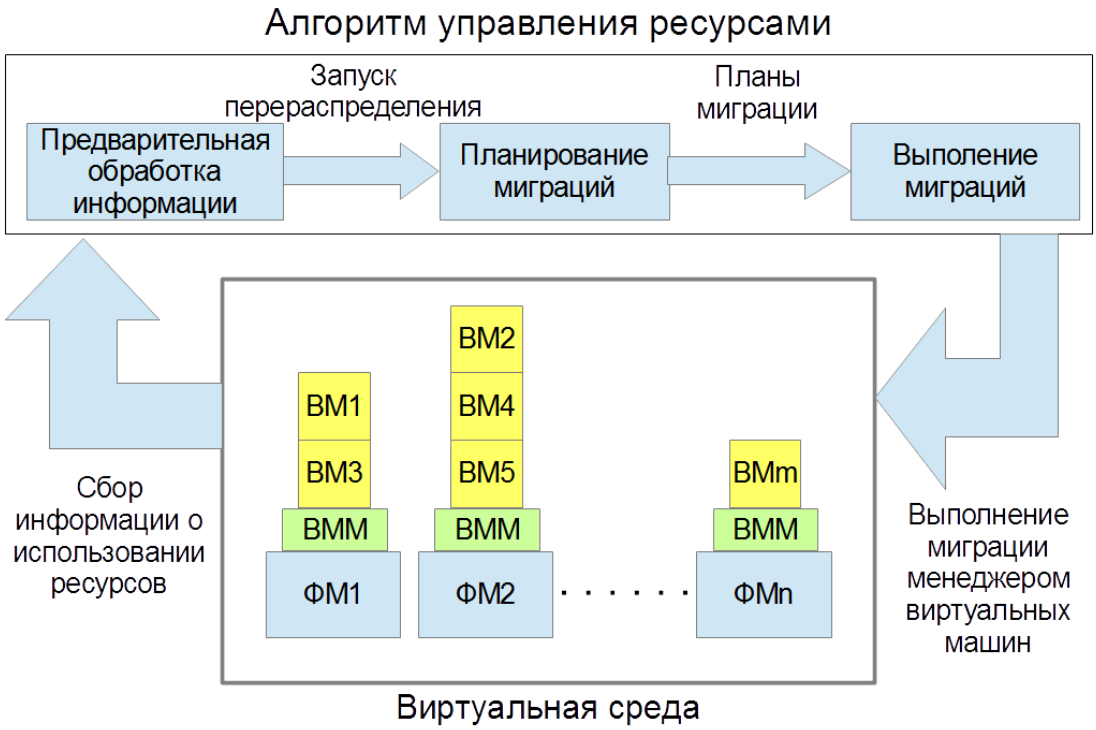

Рис.5. Обобщённая архитектура менеджера ресурсов.

Опишем фазы управления ресурсами в общем случае:

1. Подготовительная фаза. Менеджер на этом этапе собирает на информацию об используемых ресурсах. Способ получения, равно как и объём собираемых данных, зависит от конкретной

реализации. На основе предварительной обработки принимается решение о запуске перераспределения ресурсов в случае нарушения предварительно заданных условий.

2. Фаза планирования миграций. Во время этого этапа менеджер непосредственно обрабатывает полученную информацию и на основании конкретного алгоритма формирует план миграции, который должен устранить или минимизировать возникшие нарушения условий. Обычно такой план состоит из троек «NUMA узел-отправитель, виртуальная машина, NUMA узел-получатель».

3. Фаза выполнения миграции. На этом этапе происходит выполнение плана ядром операционной системы.

Подходы при балансировке

Существует два основных подхода в балансировке нагрузки, которые можно обозначить как выталкивающие миграции и притягивающие миграции. Выталкивающие миграции – перенос полезной нагрузки на наименее нагруженный узел с более нагруженного. При использовании подхода с притягивающими миграциями нагрузка переносится на узел, когда этот узел начинает простаивать.

Существующие алгоритмы балансировки нагрузки

Улучшение размещения виртуальных машин с учётом трафика

В своей работе Yuxia Cheng с коллегами предлагают[10] рассматривать задачу балансировки как частный случай задачи о многомерном рюкзаке. В качестве измерений рюкзака выбраны оперативная память, процессорное время и ресурс межпроцессорной шины, используемые виртуальной машиной.

Предложенный подход использует алгоритм “первый подходящий” для первичного размещения виртуальных машин на физической машине, а дальше на каждом шаге проверяет сбалансированность потребления процессорного времени. В случае если нагрузка не сбалансирована, балансировщиком выполняются следующие действия:

1. Поиск NUMA узла с наибольшей очередью на выполнение виртуальных центральных процессорных устройств(ВЦПУ) и NUMA узла с наибольшим количеством неактивных ЦПУ. Первый является донором нагрузки, второй реципиентом. В случае если

существует несколько реципиентов, то выбирается тот узел, который использует меньше ресурса межпроцессорной шины.

2. Определение количества ВЦПУ для переноса. Это значение является минимумом из неиспользуемых физических ЦПУ на узле-реципиенте и ожидающих ВЦПУ на узле-доноре
3. Поиск ВЦПУ, которые будут перенесены с узла-донора.
4. Перенос нагрузок. Тут авторы отмечают, что происходит только перенос процессорной нагрузки без миграции памяти, что приводит к увеличению нагрузки на межпроцессорную шину.

Миграция памяти происходит при длительной разбалансировке использования памяти на NUMA узлах. В качестве механизма определения необходимости миграции предлагается вести учёт числа переносов процессорной нагрузки с “домашнего” узла (того, где размещена оперативная память виртуальной машины), и в случае если за последние 10 перебалансировок машину переносили 8 раз на другой узел, то происходит вызов алгоритма первичного размещения.

К достоинствам этого подхода следует отнести то, что он учитывает дороговизну переноса памяти с одного узла и описывает один из способов избежать лишних миграций памяти. В то же время остался нераскрытым способ измерения и учёта использования пропускной способности межпроцессорной шины каждого из NUMA узлов, что является одним из ключевых ресурсов в описанной модели.

Взвешенная рандомизированная миграция ВЦПУ

Подход, описанный Jia Rao в статье [6], предлагает для каждой виртуальной машины находить наиболее подходящий NUMA узел,

основываясь на штрафе за использование межпроцессорной шины. Исследования проводились для виртуализационной платформы Xen. Штраф, описанный в статье, вычисляется по счётчикам производительности процессора. К сожалению, точная формула не приводится, но качественно это среднее время жизни запроса к памяти из другого NUMA узла в очереди к числу таких запросов.

Отличительной особенностью предложенного алгоритма является то, что миграция нагрузки на узел с наименьшим штрафом происходит случайно, а вероятность миграции тем выше, чем больше времени нагрузка находится на неоптимальном узле. С помощью этого подхода авторы борются с пиками в нагрузке и накладными расходами, присущими изменению привязки ВЦПУ к ЦПУ на платформе Xen.

Один из недостатков метода заключается в отсутствии механизма восстановления локальности размещения оперативной памяти виртуальной машины. Также не был предложен алгоритм начального размещения виртуальных машин. Возможно, перечисленные недостатки связаны с ограничением выбранной в исследовании платформы – гипервизор Xen не позволяет напрямую влиять на то, в каком NUMA узле будет выделена память.

Демон оптимизации и уменьшения задержек при работе с памятью

NUMAd – это сервис балансировки нагрузки с открытым исходным кодом. Данное решение является частью программного комплекса Red Hat Enterprise Virtualization (RHEV)[12], и создан для работы в связке с гипервизором QEMU-KVM под управлением менеджера виртуальных машин libvirt [13].

Для решения задачи первичного размещения numad использует

алгоритм поиска первого подходящего. В качестве размеров рюкзака для каждого NUMA узла рассматривается объём оперативной памяти и количество физических ядер с учётом гипер-тредингa. Динамическая часть алгоритма выполняется следующие шаги:

1. Сбор информации о потреблении процессорного времени и оперативной памяти виртуальными машинами отдельно в каждом NUMA узле.
2. Упорядочивание ВМ в порядке убывания “значительности”. Значительность – метрика, введённая для оценки степени влияния работы отдельной ВМ на производительности системы в целом[12], и равная произведению процессорного времени на оперативную память, потреблённую на всех узлах.
3. Вычисление для всех ВМ, начиная с самых значимых, узла, на котором уже размещена большая часть занятой оперативной памяти, и перенос вычислений и данных на найденный узел.

К безусловным преимуществам решения можно отнести глубокую интеграцию с существующими системами оптимизации использования вычислительных ресурсов, таких как сервисы прозрачного для процессов предоставления больших страниц памяти и объединения одинаковых страниц памяти на уровне ядра.

К недостаткам алгоритма можно отнести отсутствие механизма балансировки потребления вычислительного времени в случае отсутствия фрагментации памяти. В рассмотренном решении общее состояние системы рассматривается только с позиции общего доступного для использования количества ресурсов, поэтому, как показал эксперимент в

ситуации с неустранимой фрагментацией (например 3 ВМ требующие по 4 гигабайта ОЗУ, на сервере с двумя NUMA узлами по 6 гигабайт ОЗУ), алгоритм зацикливается на локализации фрагментированной ВМ, которая приводит к фрагментации другой ВМ, и, как результат, к просадке производительности.

Взвешенный циклический алгоритм

Предлагаемый исследователями алгоритм представляет собой взвешенный аналог существующего циклического алгоритма (round robin) с некоторыми изменениями в нем. В этом алгоритм взвешенного циклического алгоритма, который распределяет все входящие запросы доступным виртуальным машинам круговым способом на основе веса без учета текущей нагрузки на каждую виртуальную машину. Шаги при планирования следующие:

1. Планировщик получает информацию о виртуальной машине от подчиненного устройства. Если главный узел не получит данные от подчиненного устройства, то это будет означать, что виртуальная машина выключена. Эта ситуация определяется параметром W:
    - Если W = 0, то мастер-система определит, что виртуальная машина работает.
    - Если W = 1, то узел выключен.
2. Если планировщик получает данные от ведомого устройства, он получает информацию об используемых ресурсах (используемая память, время процессора и т. д.).
3. Затем главный узел строит взвешенную таблицу с помощью данных, которые были собраны на шаге 2.

4. Затем главный узел сортирует все виртуальную машину в соответствии с их производительностью.
5. Планировщик распределяет виртуальные машины по узлам в соответствии с взвешенным значением.

Ребалансировка нагрузки методом первого подходящего с весами

Постановка задачи

Существующие решения задачи распределения ресурсов в системе с неравномерным доступом к памяти либо не нашли промышленного применения в силу недостатков, описанных ранее, либо всё ещё проигрывают ручному распределению на реальных нагрузках. Таким образом, проблема распределения ресурсов с учётом NUMA архитектуры актуальна.

В результате анализа имеющихся решений и предметной области были выделены несколько наиболее эффективных принципов, и данная работа призвана объединить их в одну сбалансированную политику. В частности, алгоритм и его реализация должны:

1. Размещать все потоки процесса виртуальной машины в пределах одного NUMA узла.
2. Определять степень использования вычислительных ресурсов NUMA узла по счетчикам производительности процессора (perf counters).
3. Восстанавливать локальность размещения ВЦПУ и данных виртуальной машины.
4. Оценивать улучшение производительности системы в целом в результате миграции памяти отдельных ВМ для уменьшения

накладных расходов.

Методология эксперимента

В качестве объектов исследования была выбрана платформа: OpenVZ 7. Целью эксперимента было сравнение разработанного алгоритма балансировки виртуальных окружений с алгоритмом балансировки, реализованным в ядре Linux, и ручным размещением ВМ.

Описание стенда

В качестве стенда для проведения экспериментов использовался сервер IBM System x3650 M3 7945L0F с двумя одинаковыми NUMA узлами. Каждый узел включал в себя процессор Intel® Xeon® CPU E5645 2.39 GHz и 64 гигабайта оперативной памяти DDR3. В качестве операционной системы и виртуализационной платформы использовались тестовые сборки продукта Virtuozzo седьмой версии. Для того чтобы исключить влияние тестовой нагрузки на обработку результатов, управление тестом и наоборот, был использован дополнительный сервер.

Выбор полезной нагрузки

В рамках данной работы было решено отойти от традиционного подхода оценки серверных платформ на основе отдельных параллельных приложений или тестов (TPC-C, TPC-E, TPC-W, SPECjbb, SPECjappserver и NAS [16]) в качестве целевых нагрузок. Анализ результатов тестов производительности сервера сам по себе представляет собой серьезную задачу, поскольку эталонные тесты имеют сложное поведение, требуют множественных клиент-серверных систем и трудны для запуска на

полнофункциональных симуляторах. Тем не менее, отраслевые и академические исследователи справились с этой проблемой, разработав симуляции. Поскольку центры обработки данных начали использовать виртуализацию как средство консолидации нескольких приложений на сервере, становится критически важным, корректно оценить производительность как виртуализационных технологий, так и средств управления ними.

Отраслевыми эталонами для решения этой проблемы являются vConsolidate [17] и SPECvirt [21]. Одно из них предоставил комитет SPEC занимающийся стандартизацией бенчмарков.

Основные соображения при моделировании производительности виртуальных машин (ВМ) можно резюмировать следующим образом:

1. Производительность виртуальной машины зависит не только от ее собственных характеристик, но также зависит от помех, вызываемых другими виртуальными машинами, работающими на одной платформе с ней.

2. Вышеуказанные помехи могут быть вызваны:

    a. использованием общих ресурсов (например, машинного времени, оперативной памяти), которое можно отследить через счетчики производительности

    b. использованием общих ресурсов (кэш-памяти, пропускной способности межпроцессорной шины, шины памяти и т.д.), которые невидимы для операционной системы, поскольку они являются прозрачными ресурсами, управляемыми аппаратным обеспечением.

3. Особенности технологии виртуализации и дисциплины планирования, принятые монитором виртуальной машины, могут быть совершенно разными на любой данной платформе. Подход моделирования должен учитывать технологию виртуализации, а также требуемую эвристику планирования.

Таким образом, выбранный подход к моделированию должен принимать во внимание как видимые, так и невидимые факторы влияния ресурсов на производительность.

В соответствии со всем вышесказанным в качестве полезной нагрузки использовались vConsolidate и SPECvirt.

Тест vConsolidate состоит из вычислительной интенсивной рабочей нагрузки, рабочей нагрузки базы данных и рабочей нагрузки веб-сервера. Каждая из этих рабочих нагрузок выполняется в собственной виртуальной машине. Для эмуляции среды реального мира в микс добавляется простаивающая виртуальная машина, поскольку центры обработки данных не используются полностью все время.

Симуляция каждого типа нагрузки выполняется на отдельной виртуальной машине:

- интенсивные вычисления эмулируются на VM с набором тестов SPECjbb;
- нагрузку на память и диск создаёт VM с тестом Sysbench;
- за нагрузку на сеть отвечает VM с тестом Webbench;
- отдельная виртуальная машина находится в состоянии “простоя”, когда исполняется только рутинные задачи гостевой операционной системы.

Обычно, SPECjbb – это интенсивная загрузка процессора, которая потребляет столько CPU, сколько возможно. Однако в этой среде, а именно vConlidate, SPECjbb был изменен, чтобы потреблять примерно 75% процессора, вставляя в случайное время простоя каждые несколько миллисекунд. Тест Webbench, использует Apache Webserver. Конфигурация, как описано выше, с 4 ВМ, работающими с разными рабочими нагрузками, включает консолидированный стековый блок (consolidated stack unit) или CSU. Отдельно стоит отметить, что в нашем тестовом стенде клиент и канал для запросов на веб-сервер были выделены отдельные для снижения влияния состояния внутренней сети на измерения.

SPECvirts – первая сравнительная оценка производительности SPEC для серверов, предназначенных для работы в центрах обработки данных с использованием виртуализированных сред. В тесте используется несколько существовавших ранее нагрузок, разработанных SPEC, представляющих приложения, которые являются общими задачами, решаемыми серверами в центрах обработки данных. Эти рабочие нагрузки являются модифицированными версиями SPECweb2005, SPECjAppServer2004 и SPECmail2008. Консолидированный стековый блок (consolidated stack unit) или CSU в SPECvirt состоит из шести виртуальных машин, каждая из которых выполняет определенную рабочую нагрузку.

Существуют тесты производительности семейства VMmark [22], разработанные компанией VMware. Тесты версии 1.x [19] запускают на физической машине наборы виртуальных машин с запущенными внутри тестами. Оценка теста производится также, как и в тесте VConsolidate: оценка достигается путем измерения совокупной пропускной способности, достигаемой несколькими рабочими нагрузками,

выполняемыми одновременно на платформе виртуализации; далее вычисляется среднее геометрическое для нормированных индивидуальных оценок по блокам виртуальных машин, а показания по этим блокам затем суммируются.

Набор из шести конкретных рабочих нагрузок, каждый на своей собственной виртуальной машине, запускается в течение определенного периода времени. Эти шесть виртуальных машин рабочей нагрузки совместно определяются как плита VMmark (от англ. tile). В дополнение к этому счету, каждый результат VMmark также включает в себя количество плиток VMmark, используемых в эталонном прогоне. Отчет VMmark также включает исходные и нормализованные результаты для каждой основной рабочей нагрузки, а также полную информацию о конфигурации платформы виртуализации.

Плитка представляет собой набор из шести разнообразных рабочих нагрузок, одновременно выполняющих определенное программное обеспечение. Работая на одной из двух отдельных операционных систем, каждая рабочая нагрузка запускается на собственной виртуальной машине и выполняет приложения, найденные во всех мировых центрах обработки данных. В одну плиту входят:

- веб-сервер
- файловый сервер
- почтовый сервер
- база данных
- java-сервер

● незанятая машина

Каждая виртуальная машина в плитке настроена на использование только доли общих ресурсов системы.

Плитка VMmark 2.x [20] представляет собой группу из восьми виртуальных машин, одновременно выполняющих коллекцию разнообразных рабочих нагрузок. Каждая из этих рабочих нагрузок представляет собой общую рабочую нагрузку приложения, найденную в современных центрах обработки данных. В каждую плитку входят:

● почтовый сервер

● база данных Web 2.0

● веб-система, связанная с этой базой данных (3 реплики)

● база данных электронной коммерции

● интерфейсный веб-уровень, связанный с этой базой данных

● незанятая машина.

Таким образом, одна плитка состоит из двух рабочих нагрузок одноуровневого приложения, двух рабочих нагрузок многоуровневых приложений и четырех рабочих нагрузок на уровне инфраструктуры.

VMmark 2.x был спроектирован как тест, запускаемый на сервере. Он отражает типичное современное использование виртуализованной инфраструктуры.

Более высокая оценка VMmark 2.x означает, что платформа виртуализации способна поддерживать большую пропускную способность в смешанной среде консолидации рабочей нагрузки, одновременно

работая в центрах обработки данных в фоновом режиме. Большее количество плиток VMmark 2.x, используемых для создания эталона, означает, что платформа поддерживала большее количество виртуальных машин на несколькиххостах во время тестового теста.

В дополнение к измерению производительности в предыдущих версиях VMmark, VMmark 2.5 добавляет возможность измерения как абсолютного потребления энергии, так и энергоэффективности серверов и хранилищ, используемых в тестовых тестах. Это позволяет принимать решения о покупке не только абсолютной производительности, но и производительности на киловатт, что становится все более значительным фактором общей стоимости владения вычислительными ресурсами.

Хотя данное семейство тестов покрывает больший функционал, они заточены под использование с виртуализационными средствами компании VMware. Целью данных тестов является оценка производительности серверов, а не виртуализационных уровней.

Данное ограничение не позволяет использовать данное семейство тестов для исследования, проводимого в работе.

Описание алгоритма

Для выбора NUMA узла при первичном размещении виртуальных машин используется приближенное решение задачи о многомерном рюкзаке «первый подходящий». В качестве размеров рюкзака используется объём оперативной памяти и доступные для использования физические процессоры. Это необходимо для соблюдения локальности данных и вычислителя, что повышает производительность на системах с NUMA архитектурой. Однако это не всегда возможно, например, как в

случае, приведённом в обзоре существующих алгоритмов.

Для того чтобы восстанавливать локальность данных, не допускать простоя вычислительных мощностей и исключить накладные расходы, которые не приводят к улучшению производительности системы в целом, введём понятие потребления ресурса. Потреблением ресурса будем называть отношение использованного ресурса (машинного времени, оперативной памяти, канала межпроцессорной шины) к общему количеству доступного ресурса для использования в рамках указанного NUMA узла.

При каждом изменении состояния виртуальной машины (запуск, остановка, пауза и т.д.) или раз в несколько минут выполняется балансировка потребления ресурсов виртуальными машинами между NUMA узлами:

1. Вычисление потребления ресурсов каждой ВМ в рамках каждого NUMA узла.
2. Формирование списка ВМ, для которых не выполняется условие локальности (данные размещены в рамках двух или более NUMA узлов) и межпроцессорная шина использовалась слишком активно.
3. Далее из списка узлов выбирается наиболее загруженный и наименее загруженный узлы (те у которых выше или ниже чем у остальных сумма потребленной памяти соответственно).
4. Далее для каждой виртуальной машины из списка, составленного на 2 шаге, проверяется гипотеза, что её привязка к узлу с меньшим потреблением приведёт к уменьшению суммы

квадратов отклонения от средней загрузки узлов.

5. Если на 4 шаге было найдено несколько ВМ, удовлетворяющих условию, привязывается ВМ с наименьшим суммарным потреблением памяти не на узле назначения для уменьшения накладных расходов. В случае если таких ВМ не было найдено, никаких действий не выполняется.

В алгоритме используются эмпирически найденные константы для определения оптимального времени балансировки, процента использования межпроцессорной шины наиболее активными ВМ, граничные значения потребления ресурсов узлов, при которых балансировка не осуществляется. Значения перечисленных констант не важны для понимания алгоритма, однако составляют часть коммерческой тайны.

Результаты

Для реализации был выбран программный комплекс Virtuozzo, который состоит из следующих основных частей:

- ядро операционной системы linux;
- гипервизорный модуль kvm для работы с аппаратной виртуализацией собранный в составе ядра;
- менеджер управления ресурсами виртуальных окружений vcmmd;
- гипервизор пользовательского пространства QEMU.

Описанный алгоритм был реализован в виде политики управления ресурсами для менеджера vcmmd. Также в рамках работы было проведено сравнительное исследование производительности алгоритмов:

1. Vanilla – пассивный алгоритм с выбором NUMA узла для выделения новой памяти ВМ в зависимости от статистики потребления вычислительного времени виртуальной машиной. Реализация – использование ядра linux с CFS в роли менеджера машинного времени.
2. FFbind – статическое распределение ВМ по узлам на основании экспертного решения человека.
3. Perf – разработанный в данной работе алгоритм.

В приведённых ниже графиках ширина линии соответствует доверительному интервалу, вычисленному по трём измерениям для каждой точки.

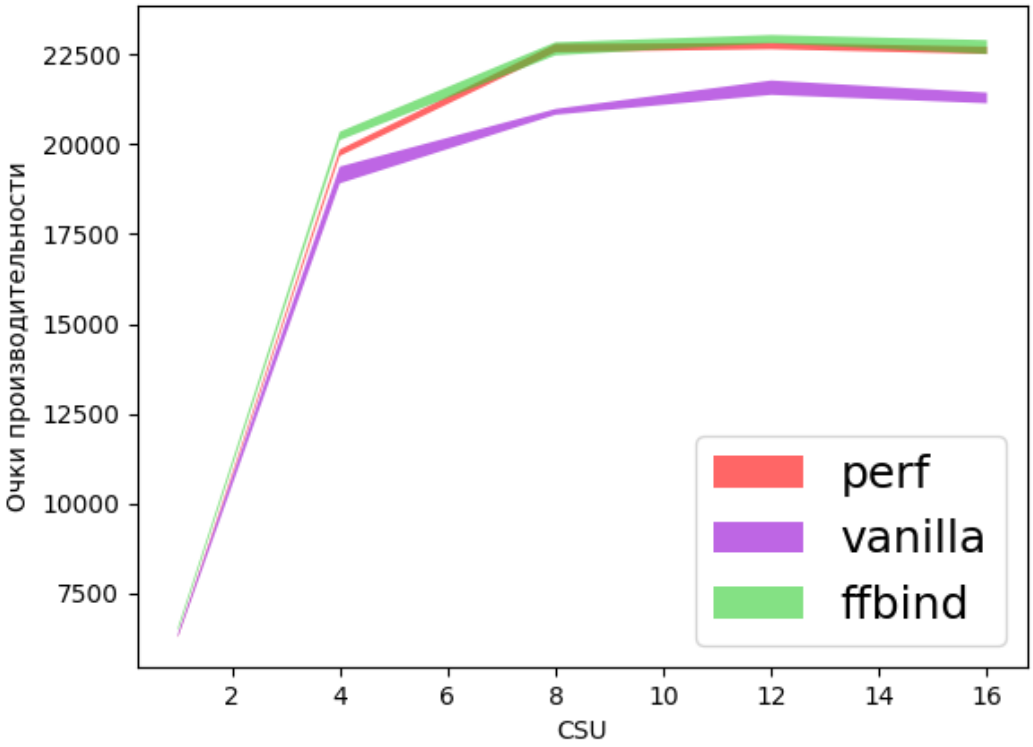

Рис.6. Результаты теста vConsolidate.

На рисунках 6 и 7 представлены оценки работы алгоритмов в ходе теста vConsolidate в абсолютных значениях и относительно работы алгоритма CFS. На рисунках 8 и 9 приведены аналогичные графики для теста SPECvirt. Стоит отметить, что уже при малом количестве CSU (3 для SPECvirt и 4 для vConsolidate) использование базового метода приводило к равномерному распределению памяти ВМ по NUMA узлам и просадке производительности.

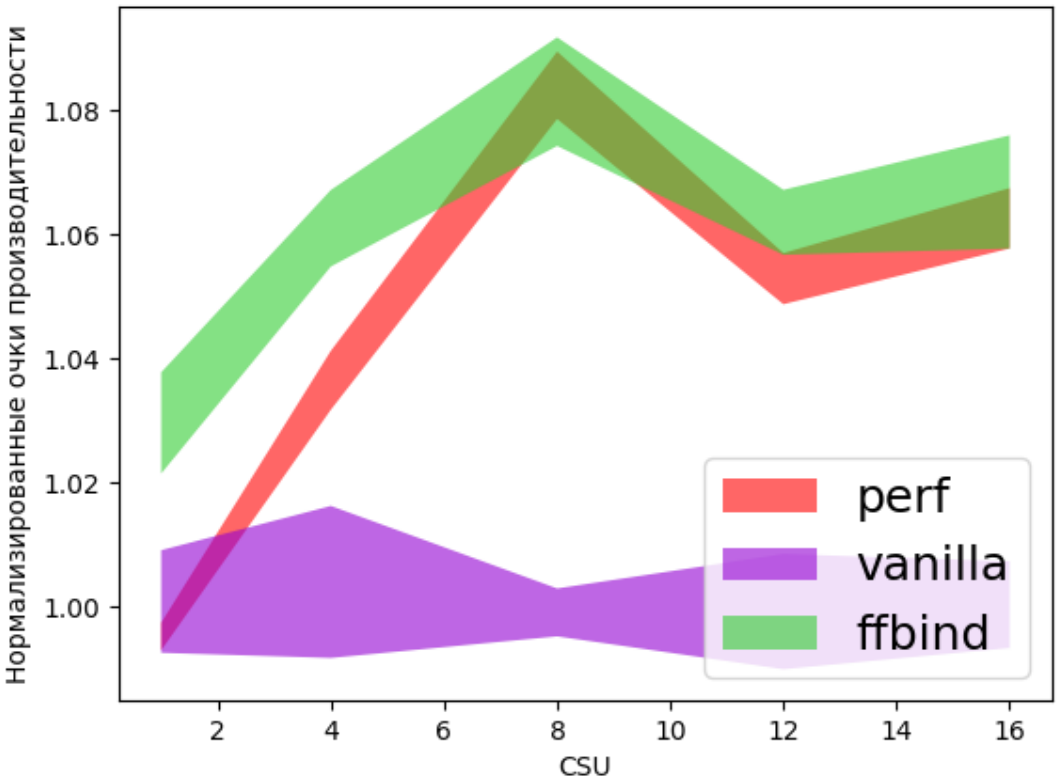

Рис.7. Результаты теста vConsolidate, нормированные на результат CFS.

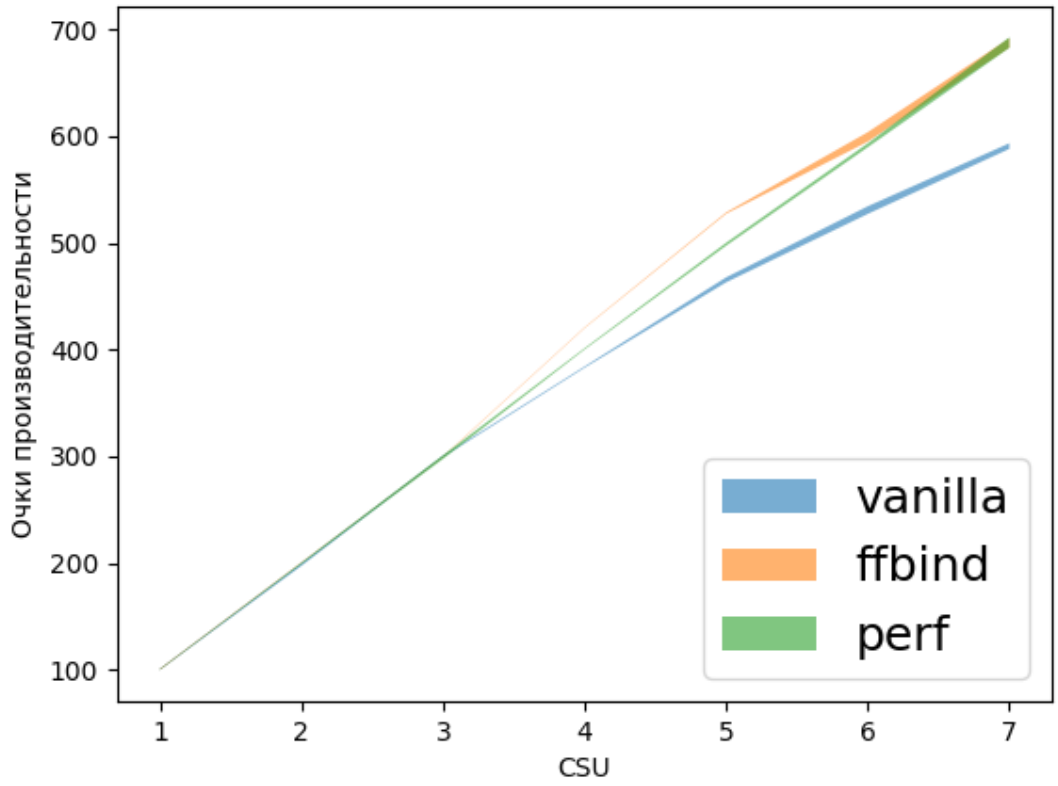

Рис.8. Результаты теста SPECvirt.

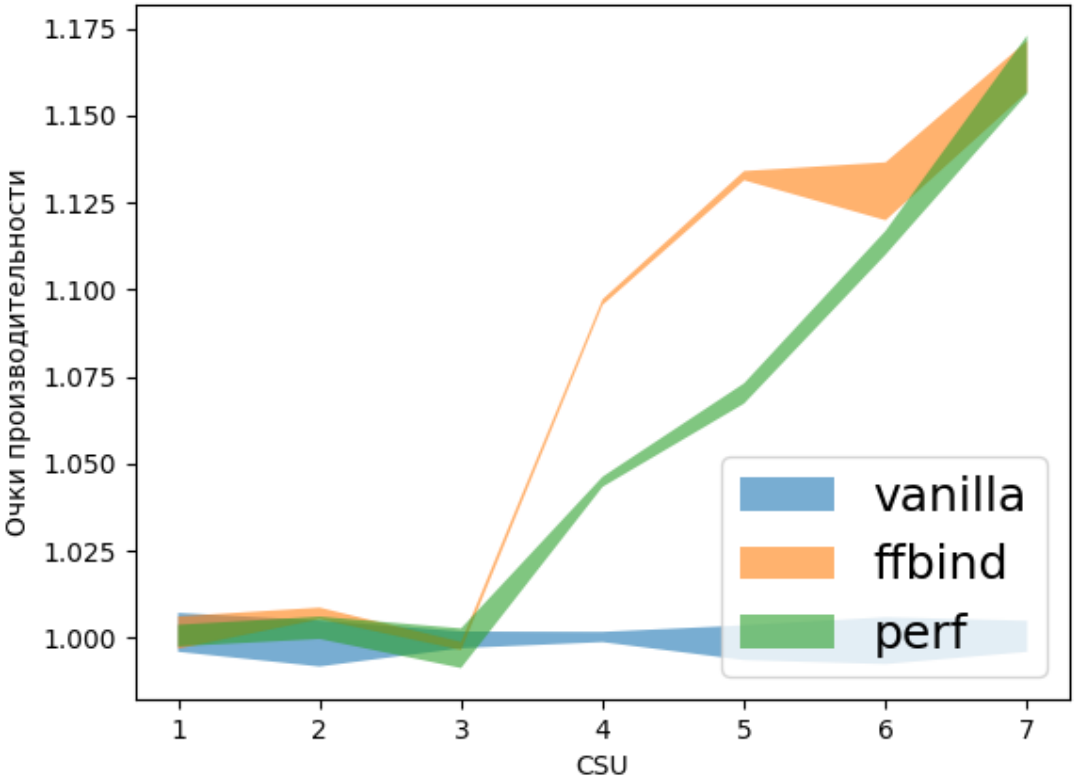

Рис.9. Результаты теста SPECvirt, нормированные на результат CFS.

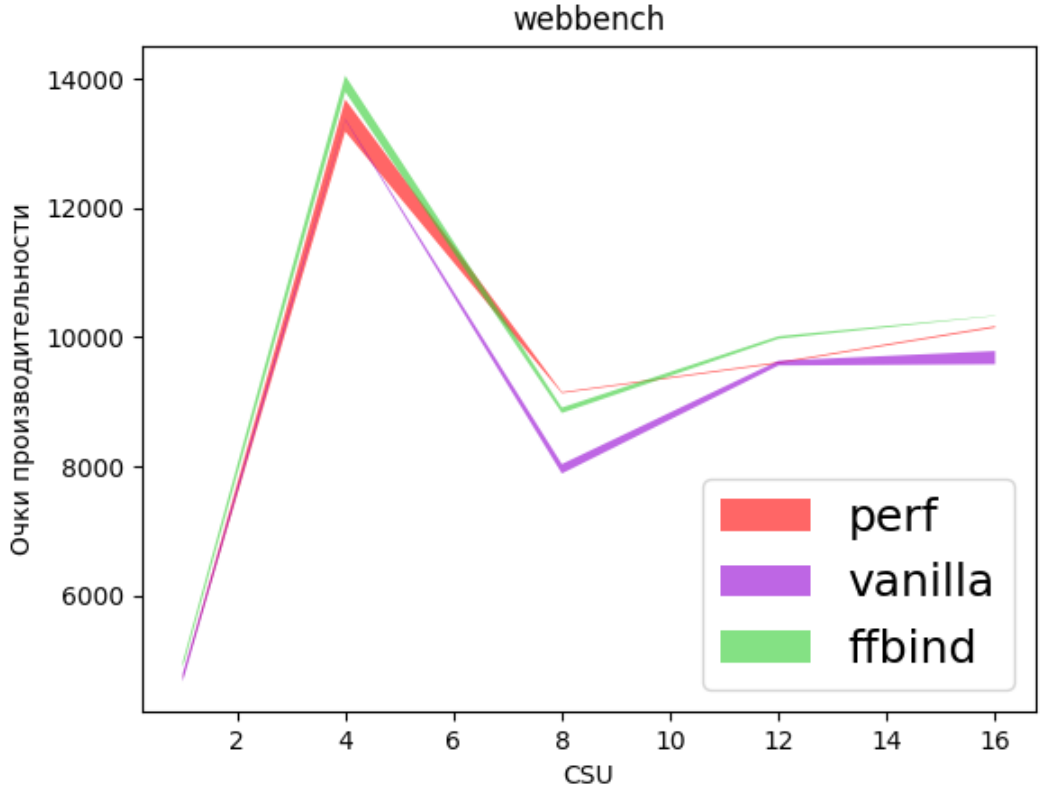

Рис. 10. Результаты теста WebBench (часть vConsolidate).

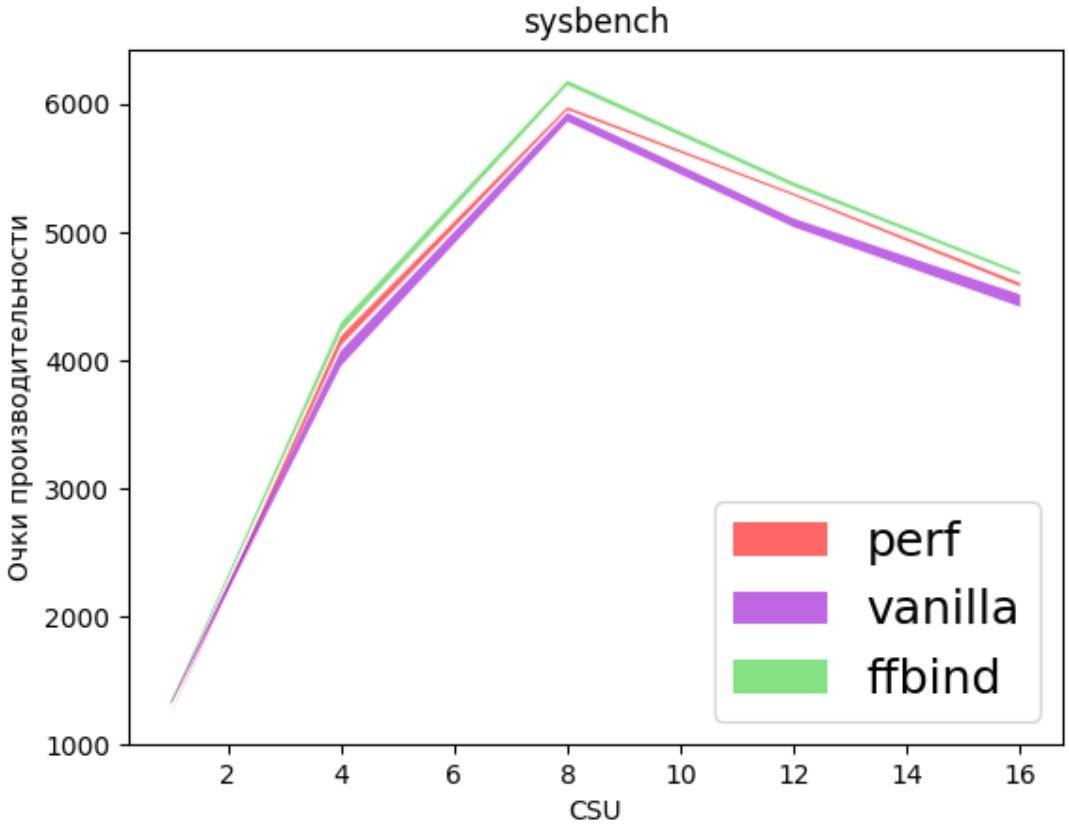

Рис. 11. Результаты теста Sysbench (часть vConsolidate).

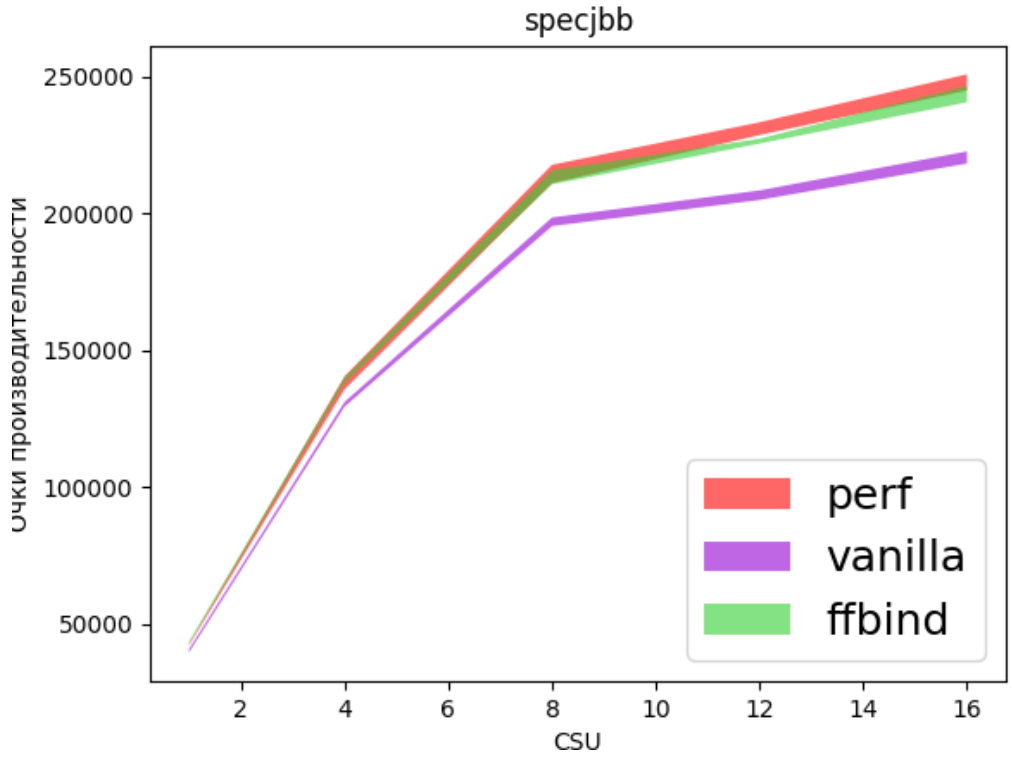

Рис. 12. Результаты теста Sysbench (часть vConsolidate).

Выводы

В ходе работы был разработан и реализован алгоритм балансировки нагрузки от виртуальных окружений на NUMA узлы. Данный алгоритм позволил получить 5% прирост производительности на нагрузке, приближенной к реальной, и достичь 90% эффективности размещения, выполненного экспертом. Данный результат равен или незначительно превосходит эффект, описанный в изученных статьях, но достоверно сравнить с ними не представляется возможным в силу разных виртуализационных платформ в случае метода взвешенной рандомизированной миграции ВЦПУ и отсутствия в открытом доступе реализации алгоритма улучшения размещения виртуальных машин с учётом трафика. Малый прирост производительности можно объяснить среди прочего прогрессом аппаратной платформы и тенденцией к снижению затрат на доступ к памяти чужого NUMA узла. Однако ограниченная пропускная способность межпроцессорной шины оставляет проблему балансировки актуальной, на что указывает работы исследователей из Университета Британской Колумбии[22].

О результатах исследованиях был проведён доклад на 59 научной конференции МФТИ, а разработанный алгоритм используется в продукте Virtuozzo 7.

Список литературы

1. Mell P., Grance T. The NIST definition of cloud computing. – 2011.
2. Armbrust M. et al. A view of cloud computing //Communications of the ACM. – 2010. – Т. 53. – №. 4. – С. 50-58.
3. Marston S. et al. Cloud computing—The business perspective //Decision support systems. – 2011. – Т. 51. – №. 1. – С. 176-189.
4. Tang L. et al. Optimizing Google's warehouse scale computers: The NUMA experience //High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on. – IEEE, 2013. – С. 188-197.
5. Drepper U. What every programmer should know about memory //Red Hat, Inc. – 2007. – Т. 11. – С. 2007.
6. Rao J. et al. Optimizing virtual machine scheduling in numa multicore systems //High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on. – IEEE, 2013. – С. 306-317.
7. Сайт F. D. N. NUMA Deep Dive Part 1: From UMA to NUMA. http://frankdenneman.nl/2016/07/07/numa-deep-dive-part-1-uma-numa/
8. Dash M., Mahapatra A., Chakraborty N. R. Cost effective selection of data center in cloud environment //International Journal on Advanced Computer Theory and Engineering (IJACTE). – 2013. – Т. 2. – С. 2319-2526.
9. Rimal B. P., Choi E., Lumb I. A taxonomy and survey of cloud computing systems //INC, IMS and IDC. – 2009. – С. 44-51.

10. Cheng Y. et al. Performance-Monitoring-Based Traffic-Aware Virtual Machine Deployment on NUMA Systems //IEEE Systems Journal. – 2015.

11. Jiang Y. et al. Analysis and approximation of optimal co-scheduling on chip multiprocessors //Proceedings of the 17th international conference on Parallel architectures and compilation techniques. – ACM, 2008. – С. 220-229.

12. Hollowell C. et al. The Effect of NUMA Tunings on CPU Performance //Journal of Physics: Conference Series. – IOP Publishing, 2015. – Т. 664. – №. 9. – С. 092010.

13. Ali S. Troubleshooting Tools //Practical Linux Infrastructure. – Apress, 2015. – С. 255-281.

14. Alrokayan M., Smith W., Kurnia S. Understanding How Start-ups Gain a Competitive Advantage from Cloud Computing //4TH ANNUAL DOCTORAL COLLOQUIUM. – 2016. – С. 64.

15. Khodashenas P. S. et al. Service provisioning and pricing methods in a multi-tenant cloud enabled RAN //Standards for Communications and Networking (CSCN), 2016 IEEE Conference on. – IEEE, 2016. – С. 1-6.

16. Bailey D. H. et al. The NAS parallel benchmarks summary and preliminary results //Supercomputing, 1991. Supercomputing'91. Proceedings of the 1991 ACM/IEEE Conference on. – IEEE, 1991. – С. 158-165.

17. Casazza J. P., Greenfield M., Shi K. Redefining server performance characterization for virtualization benchmarking //Intel Technology Journal. – 2006. – Т. 10. – №. 3.

18. Makhija V. et al. VMmark: A Scalable Benchmark for Virtualized Systems: Technical Report VMware-TR-2006-002 // bearing a date of. 2006. P. 1–13.

19. Fujitsu. Benchmark Overview VMmark V1. 2011. 6 p.

20. Fujitsu. Benchmark Overview VMmark V2. 2015. 9 p.

21. Сайт S. P. E. C. SPEC releases server virtualization benchmark https://www.spec.org/virt_sc2010/press/release.html

22. Dashti M. et al. Traffic management: a holistic approach to memory placement on NUMA systems //ACM SIGPLAN Notices. – ACM, 2013. – Т. 48. – №. 4. – С. 381-394.

23. Diener M., Cruz E. H. M., Navaux P. O. A. Locality vs. Balance: Exploring data mapping policies on NUMA systems //Parallel, Distributed and Network-Based Processing (PDP), 2015 23rd Euromicro International Conference on. – IEEE, 2015. – С. 9-16.

24. Garey M. R., Johnson D. S. Computers and intractability, vol. 174. – 1979.