УДК 004.772

П. О. Тихомиров [1], **П. В. Емельянов** [2], **Н. С. Плотник** [1], **А. В. Зырянов** [1]

[1] *Московский физико-технический институт*
Институтский пер., 9, Долгопрудный, 141700, Россия

[2] *Параллелс*
Алтуфьевское ш., 44, Москва, 127566, Россия

snorcht@gmail.com, xemul@parallels.com, nikolay.plotnik@phystech.edu, and@swsoft.mipt.ru

МИНИМИЗАЦИЯ ВРЕМЕНИ ПРОСТОЯ ПРОЦЕССОВ ПРИ ИХ МИГРАЦИИ В ОБЛАЧНОМ ХОСТИНГЕ [*]

Предлагается метод, позволяющий минимизировать время простоя пользовательских процессов при их миграции в облачном хостинге. Метод основывается на реализации двух алгоритмов: прогнозирования момента окончания миграции и уменьшения размера требуемой оперативной памяти при выполнении миграции. Приводятся результаты экспериментов, подтверждающие сокращение времени останова процессов и уменьшение размера требуемой памяти при выполнении миграции процессов с использованием предлагаемого метода.

Ключевые слова: живая миграция, итеративная миграция процессов, дедупликация памяти, прогнозирование момента окончания миграции.

Введение

В проектах, предоставляющих сервисы для магистральных и провайдерских сетей, для почтовых служб и высоконагруженных веб-ресурсов, возникает задача переноса данных и порожденных процессов с одного физического сервера на другой без длительного прекращения работы процессов и останова сервисов. Такой перенос данных и процессов принято называть живой миграцией пользовательских процессов.

Основной целью при организации миграции процессов является обеспечение непрерывного их выполнения, чтобы в период миграции работающего приложения замедление в его работе не вызвало негативную реакцию пользователя. Характеристикой такого замедления выступает время простоя, в течение которого мигрирующий процесс не выполняется. Время простоя складывается из суммы времен, затрачиваемых на останов процесса на исходном сервере, передачу его состояния на целевой сервер с его последующим восстановлением с точки останова.

Наиболее распространенным способом снижения времени простоя процесса при миграции является итеративный метод (итеративная миграция), суть которого в следующем. Полная остановка процесса на исходном сервере производится в момент, когда большая часть памяти, занимаемой процессом, в виде последовательности снимков (итераций) уже передана на целевой серверный узел. После остановки процесса на целевой серверный узел досылается только та часть памяти, которая изменилась за время последней итерации.

[*] Работа выполнена в Московском физико-техническом институте (ГУ) при финансовой поддержке Министерства образования и науки Российской Федерации (договор № 02.G25.31.0054).

Тихомиров П. О., Емельянов П. В., Плотник Н. С., Зырянов А. В. Минимизация времени простоя процессов при их миграции в облачном приложении // Вестн. Новосиб. гос. ун-та. Серия: Информационные технологии. 2014. Т. 12, вып. 4. С. 112–120.

Следует заметить, что выбор момента полного останова процесса при итеративной миграции оказывает существенное влияние на время простоя процесса, так как от него (момента останова) зависит количество выполняемых итераций и, следовательно, количество передаваемых снимков памяти процесса по сети.

К недостатку итеративной миграции следует отнести значительный расход ресурсов памяти на целевом серверном узле, так как с каждой итерацией передаются на хранение снимки памяти процесса.

Основные способы профилирования итеративной миграции

Существуют различные способы реализации итеративной миграции. В большинстве случаев все действия по сохранению и восстановлению процессов реализуются в ядре операционной системы, что является недостатком таких решений. Расширение функциональных возможностей на уровне основного ядра становится трудоёмким из-за необходимости постоянной поддержки кода ядра и, соответственно, программных приложений, использующих новую возможность. Примеры таких решений можно найти в OpenVZ, Berkeley Lab Checkpoint/Restart, Linux Checkpoint/Restart by Oren Laadan [1].

В ряде решений применяется альтернативный подход, при котором функция сохранения и восстановления процессов отводится специальной библиотеке, которая вызывается либо предварительно, либо динамически в процессе работы приложения. Вариант подхода с предварительным вызовом библиотеки сложно использовать при работе с приложениями, имеющими закрытый код. Динамическое включение дополнительной библиотеки может отрицательно повлиять на скорость процесса миграции. Примеры использования такого подхода можно найти в Berkeley Lab Checkpoint / Restart и Distributed MultiThreaded CheckPointing [2].

Для реализации итераций при миграции необходим контроль текущего состояния отведенной процессу памяти. Существующие способы мониторинга с использованием приложений ps, page-collect недостаточно точны и не применимы для анализа памяти процесса в произвольный момент времени. Такие способы позволяют получить информацию лишь об общем размере отведенной процессу виртуальной памяти, а также о размере предоставленной физической памяти, что не может быть достаточным для получения результатов полного анализа.

В апреле 2013 г. в ядро Linux для решения CRIU (англ. Checkpoint / Restore In Userspace) [3] была добавлена новая функциональная возможность soft-dirty tracking для отслеживания занимаемой процессом памяти. Это позволило включить мониторинг памяти на уровне ядра системы и, следовательно, получить битовую маску памяти процесса, используя файловую систему procfs. В результате появилась возможность выделять изменяющиеся страницы памяти из общего объема памяти, отведенного для выполнения миграции. Последнее дало возможность проводить мониторинг насыщения рабочего набора, необходимого для итеративной миграции. Эта функциональная особенность была использована в исследованиях, приведенных в данной статье.

Метод минимизации времени простоя при итеративной миграции процессов

Цель данной статьи – представить разработанный метод (вариант) итеративной миграции пользовательских процессов в хостинге, направленный на минимизацию времени простоя, а также результаты экспериментов, подтверждающие его эффективность. Разработанный метод включает одновременную реализацию двух алгоритмов:

---

[1] См.: Windows Server 2008 R2 & Microsoft Hyper-V Server 2008 R2 – Hyper-V Live Migration Overview & Architecture; *Duell J.*, *Hargrove P.*, *and Roman E.* The Design and Implementation of Berkeley Lab's Linux Checkpoint/Restart.Berkeley Lab Technical Report (publication LBNL-54941), December 2002.

[2] См.: Linux-CR: Transparent Application Checkpoint-Restart in Linux; DMTCP: Transparent Checkpointing for Cluster Computations and the Desktop.

[3] URL: http://www.criu.org.

- прогнозирования момента окончания итеративной миграции;
- минимизации размера оперативной памяти, необходимой для получения, хранения снимков памяти и восстановления процесса на целевом узле при выполнении итеративной миграции.

Работа первого алгоритма направлена на минимизацию общего времени простоя процесса, второго – на ограничение размера требуемой оперативной памяти до величины полного рабочего набора процесса, необходимого для сборки и восстановления процесса с сохранением его целостности.

Алгоритм прогнозирования
момента окончания итеративной миграции

Предлагаемое решение алгоритма прогнозирования момента окончания итеративной миграции. Предлагаемое решение по прогнозированию момента окончания миграции основывается на эвристическом методе с использованием CRIU. Следует заметить, что в некоторых существующих решениях миграции процессов уже используются методы [4] с применением следующих ключевых эвристик.

1. Ограничение максимального количества итераций. Применение этого метода позволяет предотвратить длительное «зависание» процесса миграции, но увеличивает вероятность незавершенности процесса горячей миграции.

2. Остановка процесса из-за сбоя в системе. В случае если на какой-то итерации объем переданной памяти оказывается меньше объема «модифицированной» памяти, т. е. измененной за время итерации, процесс миграции останавливается. Недостатком этого метода является высокая вероятность раннего завершения итераций без достижения оптимума из-за возможных случайных флуктуаций насыщенности рабочего набора.

3. Остановка процесса миграции при достижении необходимого минимального количества страниц памяти. Метод позволяет обрабатывать малое количество измененных страниц при успешном завершении процесса миграции. Недостаток его заключается в том, что граничное значение задается статически и не является какой-либо характеристикой состояния соединения.

В настоящей работе для прогнозирования момента окончания итеративной миграции предложено использовать функцию насыщения рабочего набора процесса, которая оценивается на основе данных мониторинга soft-dirty tracking CRIU. Полученная функция используется при моделировании процесса итеративной миграции. На основе критерия минимизации суммарного времени простоя при миграции вычисляется оптимальное значение количества необходимых итераций.

Оценка и аппроксимация функции насыщения. Для оценки функции насыщения по экспериментальным данным был использован метод наименьших квадратов:

$$M(t) = \frac{a\,t + b}{c\,t + d}, \tag{1}$$

где t – время миграции;

a, b – параметры оценочной функции изменений в памяти во время работы процесса;

c, d – параметры оценочной функции повторного и последующих изменений в «модифицированной» памяти;

M – размер «модифицированной» памяти.

Учитывая ограниченность графика насыщения рабочего набора, можно допустить, что $c = 1$. Кроме того, можно считать, что $b = 0$, так как изначально график проходит через $t = 0$, $M = 0$.

С учетом этих допущений функция (1) принимает вид

$$M(t) = \frac{a\,t}{t + d}$$

[4] См.: OpenVZ Documentation. https://wiki.openvz.org/Checkpointing_and_live_migration; KVM Live Migration Uri Lublin, Qumranet, Anthony Liguori, IBM.

Пусть $(t_1, m_1), \ldots, (t_n, m_n)$ являются экспериментальными данными, тогда условие минимума суммы квадратов отклонений можно записать в следующем виде

$$F = \sum_{i=1}^{n} \left[m_i \frac{a\, t_i}{t_i + d} \right]^2 \rightarrow min\ , \tag{2}$$

где $i = 1$, n – текущая итерация в заданном интервале $[1; n]$.

Условие того, что значение производной в точке минимума становится равным нулю, можно записать в виде

$$\frac{F}{d} = \sum_{i=1}^{n} 2 \left[m_i \frac{a\, t_i}{t_i + d} \right] \frac{a\, t_i}{(t_i + d)^2}\ . \tag{3}$$

Уравнение (3) нельзя решить аналитически. Для его решения следует использовать алгоритм Левенберга – Марквардта.

Модель процесса итеративной миграции. Выражения (2) и (3) позволяют получить приближенную функцию насыщения рабочего набора, на основании которой можно смоделировать процесс миграции при условии, что эта функция не меняется во времени. На каждой итерации есть некоторый размер памяти, накапливаемый для передачи. Время передачи данных этой памяти можно оценить, исходя из размера памяти и средней скорости передачи данных соединения. Используя приближенную функцию по полученному времени можно определить объем новых данных, который будет накоплен к началу следующей итерации. В результате будет построена модель реального итеративного процесса, происходящего при итеративной миграции.

Итеративный процесс имеет сходимость. Ввиду того, что приближенная функция насыщения рабочего набора процесса является непрерывной, монотонно возрастающей и ограниченной сверху ($M(t) < 1$), получаемая последовательность времен передачи данных на каждой итерации будет монотонно убывающей и ограниченной снизу нулем. Используя определенное количество итераций, можно найти приближенный инфимум времени, необходимый для передачи образов памяти, а также количество итераций для его достижения с некоторой заданной точностью.

На рис. 1 изображен пример построенного графика модели миграции с начальным объемом данных около 1 ГБ и скоростью передачи по сети 300 МБ/с. Кривая графика показывает сходимость объема памяти к значению 200 МБ. При меньшей скорости соединения процесс будет сходиться к большему значению объема памяти.

Оптимизация процесса. При построении модели итераций не учитывалось время, затрачиваемое на получение снимка на каждой итерации. Учитывая этот параметр при построении модели, можно определить суммарное время простоя мигрируемого процесса.

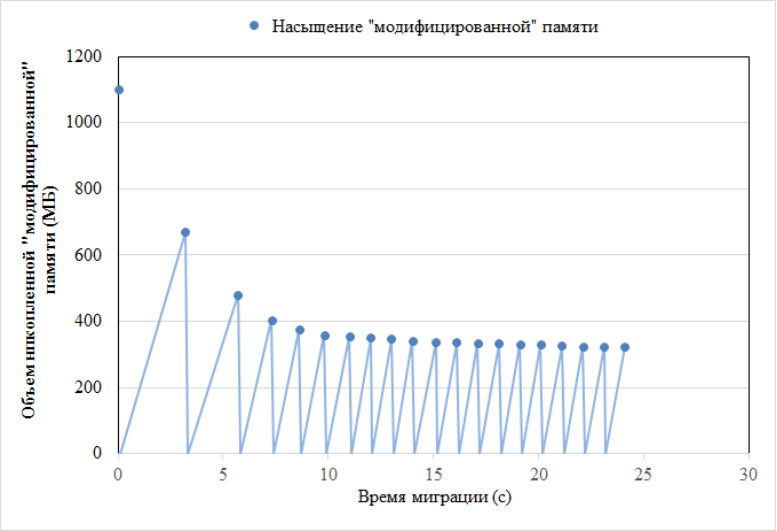

Рис. 1. График модели миграции

Время, затрачиваемое на получение снимка, можно считать неизменным от итерации к итерации, так как основная нагрузка при получении снимка создается за счет полного обхода памяти процесса и проверки каждой из страниц на ее изменение с момента предыдущего снимка. С учетом этого суммарное время простоя на i-й итерации $T(i)$ будет определяться как

$$T(i) = (i - 1)t^0 + t(i),$$

где t^0 – время простоя, затрачиваемое на получение снимка;

$t(i)$ – время для передачи «модифицированной» памяти на i-й итерации.

Так как последовательность $\{t(i)\}$ сходится к инфимуму при значении i, стремящемся к бесконечности, а величина $\{(i - 1)t^0\}$ при этом линейно возрастает, то можно утверждать, что у последовательности сумм $T(i)$ существует минимум.

Используя предложенную модель, можно определить количество итераций в процессе миграции, необходимое для минимизации суммарного времени простоя.

Проверка работы алгоритма прогнозирования момента окончания миграции. С целью проверки эффективности алгоритма было проведено экспериментальное тестирование на оборудовании с конфигурацией:

- Intel Core i73520M Ivy Bridge, 2900 МГц с 6 ГБ DDR3 SDRAM, хостовая ОС Ubuntu 14.04;
- Виртуальная машина KVM – 2 ГБ кэш, (64-bit) ядро Linux 3.11, дистрибутив Fedora 20.

Методика тестирования заключалась в выполнении следующей последовательности действий:

1) предварительное тестирование скорости соединения с получением данных работы сети;
2) установка тестового приложения, активно загружающее RAM;
3) исполнение тестового приложения на гостевой ОС;
4) определение оптимального количества итераций с помощью рабочего кода алгоритма;
5) построение графика модели миграции;
6) выполнение итеративной миграции тестового приложения на другую виртуальную машину с получением результатов.

Тестирование скорости передачи снимков процесса (образов) по сети показало ее линейную зависимость от объема передаваемых данных.

Для определения оптимального количества итераций был построен график зависимости времени простоя от количества итераций (рис. 2). Вертикальная ось графика показывает время простоя при миграции в секундах, горизонтальная ось – количество используемых итераций в работе алгоритма. Кривая графика суммарного времени простоя показывает, что оптимальным количеством итераций в эксперименте является 5.

Для сравнения результатов эксперимента с предсказаниями модели были построены два графика: модельный график миграции и график замера времени при реальной миграции (рис. 3).

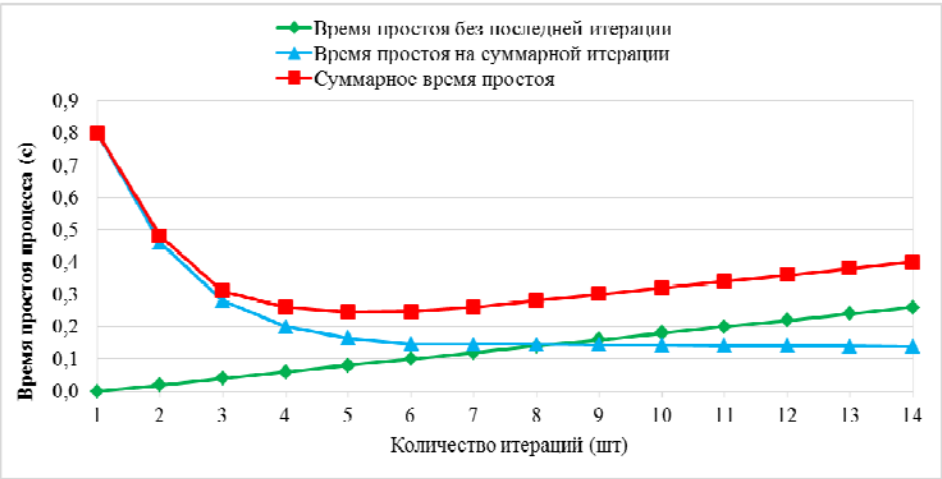

Рис. 2. График зависимости времени простоя от количества итераций

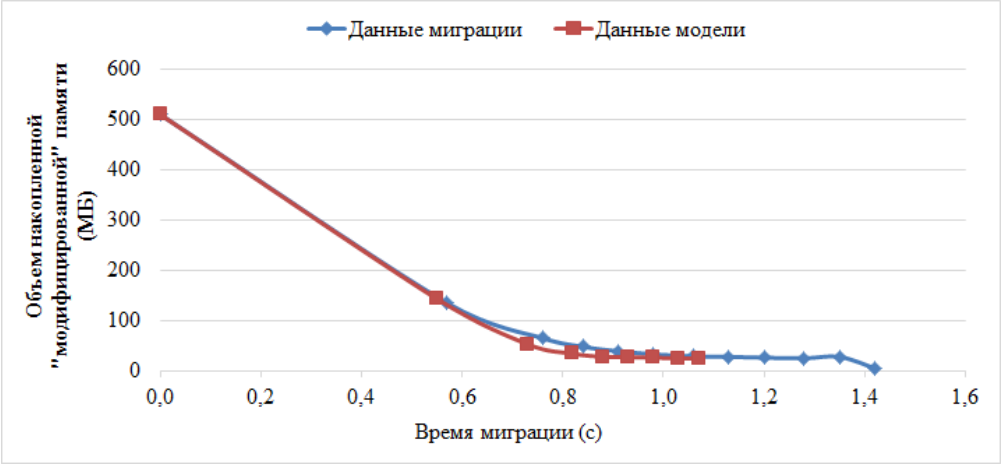

Рис. 3. Сравнение графиков, построенных на основе данных модели и реальной миграции

Вертикальная ось на рис. 3 показывает объем памяти, необходимый для сохранения перед выполнением процесса миграции («модифицированная» память). Горизонтальная ось показывает время выполнения миграции. На рисунке видно, что кривая графика смоделированного процесса практически совпадает с кривой графика реального процесса, что указывает на достаточную точность измерений в рамках предлагаемого алгоритма.

Для проверки эффективности работы предлагаемого решения замерялось время простоя при выполнении итеративных миграций (с произвольным и оптимальным количеством итераций в виртуальной сети, имеющую скорость 1 ГБ/с). В результат времени простоя входило время от начала миграции процесса на исходной машине до продолжения его выполнения на целевой. Усредненные результаты замеров показали, что даже в высокоскоростной сети миграция с произвольным числом итераций (время простоя 0,4) уступает по производительности миграции с оптимальным числом итераций (время простоя 0,25), что подтвердило эффективность предлагаемого решения.

Алгоритм дедупликации памяти с использованием CRIU

Предлагаемое решение алгоритма дедупликации памяти. Для минимизации размера оперативной памяти, необходимой при получении и хранении снимков памяти, а также при восстановлении процесса на целевом узле при выполнении итеративной миграции, предлагается эффективное решение – применение алгоритма дедупликации памяти процесса.

Работа алгоритма предполагает использование программного обеспечения CRIU (англ. Checkpoint / Restore In Userspace) для ОС семейства Linux, предоставляющее возможность внешнего создания контрольной точки для произвольного приложения, а также возобновления работы приложения с этой точки. Основной целью программного обеспечения CRIU является поддержка миграции отдельных процессов или их групп.

Большое количество итераций может создать высокую нагрузку на оперативную память целевого узла с последующим перемещением данных во внешнюю среду, что может существенно замедлить выполнение миграции. С целью устранения этого недостатка предлагается расширить функциональность программного обеспечения CRIU, а именно добавить возможность дедупликации данных снимков памяти.

Алгоритм дедупликации включает в себя поиск и удаление повторяющихся участков данных в памяти без ущерба для их точности и целостности. Целью этого процесса является уменьшение объема пространства, занимаемого данными. Файлы разбиваются на небольшие блоки размером не более 1 МБ, среди которых выявляются повторяющиеся, и для каждого повторяющегося блока оставляется только одна копия. Избыточные копии блока заменяются ссылками на единственную копию.

Эффективность предлагаемого метода миграции процессов с использованием алгоритма дедупликации основана на том, что для реального восстановления процесса необходимо хранить в памяти его образ размером, не превышающим размер памяти, выделенной процессу на исходном узле. Использование этой технологии позволяет сократить многократно увеличивающийся объем памяти на целевом узле от итерации к итерации до размера одного полного рабочего набора процесса.

При итеративной миграции происходит накопление данных в памяти, как за счет новых измененных страниц, так и за счет старых снимков, являющихся лишними, поскольку процесс живой миграции не предусматривает восстановление данных из устаревших образов. Последние используются только для восстановления неизмененных страниц на завершающей стадии процесса. Следовательно, освобождение памяти от старых страниц в родительских снимках не может оказать влияния на целостность миграции.

В основу работы алгоритма положен механизм Sparce-файлов (разряженных файлов), поддерживаемый файловой системой tmpfs. Алгоритм позволяет взять файл из оперативной памяти (или tmpfs) и в нужном месте освободить заданный объем памяти. При этом размер файла уменьшится с сохранением и расположением в нем необходимых блоков данных. При чтении этого файла лишняя информация будет отображаться в виде поля, заполненного нулями.

Для работы алгоритма в реальных условиях предлагается объединять смежные блоки для последующей чистки памяти за меньшее количество обращений. При этом размер объединенного блока не должен превышать 1 МБ. Это связано с тем, что девиация в 1 МБ является вполне допустимой величиной в современных технологиях.

На рис. 4 изображен пример работы алгоритма дедупликации памяти при создании снимка. Части пространства виртуальной памяти процесса показаны в виде горизонтально расположенных прямоугольников. Блоки памяти выполнены в виде отдельных прямоугольников.

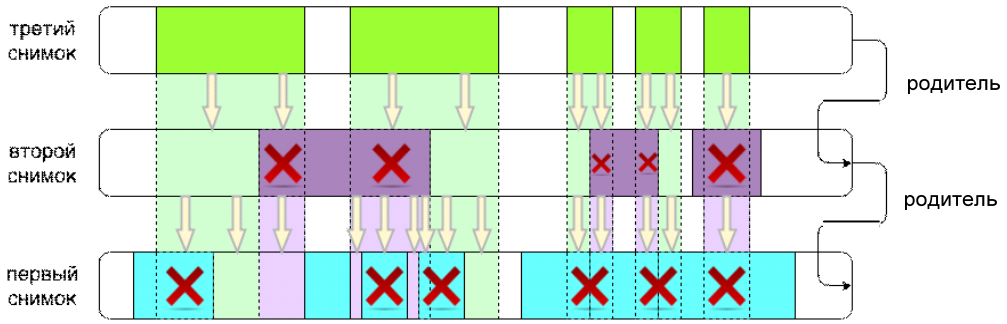

Рис. 4. Дедупликация 1-го и 2-го снимков при создании 3-го

При поиске нового блока памяти происходит последовательный просмотр снимков от последнего к первому. Просмотр начинается сразу после сохранения последнего снимка. В данном примере будут выполнены следующие действия:

1) поиск во втором снимке максимально пересекающегося с новым блоком свободного участка памяти или блока данных;
2) переход на следующий родительский снимок;
3) передача старой информации из блока данных в накопительный блок очистки.

Для каждого снимка предусмотрен накопительный блок очистки, в который записываются данные последовательно расположенных блоков размером до 1 МБ. По окончании записи выполняется одновременное удаление блоков. На рис. 4 видно, что на первом снимке отмеченные крестиками второй и третий блоки размещены в файле непосредственно друг за другом. Следовательно, они будут удалены одновременно.

Реализация алгоритма дедупликации памяти при итеративной миграции позволяет хранить в памяти снимки процессов размером, соизмеримым с объемом памяти, занимаемым самим процессом. Алгоритм обеспечивает прямое взаимодействие с оперативной памятью и оптимальное ее использование.

Тестирование работы алгоритма дедупликации памяти. С целью проверки эффективности предложенного решения по минимизации размера используемой памяти на целевом узле было проведено тестирование миграции процесса с применением и без применения алгоритма дедупликации памяти на оборудовании с конфигурацией:

- Intel Core i73520M Ivy Bridge, 2900 МГц с 6 ГБ DDR3 SDRAM, хостовая ОС Ubuntu 14.04;
- две виртуальные машины KVM (исходная и целевая) – 2 ГБ кэш, (64-bit) ядро Linux 3.11, дистрибутив Fedora 20.

Тестирование проводилось на наборе 4 видов функциональных тестов:

- run-snap-auto-dedup.sh [5];
- run-snap-dedup-on-restore.sh [6];
- run-snap-dedup.sh [7];
- run-snap-maps04.sh [8].

Кроме этого, была использована система автоматического тестирования ZDTM [9], адаптированная к подключению режима дедупликации.

Методика тестирования заключалась в выполнении последовательности действий:

1) монтирование файловой системы tmpfs для хранения снимков на целевой виртуальной машине;

2) установка тестового приложения, активно загружающего RAM исходной виртуальной машины;

3) выполнение двух итеративных миграций тестового приложения на целевую виртуальную машину с включенной и выключенной дедупликацией;

4) определение объема занимаемого образами пространства с помощью программы disk usage.

Были получены результаты в виде двух кривых графика (рис. 5), отображающие размеры занимаемой образами памяти при миграции с 20 итерациями без применения и с применением дедупликации. Вертикальная ось на рисунке показывает размер требуемой памяти, горизонтальная – время выполнения итеративной миграции. Кривые графика демонстрируют сокращение занимаемого образами пространства памяти при итеративной миграции в случае применения метода дедупликации.

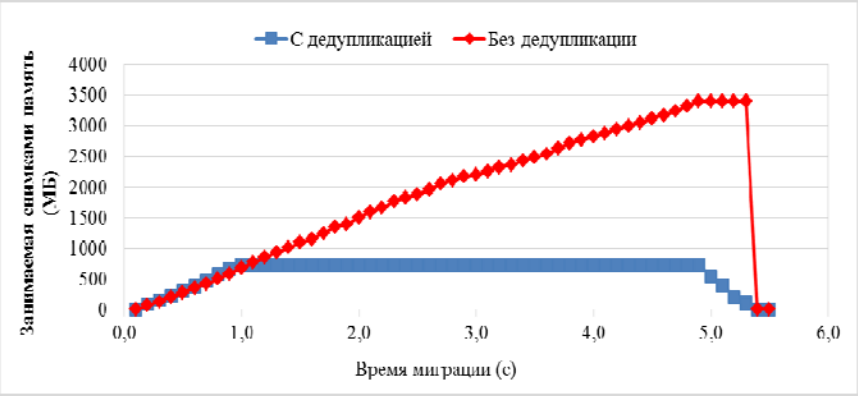

Рис. 5. График сравнения объемов памяти, занимаемых снимками при миграции с применением и без применения дедупликации

[5] URL: https://github.com/xemul/criu/blob/master/test/mem-snap/run-snap-auto-dedup.sh.
[6] URL : https://github.com/xemul/criu/blob/master/test/mem-snap/run-snap-dedup-on-restore.sh.
[7] URL : https://github.com/xemul/criu/blob/master/test/mem-snap/run-snap-dedup.sh.
[8] URL: https://github.com/xemul/criu/blob/master/test/mem-snap/run-snap-maps04.sh.
[9] URL: http://criu.org/ZDTM_Test_Suite.

Выводы

1. Результаты тестирования подтверждают эффективную работу алгоритмов прогнозирования момента окончания итеративной миграции и дедупликации памяти для хранения снимков процесса и его восстановления.

2. Предложенная модель итеративной миграции на основе аналитической аппроксимации динамики рабочего набора процесса позволяет использовать спрогнозированное количество итераций для выполнения живой миграции процессов.

3. Механизм мониторинга по насыщению рабочего набора и его визуализации обеспечивает получение достоверной информации.

4. Применение алгоритма дедупликации позволяет снизить объем требуемой памяти для хранения снимков процесса до размеров полного рабочего набора восстанавливаемого процесса и не допускает увеличения времени простоя процесса на этапе восстановления при итеративной миграции.

Предложенный в статье метод минимизации времени простоя процесса при выполнении итеративной миграции может быть применен при решении задач:

- построения компьютерных систем высокой доступности;
- балансировки нагрузки на сервисные узлы кластера, когда необходимо мигрировать часть ресурсов с перегруженного узла на запасной или менее нагруженный сервисный узел без заметного побочного эффекта для пользователей;
- технического и программного обслуживания сервисного узла кластера без останова запущенных на нем процессов.

Материал поступил в редколлегию 11.12.2014

P. O. Tikhomirov [1], **P. V. Emelyanov** [2], **N. S. Plotnik** [1], **A. V. Zyryanov** [1]

[1] *Moscow Institute of Physics and Technology*
Institutskii per., *9*, *Dolgoprudny*, *141700*, *Russian Federation*

[2] *Parallels*
Altufevskoe sh. 44, *Moscow*, *127566*, *Russian Federation*

snorcht@gmail.com, *xemul@parallels.com*, *nikolay.plotnik@phystech.edu*, *and@swsoft.mipt.ru*

MINIMIZING DOWNTIME PROCESSES DURING THEIR MIGRATION IN THE CLOUD HOSTING

In this article it is proposed a method to minimize downtime custom processes while they migrate in the cloud hosting. The method is based on the realization of two algorithms: prediction of the end of the migration and reducing the size of the required memory when the migration occurs. In the article you can find experimental results confirming the reduction of downtime processes and reducing the size of memory required when performing the migration process using the proposed method.

Keywords: live migration, iterative process migration, deduplication storage, forecasting the end of the process migration.