Федеральное государственное автономное образовательное учреждение высшего профессионального образования “Московский физико-технический институт (государственный университет)”

На правах рукописи

МЕЛЕХОВА Анна Леонидовна

УПРАВЛЕНИЕ ФИЗИЧЕСКОЙ ПАМЯТЬЮ ВИРТУАЛЬНОЙ МАШИНЫ

Специальность 05.13.11 – математическое и программное обеспечение вычислительных машин, комплексов и компьютерных сетей

Диссертация на соискание
ученой степени кандидата технических наук

Научный руководитель
д.ф.-м.н. Тормасов А.Г.

Москва — 2015

Оглавление

Введение

Актуальность темы

За последнюю декаду виртуализация перестала быть узкопрофессиональной технологией с ограниченным применением. Виртуализационные решения появились в большинстве популярных операционных систем, как-то: Microsoft Windows Hyper-V[1], Linux KVM [2], Apple OS X hypervisor[3]. Производители аппаратного обеспечения один за другим также добавляют поддержку виртуализационных технологий: Intel VT-x [4], AMD AMD-V [5], ARM [6]. И даже крупные производители видеокарт предлагают свои виртуализационные решения – примером тому NVIDIA GRID[7] и Intel gVirt [4].

Одним из основных применений виртуализации является консолидация серверов [8]. Такое применение обосновано пикообразным характером большинства современных рабочих нагрузок [9]. Под пикообразным характером понимается, что основную часть времени используется лишь небольшая доля ресурса, но в определенные моменты этого же объема ресурса перестает хватать, возникают существенные падения производительности. Традиционный подход при фиксированных ресурсах предполагает компромисс между пиковой производительностью и стоимостью ресурсов в простое. С помощью виртуализации становится возможным разместить несколько виртуальных серверов на одном физическом и ограничить их в фактическом потреблении ресурса. Это увеличивает утилизацию ресурсов, так как неиспользуемые ресурсы одного виртуального сервера могут быть отданы другому виртуальному серверу. Кроме того, такая консолидация снижает затраты на обслуживание серверов [10,11]. Также подобное «уплотнение» позитивно сказывается на общем потреблении энергии на питание и охлаждение серверов. [12,13].

При виртуализации операционная система, исполняющая внутри виртуальной среды, – гостевая ОС (рис. 1) – будет полагать, что в ее

распоряжении вся полнота назначенного ресурса. Но прозрачно для нее виртуализационное программное обеспечение – монитор виртуальной машины (ВММ) или гипервизор (рис. 1) - может изменять объем выделенного ресурса. Этот процесс можно назвать мультиплексированием или виртуализацией ресурса. Хорошей аналогией здесь являются процессы, исполняющиеся в системе. Прозрачно для них ОС распределяет память, вычислительную мощность и пропускную способности сети в соответствии с квотами и фактическими потребностями. Так и гипервизор управляет виртуальными машинами (под виртуальной машиной мы подразумеваем комплекс из гостевой ОС, виртуальной среды исполнения и монитором виртуальной машины, рис. 1), перераспределяя ресурсы в соответствии с их реальным потреблением.

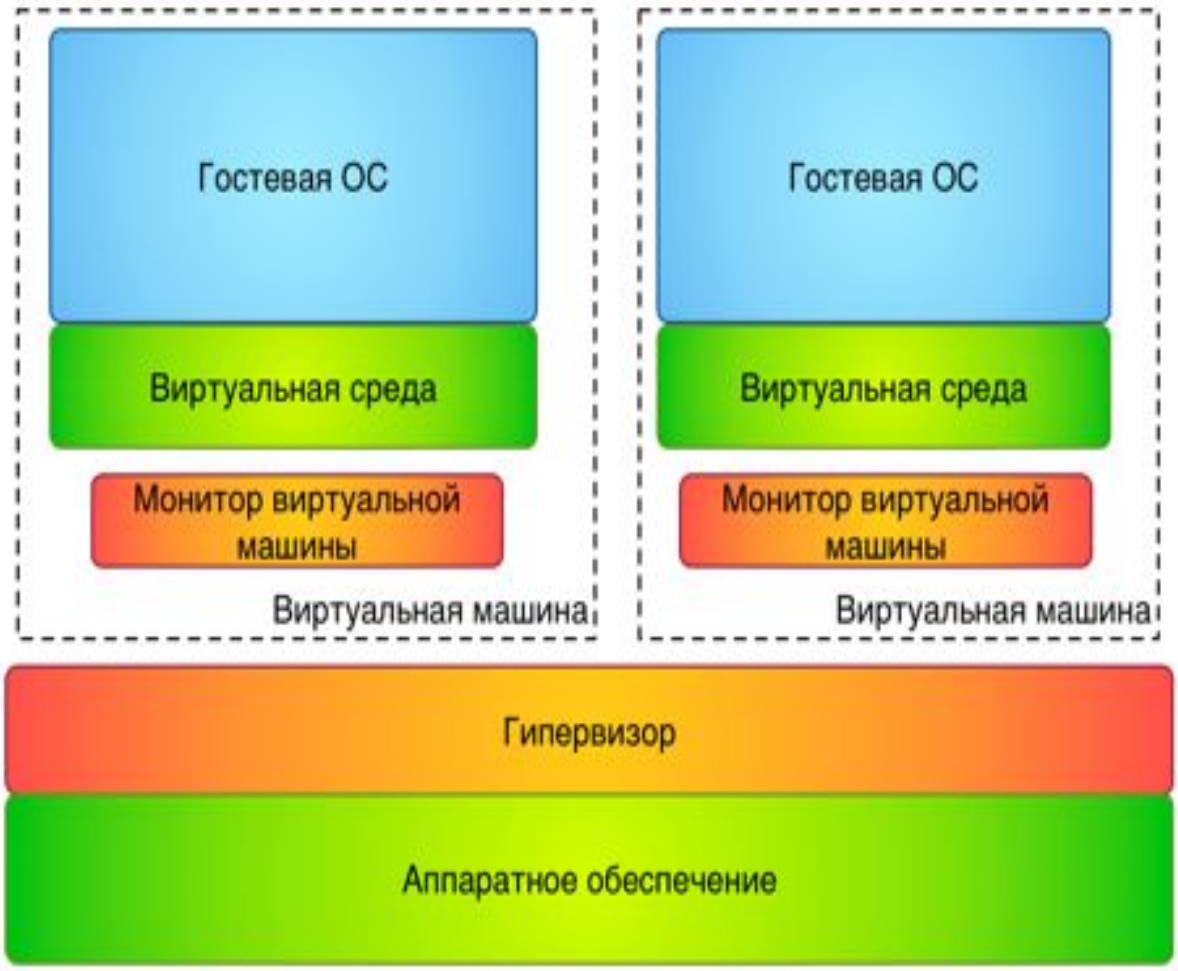


Рис. 1. Виртуальная машина

Ограничение виртуальных машин в ресурсах возникает, когда суммарный объем назначенных виртуальных ресурсов превосходит доступный объем ресурса на физической машине. Этот режим работы получил названия переподписки (oversubscription) [8] или переназначения (overcommitment) [14]. В том случае, когда суммарная потребность в виртуальных ресурсах превосходит доступный физический ресурс, возникает ситуация перегрузки (overload) [15]. При перегрузке часть виртуальных машин либо останавливается, либо мигрирует на другие менее загруженные физические машины (в случае облачной инфраструктуры) [15].

Для реализации режима переподписки необходимо эффективное распределение ресурса между потребителями (виртуальными машинами). Для этого необходимо оценить фактический уровень использования ресурса и выделить ресурс соответственно оценке. При этом нужно учитывать, что алгоритмы оценки зависят от вида ресурса, а точнее от методов обращения с ними гостевых операционных систем. Так, например, современные операционные системы крайне аккуратно обращаются с «неиспользуемым» процессорным временем, исполняя специальные инструкции – HALT-ы [15] – для информирования о простое (idle). На реальном аппаратном обеспечении эти команды используются для снижения энергозатрат. Но этот же HALT есть индикатор для монитора виртуальной машины о том, что квант процессорного времени может быть выделен другой гостевой ОС, другому потребителю.

Совершенно другой подход применяется по отношению к физической памяти. Для ОС пока еще не стало нормой предполагать работу в виртуальной среде. В архитектуре процессоров Intel x86 нет ни возможности, ни потребности помечать страницы физической памяти как неиспользуемые [15]. Поэтому требуются специальные алгоритмы оценки размера рабочего набора – страниц физической памяти, к которым гостевая ОС реально обращается. И по мере

распространения систем виртуализации и облачных сервисов востребованность данной оценки возрастает.

Оценка размера рабочего набора гостевой операционной системы может базироваться как на предыдущих наблюдениях, так и на «гостевых счетчиках» – статистике, собираемой гостевой ОС для целей внутреннего мониторинга. Кроме того, оценка может использовать статистику от монитора виртуальной машины, которая отражает запросы гостевой ОС к привилегированным операциям и внешним устройствам – «виртуализационные счетчики».

Сложность оценки связана с высокой скоростью изменения виртуализационной нагрузки, что сильно снижает точность оценки и ее применимость. При использовании оценки размера рабочего набора для расчета объема фиктивно занятой памяти виртуализационная система (гипервизор, монитор виртуальной машины) должна быть готовой к быстрой коррекции в случае неблагоприятных последствий. То есть система должна не просто использовать оценку, но и адаптировать свое решение в зависимости от наблюдаемого поведения гостевой ОС. *Построение системы, регулирующей объем фиктивно занятой памяти на основании оценки рабочего набора и корректирующей его в зависимости от реакции гостевой операционной системы, поможет сэкономить физическую память для достижения большой плотности размещения виртуальных машин без привнесения существенных накладных расходов, тем самым повысив экономическую эффективность использования технологий облачной виртуализации без снижения уровня обслуживания.*

Цель работы, задачи исследования

Целью данной диссертационной работы является разработка методов и алгоритмов, ограничивающих потребление физической памяти гостевой

операционной системы через управление агентом фиктивного занятия физической памяти (balloon), с последующей реализацией их в виде программного комплекса. Разрабатываемый программный комплекс должен осуществлять:

1) динамическую оценку размера рабочего набора гостевой ОС на основании гостевых счетчиков производительности;
2) корректировку оценки в соответствии с виртуализационной и гостевой статистиками, используя методы обучения с подкреплением;
3) управление агентом фиктивного занятия физической памяти (balloon) в соответствии с полученной оценкой для достижения целевых параметров.

Задачи исследования:

- отбор значимых параметров гостевой и виртуализационной статистики;
- проверка гомогенности отобранных виртуализационных параметров для различных версий операционных систем;
- построение оценки на основании гостевой статистики;
- разработка системы штрафов для корректировки полученной оценки через виртуализационную статистику;
- разработка программного комплекса, управляющего агентом фиктивного занятия физической памяти (balloon) в соответствии с предложенной оценкой;
- настройка и расширение имеющейся автоматизированной системы тестирования для тонкого подбора эмпирических параметров системы.

В ходе выполнения исследований автором был проведен ряд экспериментов, подтверждающих эффективность разработанной системы. Диссертационное исследование выполнялось с использованием облачной инфраструктуры Индустриального партнёра ООО «Акронис» по договору с ООО «Проект ИКС» о выполнении прикладных научных исследований при финансовой поддержке Министерства образования и науки Российской Федерации. Соглашение о предоставлении субсидий № 14.579.21.0010. Уникальный идентификатор Соглашения RFMEFI57914X0010.

Научная новизна

В работе получены следующие основные результаты, обладающие научной новизной:

1. Исследована возможность оценки объема потребляемой памяти гостевой операционной системой на основании исключительно виртуализационной статистики. Полученные результаты свидетельствуют о недостаточности информации, полученной только от систем виртуализации, для построения корректной оценки.

2. Подтверждено эмпирически, что объединение виртуализационной и гостевой статистик дает достаточно информации для адекватного описания поведения гостевой системы.

3. Разработан и реализован в рамках Parallels Server алгоритм адаптивной коррекции оценки размера рабочего набора через штрафы, назначаемые на основании виртуализационной статистики.

4. Подтверждена переносимость алгоритма посредством его реализации в системе с открытым исходным кодом Linux (модуль гостевых расширений KVM).

Теоретическая и практическая значимость исследования

Разработанный программный комплекс успешно интегрирован в систему управления ресурсами в коммерческом программном комплексе Parallels Server, который является широко распространенным среди хостинг-провайдеров программным решением и обеспечивает бесперебойную работу нескольких сот тысяч виртуальных выделенных серверов.

Отдельные наработки также перенесены в виртуализационную систему с открытым исходным кодом (Linux KVM).

Разработанные и предложенные в диссертации методы и программные средства адаптивного управления ресурсом физической памяти виртуальной машины включены в разрабатываемые ООО «Проект ИКС» технологии и программное обеспечение распределенных и высокопроизводительных вычислительных систем для хранения и обработки больших данных, отвечающие за балансировку нагрузки в облаке виртуальных машин

Публикации

По материалам данного диссертационного исследования были опубликованы работы [17-26]. Работы [17, 19-21] выполнены автором единолично. В работе [18] автору принадлежит роль исполнителя. В работах [22, 24, 26] автор выполнял постановку задачи, осуществлял общее руководство, вносил редактуру и дополнения по предметной области. В работах [23, 25] автору принадлежит постановка задачи, общее руководство, редактура модели, консультации по возникающим трудностям при разработке модели, методы верификации модели. В списке изданий на момент соответствующих публикаций присутствуют рекомендованные ВАК ([22, 23, 24]), индексируемые системами WebOfScience и SCOPUS ([21, 25, 26]), и один патент ([18]).

Апробация результатов работы

Результаты диссертационного исследования и сопутствующих результатов докладывались, обсуждались и получили высокие оценки специалистов на российских и международных научных конференциях:

- Гагаринские чтения (Москва, 2010);
- IEEE Sixth International Conference on Cloud Computing CLOUD (USA, Santa Clara, 2013);
- Second International Conference "Cluster Computing" (Ukraine, Lviv, 2013);
- «Облачные вычисления. Образование. Исследования. Разработка 2014» (Москва, 2014);
- IARIA Cloud Computing (France, Nizza, 2015).

Актуальность задачи и ее возможные решения также были обсуждены на популярной технической конференции Highload++ (Москва, 2011).

Результаты работы реализованы в виде компоненты программного комплекса Parallels Cloud Server. Также подготовлен набор изменений для внедрения алгоритма в Linux KVM.

Положения, выносимые на защиту

На защиту выносятся следующие основные положения:

1. Алгоритм управления физической памятью виртуальной машины на основании оценки размера рабочего набора гостевой операционной системы, корректируемой через штрафы, которые назначаются на основании виртуализационной статистики;

2. Обоснование эффективности алгоритма управления как имеющего приемлемые накладные расходы, дающего высокий выигрыш по потреблению памяти и не приводящего к снижению производительности гостевой ОС при быстро меняющихся нагрузках;

3. Реализация алгоритма в виде комплекса программ системы управления памятью как часть программного комплекса Parallels Server.

Личный вклад автора

Автором проведено исследование предметной области, выполнен анализ гостевых и виртуализационных счетчиков, разработана оценка размера рабочего набора на основании виртуализационных счетчиков, сформулирован подход по корректировке размера фиктивно занятой памяти на основании штрафов от виртуализационной статистики, написан алгоритм корректировки и проведено экспериментальное исследование эффективности его работы.

Разработка оценки на основании гостевых счетчиков и проверка гомогенности виртуализационных параметров проводилась совместно с Маркеевой Л.Б.

Структура и объем диссертации

Диссертация состоит из введения, пяти глав, заключения и двух приложений. Работа изложена на 105 страницах. Список использованных источников содержит ссылки на 110 публикаций.

Глава 1. Обзор методов управления памятью в виртуальных машинах

1.1. Постановка задачи управления памятью

Задача управления памятью в виртуальной машине включает нескольких подзадач:

1. Оценка размера рабочего набора (то есть число страниц, к которым гостевая ОС обращалась в недавнем прошлом);
2. Виртуальная подкачка:
    a. выбор страницы памяти, которую уберут из рабочего набора, а ее содержимое сохранят в backing store – файле подкачки;
    b. механизм сброса выбранной страницы в файл подкачки;
    c. подгрузка страниц из файла подкачки (по необходимости или с предвыборкой);
3. Сокращение общего объема потребляемого ресурса.

При решении задачи управления важно задать критерий эффективности. Когда мы говорим об эффективности алгоритмов управления памятью, то наиболее значимым показателем оказывается соотношение между объемом сэкономленного ресурса и снижением производительности гостевой ОС вкупе с возникшими издержками – то есть вычислительными ресурсами и ресурсом оперативной памяти, необходимыми для исполнения алгоритма. Так, алгоритмы компрессии и дедупликации, решающие задачу 3 (сокращение общего объема потребляемой памяти), стабильно вносят значительные издержки, в то время как объем сэкономленного ими ресурса переменен и зависит от конкретного набора данных, от задач, исполняемых в гостевой ОС. Поэтому не все системы виртуализации реализуют эти подходы.

1.2. Обзор алгоритмов управления памятью

Самым популярным решением, встречающимся во всех популярных виртуальных машинах, является алгоритм фиктивного занятия физической памяти, именующийся «виртуальным баллоном» (ballooning). (рис.2).

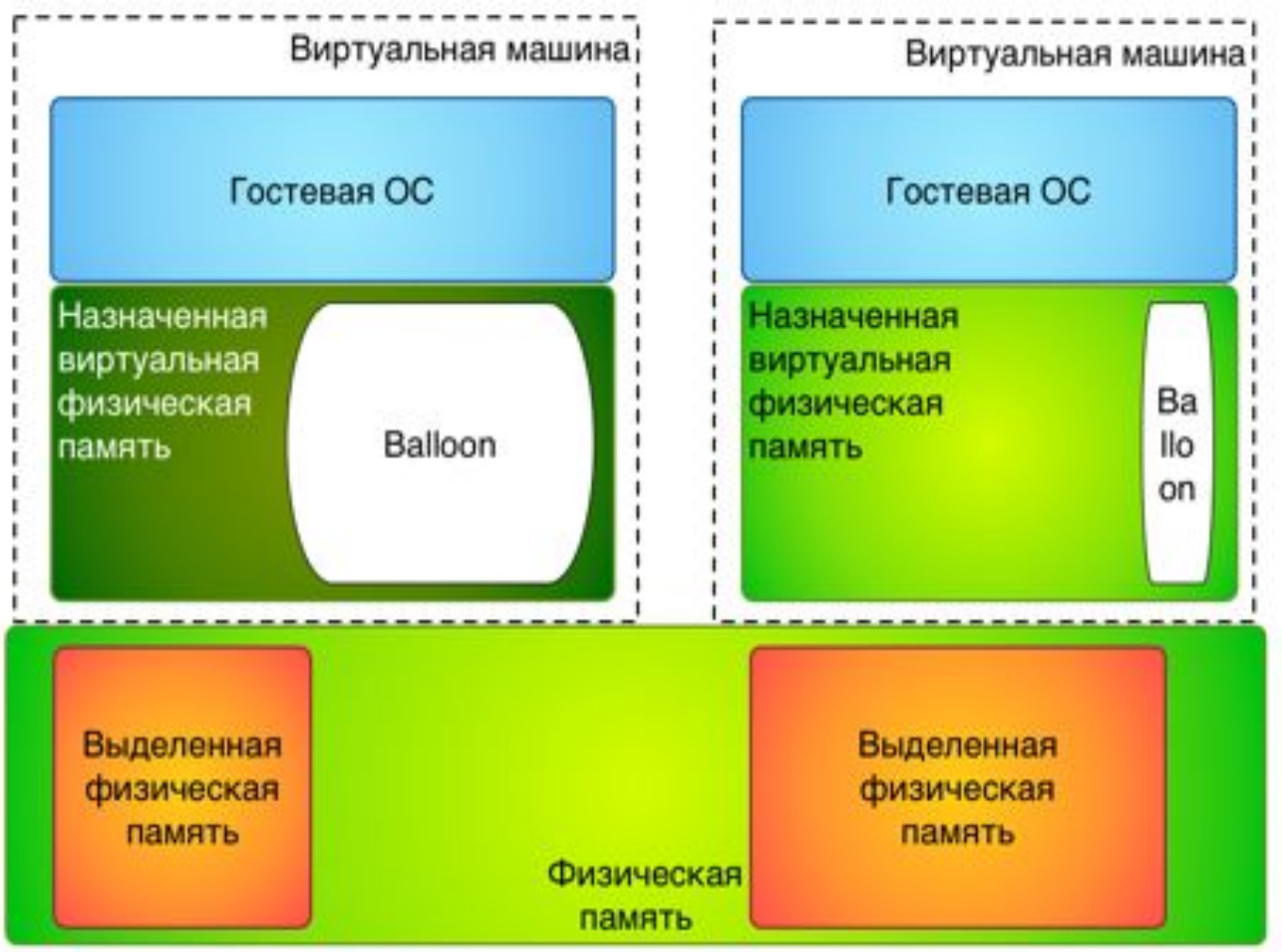


Рис. 2. Фиктивное занятие памяти

Алгоритм реализует подзадачу виртуальной подкачки (задача 2.а) – он выбирает страницы памяти для вытеснения из текущего набора страниц физической памяти [27,19,20]. Несомненное достоинство алгоритма в гарантированном отсутствии обращений гостевой ОС к вытесненным через него страницам. Это чрезвычайно эффективная техника, которая не привносит больших издержек, до определенного момента почти не снижает

производительность гостевой ОС и может «сэкономить» половину объема назначенной памяти. Ballooning реализован во всех современных гипервизорах [27,28,29,30,31].

Термин ballooning (дословно «раздувание», «создание воздушного шара») ведет свое происхождение от метафоры воздушного шарика, которая очень хорошо описывает происходящее. В гостевой ОС работает специальный агент (balloon), который выделяет память средствами гостевой операционной системы и резервирует ее, тем самым блокируя возможное обращение к ней других программ. Полученную таким образом память агент отдает виртуализационному модулю (ВММ). С точки зрения гостевой операционной системы и программ, работающих в ней, страницы заняты в работе. С точки зрения фактического использования ресурса страницы свободны и пусты: то есть в виртуальной физической памяти возникает пустотность – воздушный шар (рис.2).

Недостатком техники является потенциально возникающая гостевая подкачка: чем больше страниц высвобождается через фиктивное занятие физической памяти (через ballooning), тем больше увеличивается нагрузка (давление) на остальные страницы системы. Тем не менее, корректная оценка максимального возможного объема фиктивно занятой памяти решила бы эту проблему.

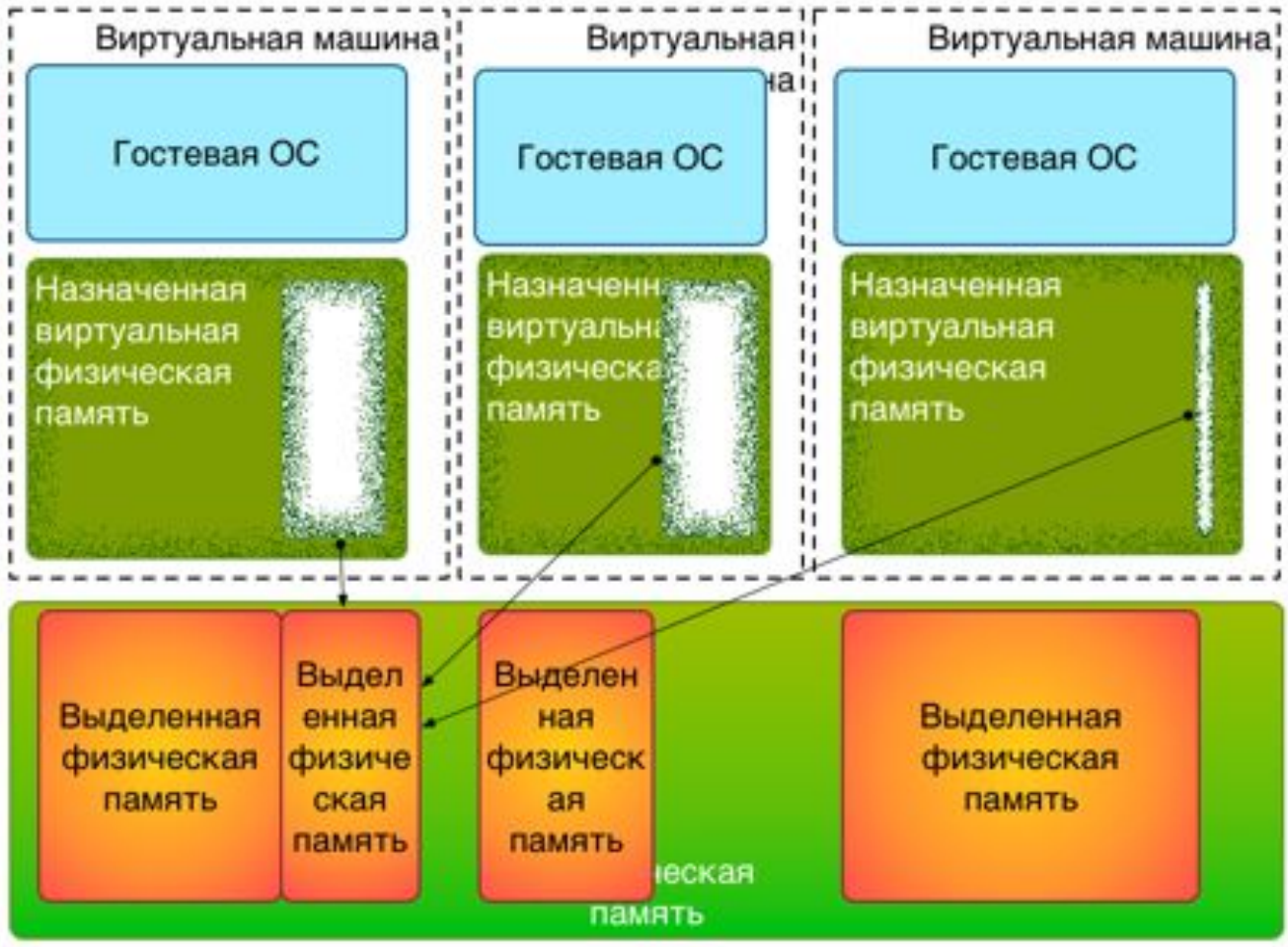


Рис. 3. Объединение одинаковых страниц

Следующим по популярности решение является объединение одинаковых страниц, или дедупликация (content-based page sharing). При наличии одинаковых страниц в памяти виртуальных машин, выгодно хранить только одну защищенную от записи копию и создавать отдельные дубликаты при попытке записи (COW – copy on write). Объединение страниц существенно снижает общее потребление памяти в системе (задача 3) при большом количестве потенциальных дублеров, но всегда привносит дополнительные расходы на расчет хэш-сумм, по которым определяются дубликаты (рис.3). Поскольку заранее предсказать процент дублей невозможно, технику стоит применять с осторожностью. Техника применяется в VMware [27,32], в Xen [28] и VirtualBox [30]. Однако, своего

триумфа дедупликация достигла в KVM, когда вошла в состав ядра Linux под именем KSM [33] – kernel same page merging. KSM оказывается относительно дешевым с точки зрения накладных расходов из-за интеграции с системным механизмом выделения.

Интересной техникой является сжатие памяти (или компрессия). Цель подхода – сократить накладные расходы на виртуальную подкачку и ускорить приостанов/миграцию виртуальной машины (задача 3). Без дополнительной компрессии штраф за неправильно вытесненную страницу высок: не менее двух дисковых операций (по работе с резервным хранилищем) и некоторое количество системных вызовов. При компрессии содержимое вытесненной страницы некоторое время хранится в виртуальной памяти процесса в сжатом виде прежде чем быть вытесненным на диск. Кэш компрессии ускоряет миграцию и приостановку (suspend) виртуальной машины. Техника используется у VMware [27] и Parallels.

Традиционным решением задачи виртуальной подкачки (задача 2) являются алгоритмы вытеснения. Этот аппарат досконально проработан с начала 60-х годов прошлого века, когда задача вытеснения впервые стала в полный рост, и продолжает успешно дополняться все более усовершенствованными алгоритмами и по сей день. Самым классическим алгоритмом вытеснения является LRU – «наиболее давно использовавшийся» - least recently used [34]. Алгоритм вытесняет страницу, к которой дольше всего не было обращений. Традиционно алгоритм реализован на структуре данных типа двусвязный список, в котором при обращении элемент «всплывает». Со времен начала своего активного внедрения в 60х годах [35] алгоритм был серьезно усовершенствован: примером тому LRU 2-Q[36] и LRU-k [37]. Другим распространенным алгоритмом является устаревание (aging), потомок алгоритма Not Frequently Used (не часто используемый) [38,39]. Основная идея этого семейства алгоритмов в том, чтобы отмечать обращение к странице поинтервально. В отличие от LRU, NFU/aging способны спасти от

вытеснения страницы, к которым не было обращений в самое последнее время, но которые активно использовались ранее. Наиболее применяемым методом в современных операционных системах является метод часов (clock) [38,39] и его модификации GLOCK [40], CLOCK-Pro [41], WSClock [42,43], CAR [44]. Можно сказать, что алгоритм часов реализует упрощенное NFU: отслеживается обращение к странице в течение последнего временного интервала. Однако, в отличие от NFU часы не подразумевают специальных служебных структур и внедрения механизма треккинга обращений. Часы отслеживают обращения к страницам на уровне страничных преобразований. Невысокие накладные расходы этого семейства алгоритмов обуславливают их широкую применимость. Алгоритмов вытеснения разработано большое количество и их перечисление не входит в задачу данной работы, но отмечу, что алгоритм выбора произвольной страницы (random) имеет самые низкие издержки на фоне приемлемой точности, а большая часть виртуализационных решений защищается от дорогих ошибок через использование «второго шанса» (second-chance) [38,39]: вытесненная страница не удаляется сразу, а попадает промежуточный буфер, откуда будет удалена в свой срок.

Описанные выше алгоритмы хорошо зарекомендовали себя в операционных системах, однако в виртуализации они могут применяться лишь в качестве запасного пути из-за проблем так называемого «семантического зазора» [45]: возможного пересечения работы алгоритмов вытеснения гостевой ОС и виртуализационного ПО. Тем не менее, в отсутствии лучшей альтернативы (при исчерпании запаса фиктивно занятой памяти, при отсутствии дупликатов и при невозможности мигрировать) алгоритмы будут запускаться и в VMware [27], и в Parallels, и в Xen hypervisor [28], и во многих других.

Продуктивным способом для управления памятью может стать паравиртуализация – модификация гостевой ОС с целью улучшения ее работы в виртуальной машине. Действительно, если операционная система «знает» о том,

что она исполняется в виртуальном окружении, то она может гарантировать отсутствие обращений и без дополнительного агента фиктивно занятой памяти (balloon), а также предоставить запрашиваемое количество страниц с минимальным повышение уровня гостевой подкачки. Но пока современные ОС предпочитают скорее предоставлять необходимые программные интерфейсы для balloon, нежели включать виртуализационную функциональность в себя. Это связано со сложностью систем управления памятью в операционных системах и желанием максимально разнести эти задачи для снижения уровня комплексности. Однако, есть немало академических разработок на Xen, которые активно модифицируют гостевое ядро. Ввиду указанных выше сложностей, эти наработки едва ли могут выйти из зоны исследований и перейти в зону внедрения.

Для того, чтобы определить размер рабочего набора (задача 1), столь нужный для корректной работы алгоритма «виртуального баллона» и реализации общего управления памятью в облачной инфраструктуре, может применяться отслеживание обращений к страницам. В самом буквальном виде такая технология дает столь существенные затраты по производительности, что становится неприменимой. Традиционной оптимизацией является сканирование по принципу Access/Dirty(Reference)-бита: если к странице не было доступа за указанный интервал времени, то она помечается неиспользуемой, иначе бит активности сбрасывается до следующей проверки [46,47]. Такой вариант работает не на всех платформах: так, технология аппаратной виртуализации страничных преобразований от компании Intel – EPT - до недавнего времени не поддерживала эти биты [48]. Адаптированный вариант предложен в [49]. Весь набор страниц разделяется на холодные и горячие. Доступ к горячим не перехватывается, доступ к холодным предполагает так называемый миноритарный страничный промах (minor page fault), по которому страница перемещается в список горячих. Холодные страницы отсутствуют в гостевом страничном преобразовании, но присутствуют в рабочем наборе виртуальной машины. То есть миноритарный

страничный промах не требует работы с файлом подкачки. Присутствие страницы в холодном списке в течение заданного интервала времени интерпретируется как устаревание страницы. Тем не менее, алгоритм все еще вносит существенные издержки. Работа Zhao et al 2011-го года [50] сокращает накладные расходы до 13% через выделение разных стадий работы программ, но события, на основании которых они это делают (L1 cache miss, TLB miss и пр.) слишком дороги для мониторинга, а прогноз недостаточно точен. Предложенный в 2007 году эксклюзивный кэш гипервизора [51] предлагает трассирование с меньшими накладными расходами, но сам поход настолько сложен, что едва ли применим вне академической среды. MEB [52] перехватывает обращения к гостевым страницам через управление привилегиями на уровне таблиц страниц и строит LRU (least recently used) гистограмму, чтобы оценить размер рабочего набора для каждой виртуальной машины и на основании его сбалансировать систему. Но накладные расходы такого подхода неприемлемы.

Достаточно продуктивный, хотя и не столь академичный подход, используется для определения размера рабочего набора в VMware [27]. Произвольно выбирается 10% присутствующих страниц для удаления из гостевого страничного преобразование. Рассчитанное число миноритарных страничных промахов распространяется на весь рабочий набор.

1.3. Обзор подходов к предсказанию уровня потребления ресурса в гипервизорах

Задачей, смежной с определением объема потребляемого ресурса, является задача предсказания объема потребляемого ресурса. Оценка потребления ресурса является тривиальной задачей для процессора, сетевой и дисковой нагрузки, только оценка нагрузки на память требует дополнительных исследований, как

было показано ранее. В то же время задача предсказания уровня потребления ресурса актуальна для всех типов ресурса.

Интересная серия работ проведена Steven Hand и Evangelia Kalyvianaki [53, 54, 55]. В них представлено предсказание потребление CPU на основе фильтров Калмана. Предсказание показывает хорошие результаты и в случае многоуровневой задачи (multi-tier application). Предсказание перегрузки системы целиком предпринято в работе [56]. Авторы производят спектральный анализ нагрузки используя вейвлет. В работе [57] предсказание уровня потребления CPU тесно связано с коррекцией вольтажа для снижения энергозатрат. Предсказание выполняется на основании гибридной техники: сначала с помощью преобразований Фурье авторы пробуют выделить «сигнатуру» нагрузки. Если сигнатура нашлась, то для нее имеется прогностическая модель. Если не нашлась, то предсказание будет базироваться на Марковских цепях. Метод обрабатывает ошибку предсказания и использует ее для более тонкой настройки. Сильная сторона работы – предсказание конфликтов приложений за ресурсы до возникновения конфликта и последующая превентивная миграция.

В работе [58] проводится сравнение различных методов машинного обучения – авторегрессии, авторегрессии комбинированной с ANOVA – и адаптивной моделью на базе многоимпульсного контроллера. Авторы предсказывают потребление CPU на основании слепков, полученных с 36 реальных серверов. В статье делается вывод о лучшем обучении и лучшем результате адаптивной модели по сравнению с прогностической.

Интересная попытка совместить обучение с подкреплением с теорией массового обслуживания предпринята в работе [59]. Здесь обучение с подкреплением используется для построения предварительной модели, а контроллер, работающий по принципу теории массового обслуживания, управляет поведением системы. Объект изучение в статье – балансировка запросов по серверам.

Моделирование соотношения между производительностью приложений, исполняющихся в виртуальном окружении, и количеством ресурсов, выделенных виртуальной машине – это задача, поставленная в работе [60]. Авторы используют SVM и нейронные сети и демонстрируют точность предсказания на уровне 95% (в среднем).

Нечеткая логика используется для распределения ресурсов в работе [61]. Ее носителями являются локальный и глобальный контроллеры, которые работают на уровне виртуального контейнера и всего кластера соответственно. Метриками модели являются усредненные показатели уровня потребления ресурсов, производительности приложения в виртуальном окружении и некий уровень нагрузки.

Еще одним подходом к решению поставленной задачи является применение теории управления чтобы корректировать объем выделенного ресурса для виртуальной машины и таким образом достигать заданного уровня производительности приложения. Такие решения часто предполагают производительность приложения как линейную модель. Например, линейная мульти-вводная-мульти-выводная модель (multi-input-multi-output model – MIMO) используется для управления разнородными ресурсами для многоуровневых (multi-tier) приложений [62]. Аналогичная модель (MIMO) используется для выделения вычислительного ресурса для компенсации интерференции между конкурирующими виртуальными машинами [63]. Такие линейные модели не могут аккуратно предсказать нелинейное поведение.

Машинное обучение широко используется для предсказания уровня потребления ресурсов, как раскрыто в работе [61]. В дополнении к методам, описанным ранее, авторы приводят значительный список работ по теме. К ним относится проект CARVE, где с помощью линейной регрессии восстанавливается зависимость производительности (а точнее пенальти по нарушению гарантий на уровень обслуживания) от объема выделенной памяти [64]. Тем не менее, работа [65]

показала, что, хотя линейная регрессия хорошо подходит для моделирования простых приложений, для более сложных приложений в виртуальных средах в результате образуется большая ошибка. В этой же работе рекомендуются нейронные сети для решения вопросов моделирования виртуализованных приложений.

Для предсказания перегрузок машин, так называемых кризисов, одна из работ предлагает использовать логистическую регрессию с L1 регуляризацией, визуализируя полученные индикаторы как ROC-кривые [66]. Работа приводит подтверждения эффективности предложенной техники, однако никак не пытается предотвратить перегрузку. Проект VCONF [67] напротив активно «вмешивается» в происходящее, пытаясь переконфигурировать виртуальные машины для достижения лучшей производительности виртуализованных приложений. Потребность приложении в ресурсах VCONF предсказывает на основании нейронной сети, объединенной с обучением с подкреплением. И, хотя сам метод воздействия на систему нам не подходит, математические методы работ в чем-то схожи.

1.4. Управление уровнем фиктивно занятой памяти.

При управлении размером фиктивно занятой памяти нужно учитывать два фактора:

1. При работе алгоритма фиктивного занятия памяти, гостевая ОС может испытывать падение производительности в связи с слишком большим объемом занятого ресурса. В связи со спецификой программного интерфейса, через который действует агент фиктивного занятия памяти, гостевая ОС будет продолжать выделять агенту память “до последнего”, когда все кэши уже давно закончатся.

2. Колебание объема фиктивно занятой памяти приводит к большим накладным расходам со стороны всей виртуальной машины, поэтому колебаний стоит избегать.

Одним из наиболее очевидных решений является увеличение фиктивно занятой памяти до размера равного объему назначенной памяти за вычетом рабочего набора. Проблема оценки размера рабочего набора хорошо проработана в алгоритмическом смысле [46,39,68]. Однако, виртуализация вносит новое измерение. Оценка размера рабочего набора гостевой операционной системы может базироваться как на предыдущих наблюдениях, так и на гостевых показателях – статистики, собираемой гостевой ОС для целей внутреннего мониторинга. Кроме того, оценка может использовать статистику от монитора виртуальной машины, которая отражает запросы гостевой ОС к привилегированным операциям и внешним устройствам – виртуализационные счетчики.

Однако, сложность состоит в синтетичности самого понятия рабочего набора Рабочий набор – это подмножество страниц физической памяти, к которым было обращение в течение некоего времени [46,39,68]. Предполагается, что в силу локальности обращений текущий рабочий набор является хорошим предсказателем фактического использования памяти в следующем временном интервале [46]. Но это не совсем верно. Так, например, память сервисного процесса, который ставится на исполнение раз в 10 минут, окажется вне рабочего набора, если он оценивается по последней минуте. Запуск нового процесса изменит рабочий набор, добавив в него новые страницы. При этом и сама гостевая система может дополнительно применять алгоритмы для улучшения своей производительности и времени отклика. Так, при активном потребителе система может предвосхищать будущие запросы на выделение памяти, а при прекращении нагрузки система может не сразу передавать высвободившийся ресурс в пул

свободных страниц. Все вышеперечисленное делает любую оценку размера рабочего набора неточной.

Рассмотрим существующие алгоритмы для управления фиктивно занятой памятью (ballooning).

В проекте qemu-kvm управление осуществляется гипервизором без учета состояния гостевой ОС. Единственная автоматизация – это уменьшение размера в случае событий нехватки памяти (Out Of Memory = OOM) с помощью патча, разработанного под моим руководством Раушанией Максудовой [69]. Существует проект Autoballoon, где увеличение фиктивно занимаемой памяти происходит при детектировании нехватки памяти в хостовой ОС, а снижение уровня осуществляется при возрастании давления в госте [70,71]. Однако, изменения, разработанные внутри проекта, так и не были внесены в основное дерево исходных кодов, вероятно из-за запланированных работ по архитектурному изменению агента фиктивно занятой памяти (balloon) [72].

Размер фиктивно занятой памяти у VMware выставляется на основании нагрузки на память в рамках всей физической машины, гостевая нагрузка не учитывается [27,32]. Есть основания предполагать наличие параметра, отвечающего за максимальный процент фиктивно занятой памяти [73]. Такой подход при некоторых нагрузках может приводить к падению гостевых приложений и гостевой ОС, что и доказывает соответствующая статья от VMware датированная 2014 годом [74].

В Xen решение по автоматическому управлению размером фиктивно занятой памяти было реализовано в 2008 году [75,28]. Фиктивно занимается размер равный (total – CommittedAS), где CommittedAS – это гостевой счетчик, предположительно равный размеру рабочего набора, и обрабатываются ситуации нехватки памяти в системе (OOM = out of memory). Это простое и эффективное решение подвергается критике из-за медленной адаптации и колебаний размера.

Остроумное решение представлено на базе гипервизора Xen в 2013 году [76]. Алгоритм, получивший название TWS (true working set), полагается на конечный автомат с тремя состояниями («быстрое», «медленное», «охлаждение»). Конечный автомат стартует в «быстром» состоянии и активно передает страницы агенту фиктивно занятой памяти (balloon). По мере увеличения показателей гостевой подкачки и страничных промахов, алгоритм переходит в состояние «охлаждения», в котором страниц не забирает, а предположительный размер рабочего набора увеличивается. Переждав какой-то период времени в состоянии «охлаждения», алгоритм переключается в «медленное» состояние, в котором также фиктивно занимает страницы, но со скоростью меньшей, чем в «быстром». В случае возникновения гостевой подкачки, алгоритм опять переключается в состояние «охлаждения» (рис.4). Недостатком алгоритма является модификация гостевого ядра и некорректная работа алгоритма на слишком маленьких и слишком больших размерах гостевого набора. Кроме того, как показали исследования, выполненные Раушанией Максудовой в рамках ее дипломной работы, потери производительности алгоритма были существенными.

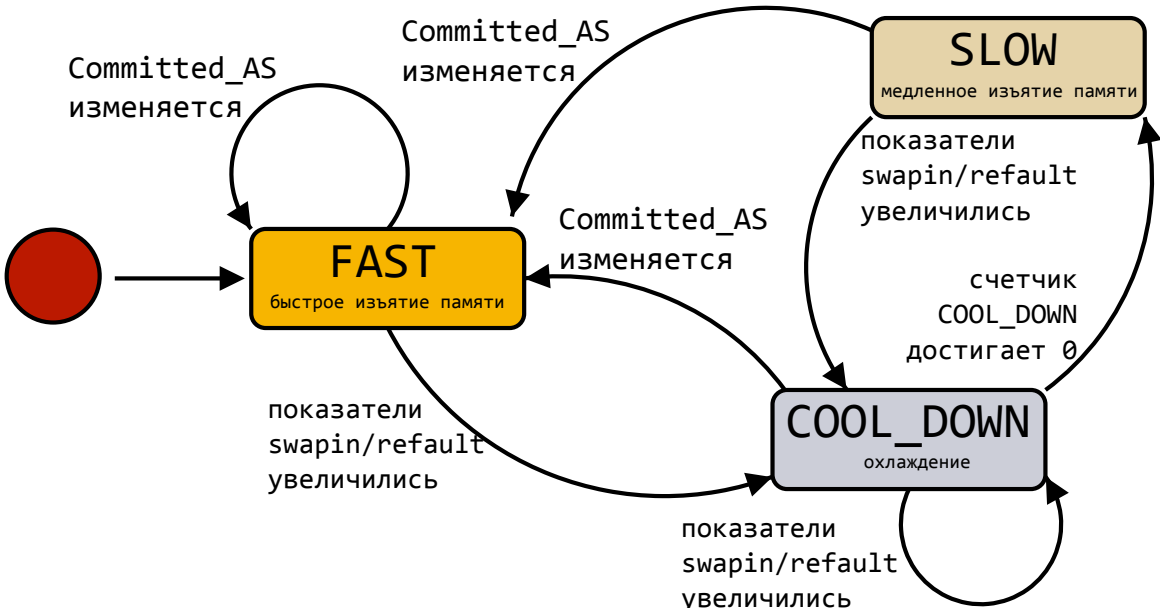


Рис. 4. Конечный автомат алгоритма TWS

Одна из последних работ на тему фиктивно занятой памяти в Xen датируется сентябрем 2015 [77] и представляет собой вариацию на тему алгоритмов VMware: основанием для забора ресурса у гостевой ОС является нехватка ресурса в рамках хостовой ОС. Авторы разделяют разные уровни давления на ресурс на физической машине и, в зависимости от конкретного уровня, определяют агрессивность гостевого агента.

Технология Hyper-V от Microsoft, похоже, содержит что-то похожее на auto-balloon, так как размер фиктивно занятой памяти меняется в зависимости от гостевой нагрузки [78]. Информации по технологии немного, но вероятно, состояние хостовой ОС также принимается во внимание [79].

Все имеющиеся решения обладают двумя недостатками:

1. Они не учитывают инертность интерфейса выделения памяти гостевой ОС и как следствие этого инертность самого агента фиктивного занятия физической памяти.
2. Они не застрахованы от колебаний размера фиктивно занятой памяти и тем самым могут вносить дополнительные накладные расходы.

Задача данной работы состоит в том, чтобы разработать такой эффективный алгоритм, который не вносил бы накладных расходов и потому мог работать постоянно, а не только в режиме переподписки. Для этого система должна не просто использовать относительно точную оценку размера рабочего набора, но и адаптировать свое решение в зависимости от наблюдаемого поведения гостевой ОС. *Построение системы, регулирующей объем фиктивно занятой памяти на основании оценки рабочего набора и корректирующей его в зависимости от реакции гостевой операционной системы, поможет сэкономить физическую память для достижения большой плотности размещения виртуальных машин без привнесения существенных накладных расходов, тем самым повысив экономическую эффективность использования технологий облачной виртуализации без снижения уровня обслуживания.* Кроме того, алгоритм должен быть достаточно простым, чтобы его было возможно реализовать на разных гостевых и виртуализационных системах.

ГЛАВА 2. Доступные параметры виртуальной машины

2.1. Виртуализационные счетчики

Системы виртуализации построены на цикле: запуск гостевой ОС в непривилегированном режиме – виртуализационное событие – обработка виртуализационного события – повторный запуск гостевой ОС. Виртуализационные события – это привилегированные инструкции, исполняемые гостевой ОС. К ним относятся различные прерывания, исключения, обращение к памяти, находящейся в прямом доступе устройств (в том числе и контроллер прерываний и различные таймера), доступ к страницам с отсутствующим преобразованием гостевого физического адреса в реальный физический адрес, доступ к MSR, контрольным регистрам и другое. Большая часть из вышеназванных событий связана либо с исполнением ядра операционной системы, либо с работой гостевых устройств. Наш опыт показывает, что последовательность событий или даже отдельные события однозначно определяют сценарий, разворачивающийся в госте.

Например, каждый процесс в современной операционной системе имеет отдельное адресное пространство и отдельное дерево страничных преобразований (paging tree). В архитектуре Intel x86 [16] смена дерева осуществляется за счет переключения контрольного регистра cr3, где хранится корневой элемент. Запись в cr3 вызывает виртуализационное событие. Таким образом, ВММ может подсчитывать число различных cr3 и частоту их смены, что позволит делать выводы о количестве процессов в гостевой ОС и даже выстроить предположение о качестве нагрузки в системе. Так, если частота переключений между приложениями превышает обычную для данного гостя, то вероятна нагрузка либо на жесткий диск, либо на физическую память: частые переключения могут

свидетельствовать об ожидании приложением ресурса, а частые переключения могут свидетельствовать о дефиците общего ресурса.

Другим примером, является запись в IA32_FSBASE_MSR, которая обозначает количество переключений между различными уровнями привилегий (user-space, kernel-space) в 64-разрядной Windows.

Разные сценарии гостевой ОС соотносятся с различными ожиданиями на объем потребляемой памяти. Возвращаясь к примеру с переключением cr3, постановка нового процесса на исполнение вызовет доступ к некоторым его страницам, по крайней мере к тем, что содержат дескриптор процесса и исполняемый код, а также некоторый объем данных. Но размер дескриптора и предвыборка данных и кода зависит от типа операционной системы.

Стоит сделать отдельный акцент на упомянутое раннее виртуализационное событие по доступу к страницам с отсутствующим преобразованием гостевого физического адрес (GPA, guest physical address) в реальный физический адрес (HPA, host physical address) – виртуальный страничный промах. Событие наступает в том случае, когда страница отсутствует в фактически предоставленной памяти виртуальной машине, но присутствует в гостевых преобразованиях, что происходит либо на начальных стадиях, либо в случае вытеснения страницы монитором виртуальной машины. Реакцией на запрос будет подкачка страницы гостевой физической памяти из файла подкачки, которое мы обозначим термином виртуальная подкачка. Объем виртуальной подкачки, а точнее количество виртуальных страничных промахов есть обратная связь на неправильно вытесненные страницы.

2.2. Гостевые счетчики

На первый взгляд гостевые счетчики – параметры, собираемые гостевой ОС для внутреннего мониторинга, - оказываются лучшим вариантом. Ведь каждая система так или иначе информирует пользователя о количестве свободной памяти. Но у большинства систем подобные счетчики недостаточно информативны. Так, например, на ОС Linux объем свободной памяти почти всегда равен нулю [80]. Часто для приблизительной оценки используют размеры буферов, CommittedAs, сумма по процессам [81], но все методы имеет свои неточности. В операционной системе Windows к страницам внутри пула свободных страниц может быть заказана операция на заполнение страницы нулями (Zero Page Thread) [82], что с точки зрения виртуализационной системы есть возобновление использования страницы памяти. Кроме того, во многих системах пул свободных страниц наполняется по требованию – то есть, если какой-то процесс использовал память, а потом прекратил её использование, его страница остается прикрепленной к нему до тех пор, пока другому процессу не потребуется память [82]. А если учесть разнообразные кэши операционных систем, алгоритмы дедупликации и компрессии, то становится ясным, что однозначного понимания о размере рабочего набора через гостевые счетчики добиться не получится.

Сам по себе сбор гостевых счетчиков достаточно дорогой. Он осуществляется в несколько этапов:

1. В модуле, работающем на стороне гостевой ОС раз в условленный интервал времени, через программный интерфейс гостевой ОС извлекаются интересующие нас значения. В случае ОС Linux счетчики доступны через виртуальную файловую систему /proc [83][84]. Для Windows программный интерфейс более сложный и базируется на работе с реестром [85]. Хотя часть информации доступна через более простые функции, как то ZwQueryInformationProcess [86], ZwQuerySystemInformation [87].

2. Информация из гостевого модуля передается в монитор виртуальной машины. Если сбор счетчиков реализовать внутри самого агента фиктивного занятия памяти (balloon), то передавать информацию можно было бы через расширение паравиртуализационного интерфейса VirtIO. Если же сбор счетчиков будет в другом модуле, то нужно искать средства для передачи собранных данных, которые могут отличаться для разных систем виртуализации. Однако, стоит отметить что мало из систем позволяют передавать данные в монитор виртуальной машины для задач, исполняющихся на user space.
3. Монитор виртуальной машины передает собранные счетчики в гипервизор или хостовую ОС для записи в файл для последующего анализа.
4. Хостовая ОС или гипервизор записывают собранные данные в CSV формат для последующей обработки.

2.3. Сравнение счетчиков

Спектр событий, видимый через призму этих двух статистик, существенно различается. Так в отношении памяти гостевые счетчики более информативны, когда речь идет о суммарном объеме виртуальной памяти процессов, о размере закрепленной памяти (locked memory - памяти, принадлежащей одному процессу и не подлежащей перераспределению), но не всегда адекватно говорят о количестве страничных промахов (page fault). В то же время количество страничных промахов свидетельствует о скорости изменения рабочих наборов процессов гостевой ОС, о возможной подгрузке страниц из страничного файла (свопинге). Иными словами, количество страничных промахов является важным индикатором состояния системы. И его чрезвычайно легко отслеживать в мониторе виртуальной машины при незначительных изменениях кода и практически при нулевых накладных расходах. Кроме того, некоторые

информативно важные показатели в принципе прозрачны для гостевой ОС. К таковым относится, например, число виртуализационных страничных промахов – показатель, описывающий как часто обращение к гостевой странице порождает запрос к гипервизору на выделение новой страницы памяти (запрос о предоставлении ресурса). Однако, количество счетчиков гостевой операционной системы существенно превосходит количество виртуализационных счетчиков. В то же время накладные расходы от считывания этих показателей значительно превышают издержки на сбор виртуализационной статистики. И, конечно, гостевые счетчики имеют существенную вариативность в зависимости от семейства операционных систем (Windows, Linux, OS/2, OSX) и конкретных версий системы (Windows 2008 и Windows 10). В таблице ниже представлены основные характеристики сравнения гостевой и виртуализационной статистик

Характеристика	Гостевые счетчики	Виртуализационные счетчики
Количество счетчиков	10^3-10^4	50-200
Издержки на сбор статистики	Много доступов к памяти, передача информации из гостевой ОС в управляющий виртуализационный модуль	Доступ к памяти + несколько тактов
Стабильность счетчиков	Меняются от версии к версии гостевой ОС	Количество счетчиков постоянно сокращается, новые не появляются
Сложность поддержки кода	Код специфичен для каждой гостевой ОС и даже для разных версий одной ОС	Код простой, не зависит от гостевой ОС
Информативность счетчиков	Детально отражает информацию о гостевой ОС, но не о состоянии монитора виртуальной машины	Косвенно отражает состояние гостевой ОС (cr3 == количество процессов),

		детально описывает виртуализацию

Таблица 1. Сравнение счетчиков

Так как задача данной работы носит не только академический, но и прикладной характер, то технические факторы, как то сложность поддержки кода, стабильность версий и накладные расходы на сбор счетчиков, оказывают существенное влияние на выбор, склоняя чашу весов к виртуализационным счетчикам. Единственным основанием для отказа в пользу гостевых счетчиков может быть недостаточная информацитивность виртуализационной статистики. Это будет предметом нашего исследования в главе 3.

Глава 3. Оценка размера рабочего набора на основе виртуализационных счетчиков

3.1. Выбор виртуализационных счетчиков

Выбор виртуализационных счетчиков производился на основе как математической, так и экспертной оценки. Математически мы убрали все счетчики с постоянным значением (в том числе нулевым). Экспертиза же выбрала из оставшегося набора наиболее значимые.

Особенное внимание уделялось выбору зависимой переменной, которая должна описывать фактическое использованию гостевою ОС на физическую память. С этой целью в код Parallels Server была добавлена отдельная переменная, которая содержала число страниц с Access и Dirty битами (то есть страниц, к которым был доступ на чтение или на запись). Каждый временной интервал все Access и Dirty биты сбрасывались и в конце интервала считалось число страниц с выставленными битами. Продолжительность интервала была выбрана 10 секунд. Каждый интервал давал одну выборку.

Основным допущением регрессии является независимость показателей (прогностических параметров, предикторов) [88]. Таким образом, ключом к успешной модели является правильный подбор наблюдаемых параметров. Избыток параметров приведет к зависимости, недостаток параметров увеличивает ошибку посчитанной модели.

Для модели после долгого анализа и неудачных моделей были выбраны следующие виртуализационные событиях [21]:

- PgIn - операции подкачки (количество страничных промахов (#PF), произошедших при присутствии страницы в гостевом страничном преобразовании и отсутствии в виртуализованном)

- PgOut - операции вытеснения (количество страниц, вытесненных из рабочего набора виртуальной машины)
- Cr3Cnt - число CR3 (количество различных CR3 – control register #3 - этот параметр обычно коррелирует с числом активных процессов)
- Cr3Chng - Количество переключений CR3 (соотносится с частотой переключения между процессами; слишком большая частота может свидетельствовать о нехватке каких-то ресурсов в гостевой ОС)
- PfCnt - Количество гостевых страничных промахов (#PF)
- TprCnt - Количество обращений к TPR (TPR – регистр приоритета задачи, соотносится с изменением IRQL в гостевой ОС)
- IrqCnt - Количество переданных прерываний (количество гостевых прерываний; коррелирует с гостевой активностью ввода/вывода и, таким образом, отражает работу с диском, сетью и другими устройствами ввода/вывода)
- GuestTime - Время, проведенное в госте
- HltCnt - Количество инструкций останова (hlt)
- HltTime - Время, проведенное в режиме простоя по результатам инструкций hlt (коррелирует с idle режимом в госте)
- MsrAccess - Количество доступов к MSR (MSR – регистры, специфичные для отдельных моделей процессора)
- VirtEvent – количество виртуализационных событий.
- SumTime – суммарная продолжительность интервала (GuestTime + HltTime + VmmTime + HostTime)

Такой значимый параметр как общее количество операций по доступу к APIC был исключен из наблюдений по причине большой корреляции с TPR и прерываниями.

Набор виртуализационных событий, включенных в исследование, будет отличаться для других конфигураций виртуальных машин. Так, например, многопроцессорная система потребует информации о количестве IPI (интер-процессорные прерывания), о количестве сбросов TLB кэша (кэш преобразований из линейного в физический адрес). Система, работающая в 64-битном режиме, много работает с MSR (GsBase, FsBase, KernelGsBase) и работает с TPR через его отображение в CR8. Исследование паттернов для каждого типа ОС, таким образом, требует применения экспертного знания.

3.2. Сбор статистики

Исследование начинается с подготовки обучающей выборки. Для того, чтобы ее собрать, виртуальная машина исполняет набор микротестов:

- простой/занятый цикл (busy loop);
- микро-тесты с дисковой активностью;
- микро-тесты на переключение процессов/потоков;
- микро-тесты с сетевой активностью;
- микро-тесты-потребители памяти;
- инсталляция MS Office;
- симуляция работы в Office приложениях;
- тесты на 3D графику.

Примеры микротестов приведены в таблице 2.

Имя теста	Описание
busyloop_test	Циклическая нагрузка на процессор
system_syscall_test	В цикле выполняет обращения ядру (syscall): Linux: syscall(0x7fffffff);

	macos: getppid(), getpgrp(); win: NtQuerySystemInformation для несуществующего класса)
process_exec_test	Каждую итерацию: - Создает процесс - Ждет завершения процесса - Каждый из процессов не производит полезной нагрузки
thread_create_test	- Создает поток - Ждет завершения потоков по сигналу - Каждый из потоков выделяет память и заполняет ее
io_hdd_seq_rand_rd_test	Последовательно и произвольно читает из файла
virtalloc_test	Выделяет память с помощью virtalloc Освобождает выделенную память
mem_read_test	Выделяет память с помощью virtalloc и читает по 4 байта из каждого блока
mem_write_test	Выделяет память с помощью virtalloc и пишет по 4 байта из каждого блока
mem_pf_read_test	Выделяет память с помощью virtalloc, читает с каждой страниц по 4 байта, освобождает страницу
mem_pf_write_test	Выделяет память с помощью virtalloc, пишет на каждую страницу по 4 байта, освобождает страницу
mem_copy_test	Выделяет страницы с помощью virtalloc и копирует данные между страницами
system_exchandler	Вызывает обработчик исключения через деление на ноль
cpu_cpuid-fast	В цикле зовет CPUID «быстрый» путь
cpu_cpuid-slow	В цикле зовет CPUID «медленный» путь
cpu_cpuid-touch	В цикле CPUID + доступ к 256кб памяти
system_tlbflush	Создает один основной поток, привязанный к процессору #0, и 15 потоков, исполняющий

	занятый цикл (busy loop) и привязанных к любому процессору кроме #0. Основной поток выделяет одну страницу памяти (VirtualAlloc/mmap), грязнит ее, освобождает ее. На каждое выделение/освобождение ОС пошлет запрос на сброс TLB другим процессорам
system_registry	Создает ключ в реестре Windows и наполняет его 100 вложенными ключами и 10 REG_SZ значениями в каждом ключе, сумарно 1000 REG_SZ значений. На каждой итерации теста открывает произвольный ключ, запрашивает произвольное значение из открытого ключа, закрывает ключ.
system_messenger	SendMessage() в дочернему процессу в цикле
time_msleep	sleep 1 мс в цикле
system_guest2host	Переключается из гостевой ОС в гипервизор и обратно
system_guest2vmapp	Переключается из гостевой ОС в командный интерфейс и обратно
process_ctxswitch	Запись/чтение 1 байта в/из дочернего процесса через именновый pipe (FIFO)
thread_ctxswitch(1)	Запись/чтение 1 байта из/в дочерний поток через анонимный apipe
thread_ctxswitch(2)	Увеличивает/уменьшает семафор-переменную в двух дочерних потоках
ext_cpu-irql	KeLowerIrql/KeRaiseIrql в цикле
ext_cpu-cli	Делает { cli, nop, sti, nop } в цикле
ext_cpu-lock	do atomic_inc() (same cache-line) in a loop
ext_cpu-invlpg	do invlpg(addr) in a loop
ext_cpu-rdtsc	Выполняет rdtsc в цикле

ext_cpu-mmio	Выполняет чтение версии APIC в цикле
ext_cpu-rdmsr	Читает MSR 0x40000000 в цикле
ext_cpu-inb	Делает inb 0x42 в цикле
ext_cpu-ins	Делает ins 0x42 d цикле
at_fs-HDD-file_attrib [+]	Запрос файловых атрибутов в цикле
at_fs-HDD-file_create	В цикле выполняет {open(O_CREAT), close(), unlink()}
at_fs-HDD-file_open	Создает файл и в цикле открывает и закрывает его (open(O_RDONLY); close();)
at_fs-HDD-dir_create	Рекурсивно создает директорию и заполняет ее 10 поддиректориями и 10 файлами. Глубина 3. То есть создается 10^3 директорий и 10^3 файлов. Все файлы пусты. Потом все директории и файлы рекурсивно удаляются
at_fs-HDD-dir_readdir	Рекурсивно создает директорию и заполняет ее 10 поддиректориями и 10 файлами. Глубина 3. То есть создается 10^3 директорий и 10^3 файлов. Все файлы пусты. Открывает корневую директорию, рекурсивно читает все директории, закрывает.
at_io-HDD_seq-wr-1M	Создает/открывает один файл размером 2048Мб по имени C:\Windows\temp\atomic_tempf, Стартует один поток и делает последовательные записи 1 мегабайтными блоками
at_io-HDD_seq-rd-1M	Создает/открывает один файл размером 2048Мб по имени C:\Windows\temp\atomic_tempf, Стартует один поток и делает последовательные чтения 1 мегабайтными блоками
at_io-HDD_rand-wr-4K	Создает/открывает один файл размером 2048Мб по имени C:\Windows\temp\atomic_tempf, Стартует один поток и делает произвольные постраничные записи

at_io-HDD_rand-rd-4K	Создает/открывает один файл размером 2048Мб по имени C:\Windows\temp\atomic_tempf, Стартует один поток и делает произвольные постраничные чтения

Таблица 1. Микротесты

Примеры макротестов представлены в таблице 3.

Тест	Описание теста
at_ext_project_compilation	Компиляция проекта MPlayer
at_ext_video_conversion	Перекодирование 17 MB MPG файла а WMV
at_ext_cinebench-cpu_rendering	Исполнение Cinebench 11.5 CPU rendering test
at_ext_java_volano	Исполнение теста Java Volano
at_2D_rect-GDI	Отрисовка 500x500 pixel прямоугольника(GDI) в цикле
at_2D_bitblt	Копирование 500x500 pixel прямоугольника(DDraw) в цикле
at_ext_msoffice	(1) Office 2007 installation: (2) Hot Word 2007 start: (3) Hot Excel 2007 start: (4) Hot PowerPoint 2007 start: (5) Word 2007 Replace test: (6) Excel 2007 Macro test: (7) Office 2007 uninstall:
at_3D_gl_tri-100K-NoTex	Отрисовка 100000 треугольников на фрейм в цикле: mode 0: light off: fog off: texture off: VSync on:
at_3D_gl_tri-5K-MultiTex	Отрисовка 5000 треугольников на фрейм в цикле: mode 0: light on: fog on: texture on: VSync on:

at_ext_msoffice-excel_cold_start	Запуск Microsoft Excel на «холодных» файлах в цикле
at_ext_msoffice-word_cold_start	Запуск Microsoft Word на «холодных» файлах в цикле

Таблица 2. Макротесты

Конечная цель описанного набора состоит в том, чтобы симулировать наиболее распространенные рабочие нагрузки. Прогон набора занял 28 часов, данные о произошедших виртуализационных событиях и о наблюдаемом рабочем наборе собирались каждые 10 секунд и сохранялась в CSV формате. Как результат прогона подготовлена выборка размером чуть более 104 пар.

3.3. Оценка размера рабочего набора на ОС Windows

Для восстановления зависимости использовалась программа порождения моделей нелинейной регрессии Multivariate Regression Composer (MVR composer), разработанная Вадимом Стрижовым [89,90]. MVR composer подбирает оптимальную параметрическую регрессионную модель из набора суперпозиции заданных гладких функций. Этот набор представляет собой экспертное знание, так что виртуализационный опыт будет задействован.

Поиск моделей выполняется по итерационной схеме “порождение-выбор” в соответствии с определенными правилами порождения моделей и критерием их выбора. Последовательно порождаются наборы конкурирующих моделей. Каждая модель в наборе является суперпозицией элементов заданного множества гладких параметрических функций. После построения модели каждому элементу суперпозиции ставится в соответствие гиперпараметр. Параметры и гиперпараметры модели последовательно настраиваются. Из набора выбираются наилучшие модели для последующей модификации. При модификации моделей по значениям гиперпараметров делаются выводы о целесообразности включения того или иного элемента в модель следующего порождаемого набора.

MVR Composer написан на языке Matlab [91]. Для генерации модели ему необходимо:

- несколько свободных переменных и одна зависимая переменная;
- набор регрессионных моделей начального приближения;
- множество порождающих функций, из суперпозиции которых будет построена модель.

На основании параметров, представленных в 3.1 и работы MVR Composer-а появилась формула [21]

$$WS = Cr3Cnt^2 + GuestTime/TotalTime + PgIn/TotalEvent + IoCnt + IrqCnt \quad (1)$$

где все обозначения соответствуют представлению в п.3.1. Коэффициенты опущены для упрощения формулы.

3.4. Верификация оценки

Верификация оценки производилась на тесте vConsolidate. vConsolidate – это парадигма проведения тестирования производительности виртуализационных систем, предложенная компанией Intel [92]. vConsolidate предполагает запуск нескольких виртуальных машин конкурентно на одном хосте. В качестве измерения производительности используется среднее геометрическое нормированных результатов нагрузочного тестирования в виртуальных машинах. Примеры результатов таких тестов: количество транзакций в секунду для базы данных, количество обработанных запросов за секунду для веб-сервера, количество сгенерированных фигур/графических операций в секунду для теста графической подсистемы. Интегральный балл от vConsolidate дает достаточно аккуратное представление о производительности виртуальных машин.

В нашем случае одна единица консолидация (consolidation stack unit – CSU) содержит 4 виртуальные машины Windows XP x32 UP 1024МБ. Виртуальная

машина №0 не исполняет никакой нагрузки (режим простоя). Виртуальная машина №1 исполняет нагрузочный тест на Java сервер - SPECjbb2005 [93]. Виртуальная машина №2 обрабатывает операции по доступу к базе данных от теста SysBench [94]. Виртуальная машина №3 содержит веб-сервер, нагруженный тестом WebBench [95]. Первая итерация vConsolidate исполняет 1 CSU. На каждой итерации число CSU увеличивается на 1. Время одного прогона/итерации – 2 часа.

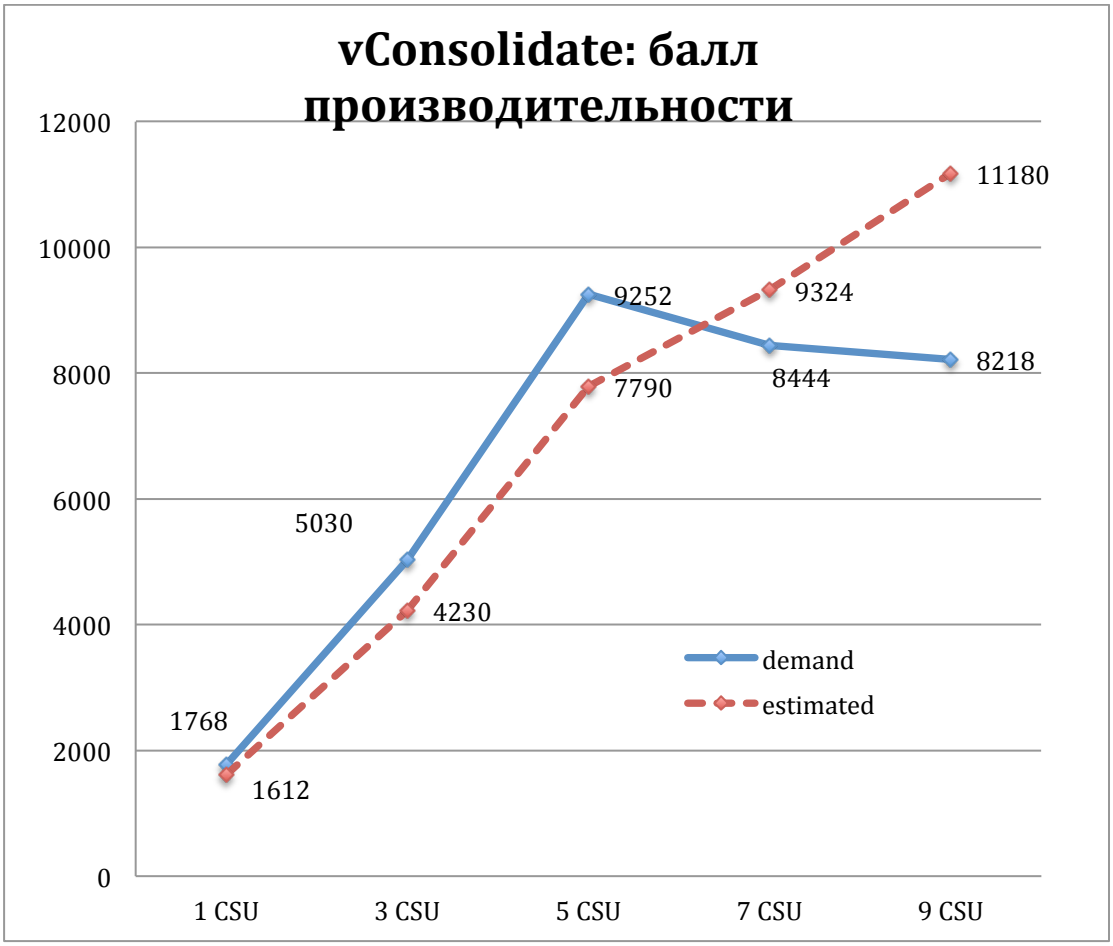


Рис. 5. Производительность vConsolidate

На рис. 5 мы видим результаты vConsolidate. Пунктирная линия (estimated) – это производительность системы, когда фиктивно занимается память в соответствие с нашей оценкой. Сплошная линия (demand) – это производительность системы, когда не используется фиктивно занятой памяти.

Как мы видим, в начале до перелома в точке 6CSU производительность системы без фиктивного занятия памяти лучше. В момент запуска 6CSU наступает состояние переназначения. И тут система с фиктивно занимаемой памятью начинает выигрывать. Это говорит о преимуществе нашего подхода в случае переназначения.

Рис.6 демонстрирует потребление ресурса обеими системами. При отказе от алгоритма «виртуального баллона» рост потребления линеен вплоть до исчерпания ресурса физической памяти. В случае же использования фиктивно занимаемой памяти в соответствии с нашей оценкой потребление ресурса существенно меньше.

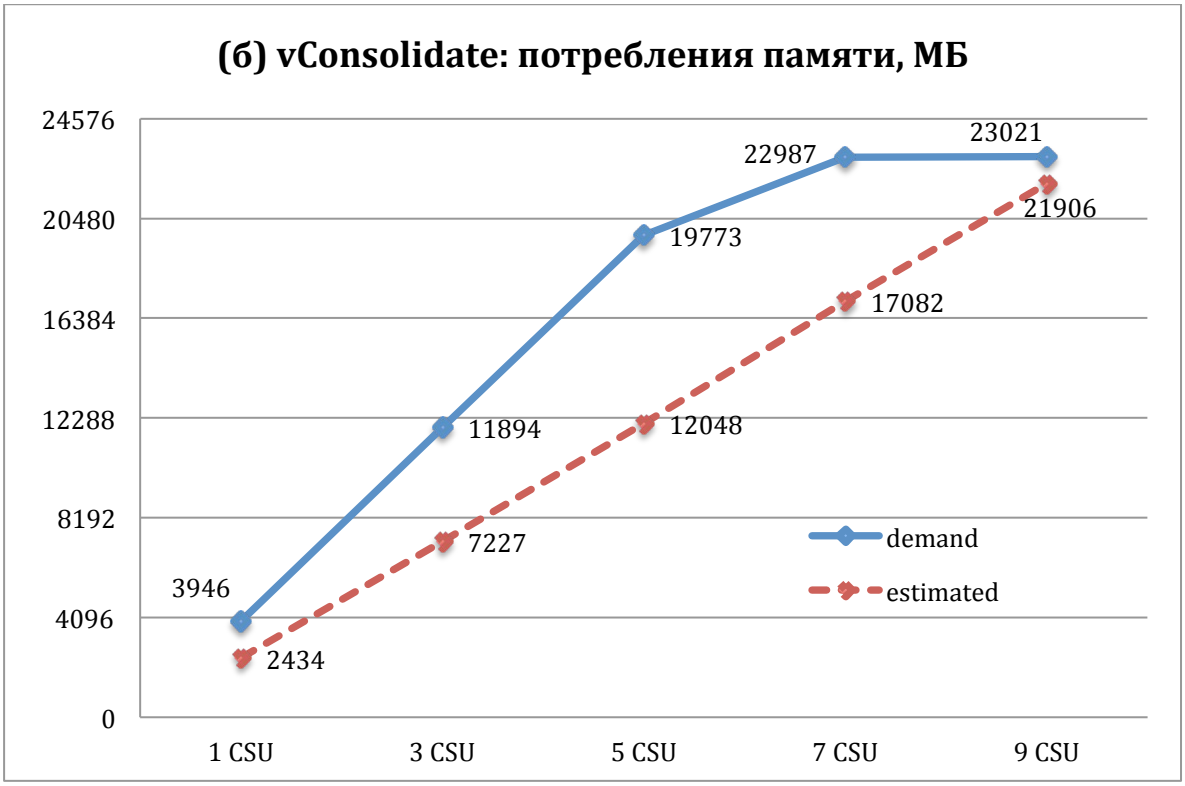


Рис. 6. Потребление памяти на vConsolidate

Однако, несмотря на неплохие результаты данная оценка не может быть применима непосредственно, ибо дает слишком заметное понижение качества обслуживания. Переключение же между моделями динамически, в момент обнаружения состояния переназначения, оказывается сложно технически из-за

возможных колебаний в объеме потребляемой памяти хостовой ОС, что может привести к частным переключениям между моделями и снижением как эффективности, так и надежности системы.

3.5. Анализ гомогенности виртуализационных выборок гостевых систем семейства Windows

Целью исследования была проверка гипотезы, что виртуализационные выборки от близких версий Windows принадлежат одному распределению. Позитивный ответ означал бы, что с точки зрения виртуализации внутрисистемные алгоритмы близких версий ОС выглядят одинаково. А значит при выборе в пользу виртуализационных счетчиков количество исследуемых систем было бы меньше, что, несомненно, сильно склоняло бы к их использованию.

3.5.1. Выбор критерия

Значительная часть проверок критерия однородностей построена на знании закона распределения анализируемых выборок. Для рассматриваемой задачи такое априори может быть лишь догадкой, и, как показано в работе Лео Бреймана [96], может негативно повлиять на результаты исследований. Поэтому предполагается, что распределение неизвестно [23].

Для проверки гипотезы H_0 о принадлежности выборок одному распределению, при неизвестном законе распределения этих выборок применяются непараметрические критерии однородности [97]. Большинство из этих критериев в качестве альтернативной гипотезы используют или гипотезу сдвига, или гипотезу масштаба. Данные гипотезы являются неестественными для

рассматриваемой задачи, так как альтернативные гипотезы сдвига и масштаба изначально несут в себе предположение возможного несовпадения алгоритмов ОС. Например, критерий Вилкоксона, один из наиболее часто встречающихся критериев однородности, с гипотезой и альтернативной гипотезой сдвига H_1, не всегда позволяет обнаружить различие функций распределения [98].

Наиболее очевидной альтернативной гипотезой является$H_1: F(x) \neq G(x)$ $H_1: F(x) \neq G(x)$, т.к. она предполагает совпадение алгоритмов ОС.

В качестве критерия однородности так же можно использовать двухвыборочный критерий согласия. Существует класс двухвыборочных критериев согласия, свободных от распределения, в случае проверки простых гипотез [99, 100]:

- критерий Колмогорова – Смирнова,
- критерий Андерсона,
- критерий ω^2 [101, 99].

Гипотеза H_0 — простая [100]. Следовательно, при помощи этих критериев можно проверить гипотезу H_0 без необходимости знать теоретическое распределение.

Из трех вышеперечисленных критериев наибольшую мощность имеет критерий Андерсона [100], который и был выбран в качестве основного.

3.5.2. Проверка гомогенности

В качестве инструментов для проверки гипотезы однородности использовалась свободная программная среда R, предназначенная для статистических расчетов и работы с графикой, распространяющаяся по лицензии GNU [102].

Для расширения возможностей R использовался дополнительный пакет kSamples - набор ранговых критериев однородности, содержащий, в том числе, критерий Андерсона [103]. Так же использовался сторонний исходный код для расчета критерия Зигеля-Тьюки [104].

Вывод результатов расчета критериев осуществляется с точностью до 6-го знака.

Оценки однородности были выполнены как на выборках, представляющих абсолютную величину используемой физической памяти, так и на выборках, демонстрирующих относительный прирост потребления памяти [23].

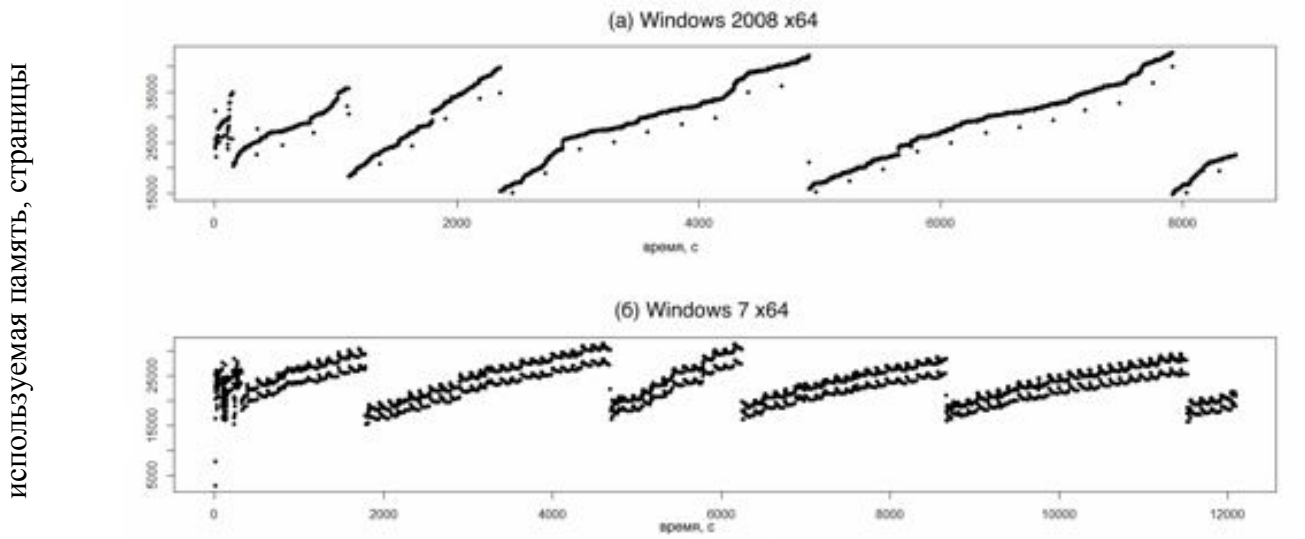


Рис. 7. График потребления памяти в Windows 2008 x64 (а) и Windows 7 x34 (б)

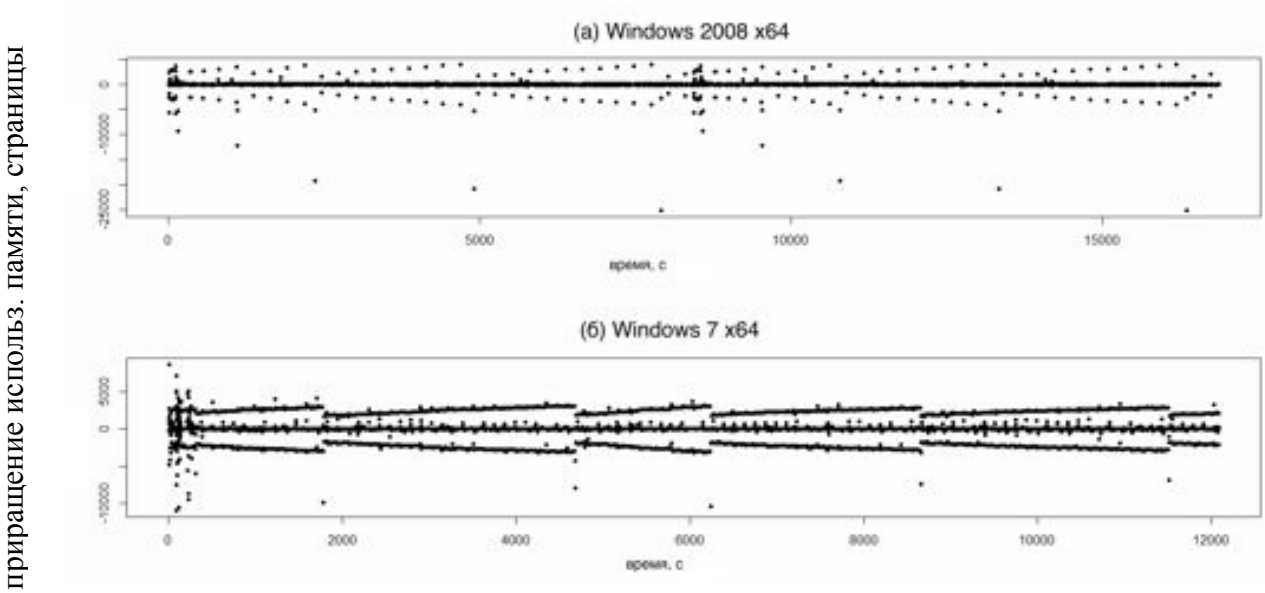


Рис. 8. График приращения используемой памяти (рабочего набора) в ОС Windows 2008 x64 (а) и Windows 7 x64 (б)

Переломы графиков на рис.7(а, б) и на рис.8(а, б) происходят в разные моменты времени на одинаковой нагрузке. Это визуально подтверждает результат анализа данных, выполненный с помощью указанных критериев (таблица 4 и 5): ни типовые схемы потребления памяти, ни типовые схемы приращения объема потребления памяти не принадлежат к одному распределению для разных операционных систем даже близких семейств.

Название критерия	***pValue***	***Однородность***
Wilcoxon	0	Нет
Siegel-Tukey	0	Нет
Anderson–Darling	0	Нет

Таблица 3. Результаты расчётов критериев однородности по потреблению памяти

Название критерия	*pValue*	*Однородность*
Wilcoxon	0	Нет
Siegel-Tukey	0	Нет
Anderson–Darling	0	Нет

Таблица 4. Результаты расчётов критериев однородности выборок роста потребления физической памяти

Похожие результаты получились при сравнении 32-битных операционных систем Windows 7 и Windows XP. На рис. 9 и 10 представлены графики потребления и приращения потребления в этих операционных системах. Переломы опять не совпадают.

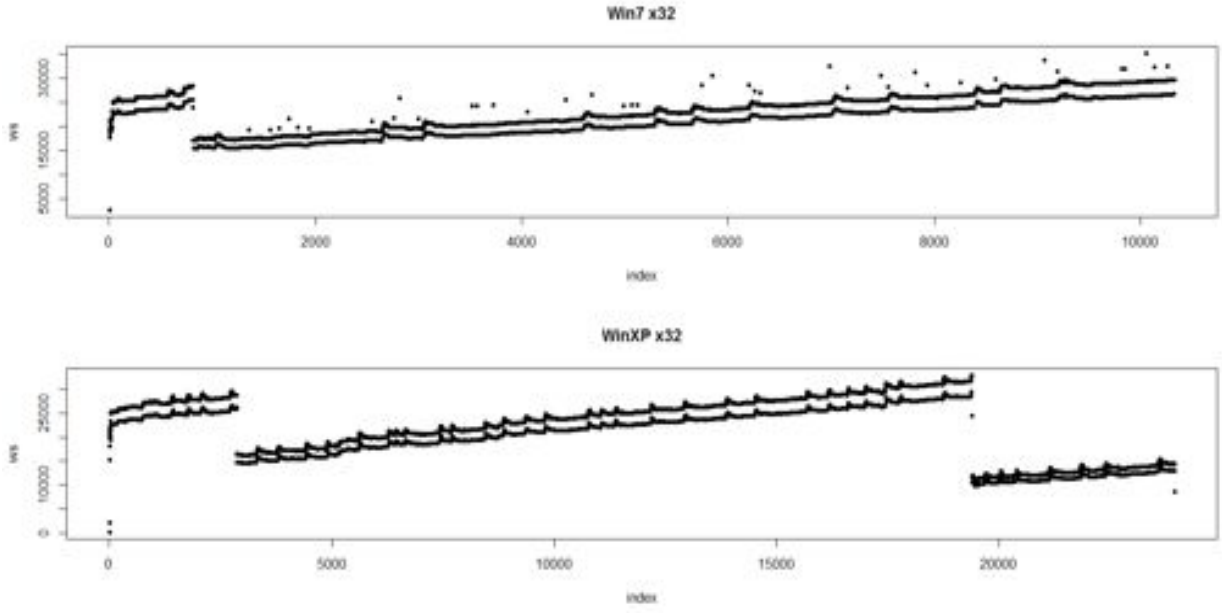


Рис. 9. График потребления памяти в Windows 7 x32 и Windows XP x32

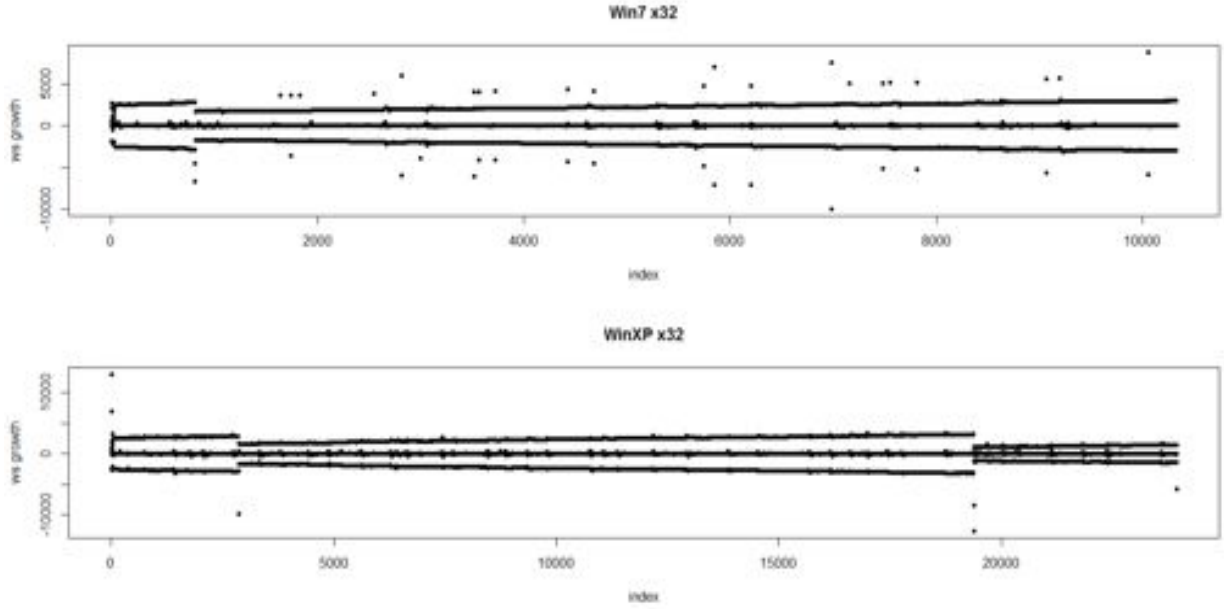


Рис. 10. График приращения потребления памяти в Windows 7 x32 и Windows XP x32

Визуально наблюдаемые различия подкрепляются расчетными данными из таблиц 4 и 5, которые доказывают неоднородность типовых схем использования памяти и приращения темпов ее роста

Название критерия	***pValue***	***Однородность***
Wilcoxon	2e-09	Нет
Siegel-Tukey	0	Нет
Anderson–Darling	0	Нет

Таблица 5. Результаты расчётов критериев однородности выборок по потреблению памяти

Название критерия	***pValue***	***Однородность***
Wilcoxon	4.1997e-05	Нет
Siegel-Tukey	0.2284418	Да
Anderson–Darling	1e-09	Нет

Таблица 6. Результаты расчётов критериев однородности выборок роста потребления физической памяти

Из отрицательного результата проверки теории о гомогенности виртуализационных выборок на схожих операционных системах следует, что даже полагаясь только на виртуализационные счетчики, оценку размера рабочего набора необходимо проводить отдельно для всех основных версий гостевых ОС. Кроме того, стандартными средствами не получилось выделить корреляцию между виртуализационной статистикой и изменением объема рабочего набора. Поэтому было принято решение строить оценку на базе гостевых счетчиков.

.

Глава 4. Оценка размера рабочего набора на основании гостевых счетчиков

4.1. Выбор гостевых счетчиков

Путём исключения нерелевантных показателей был сформирован список из наиболее значимых счетчиков. Ими оказались следующие параметры (названия соответствуют наименования гостевой ОС Windows):

- Commit total – число страниц закрепленной физической памяти;
- Working set size – размер рабочего набора;
- Physical memory usage – количество используемой физической памяти;
- Page file usage – объем использования страничного файла.

4.2. Проведение оценки

В ходе исследований были собраны значения выбранных параметров. Тесты для исследований совпадают с описанными в 3.2, размер гостевой памяти для двух исследуемых систем (Windows 7 и Windows 8) одинаков и составляет 4ГБ. По результатам (рис.11) выяснилось, что: (1) размер рабочего набора гостевой операционной системы Windows 7 фактически равен Commit Total за вычетом системных кэшей, (2) оценка справедлива и для Windows 7 x64 и для Windows 8 x64 систем, что в некоторой степени упрощает разработку и внедрение системы управления памятью. То есть можно сформулировать эмпирическую (инженерную) зависимость для оценки рабочего набора (WSS)

$$WSS = CommitTotal - BalloonSize \qquad (2)$$

где BalloonSize – это размер фиктивно занятой физической памяти (алгоритм «воздушный баллон»).

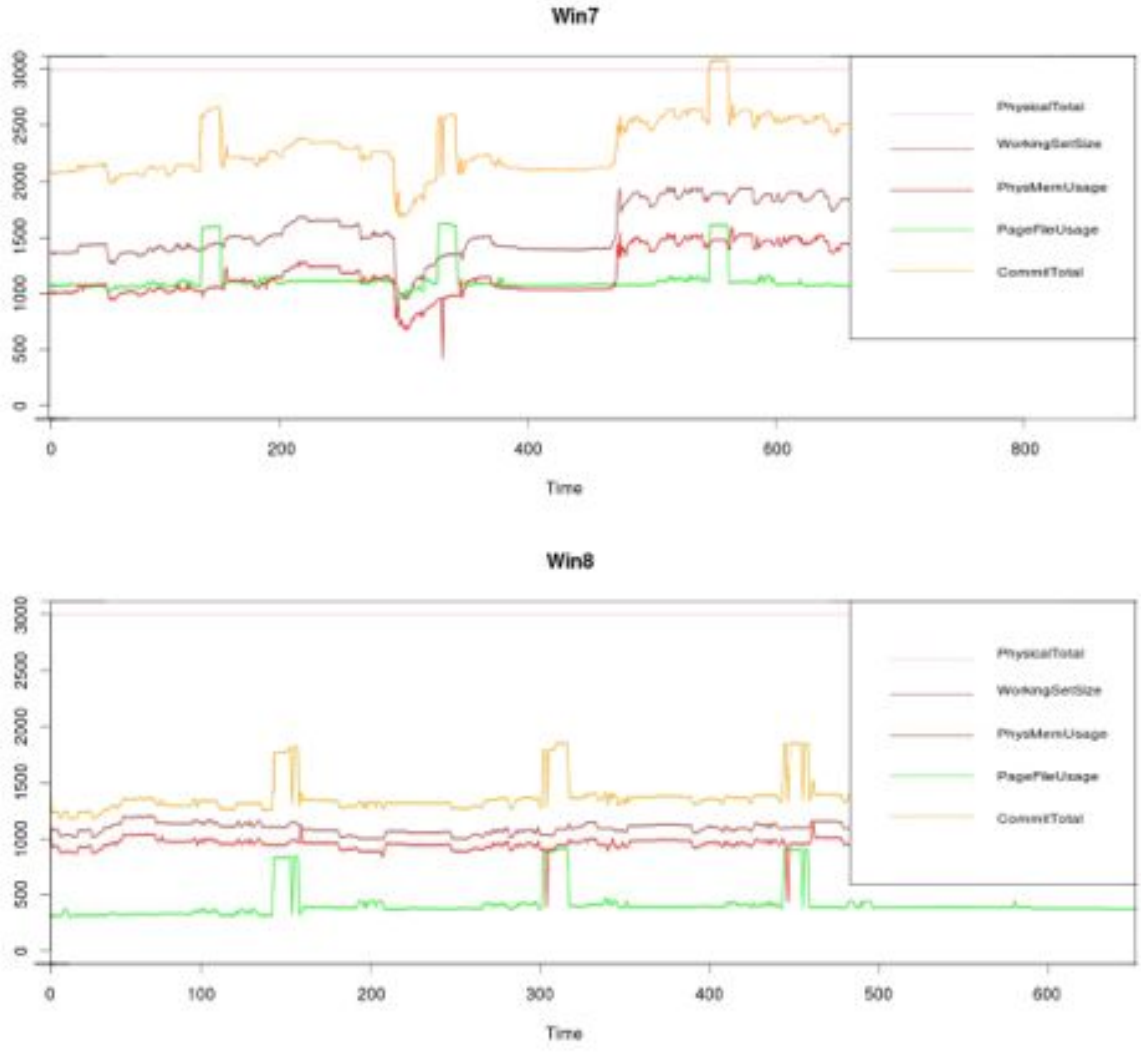


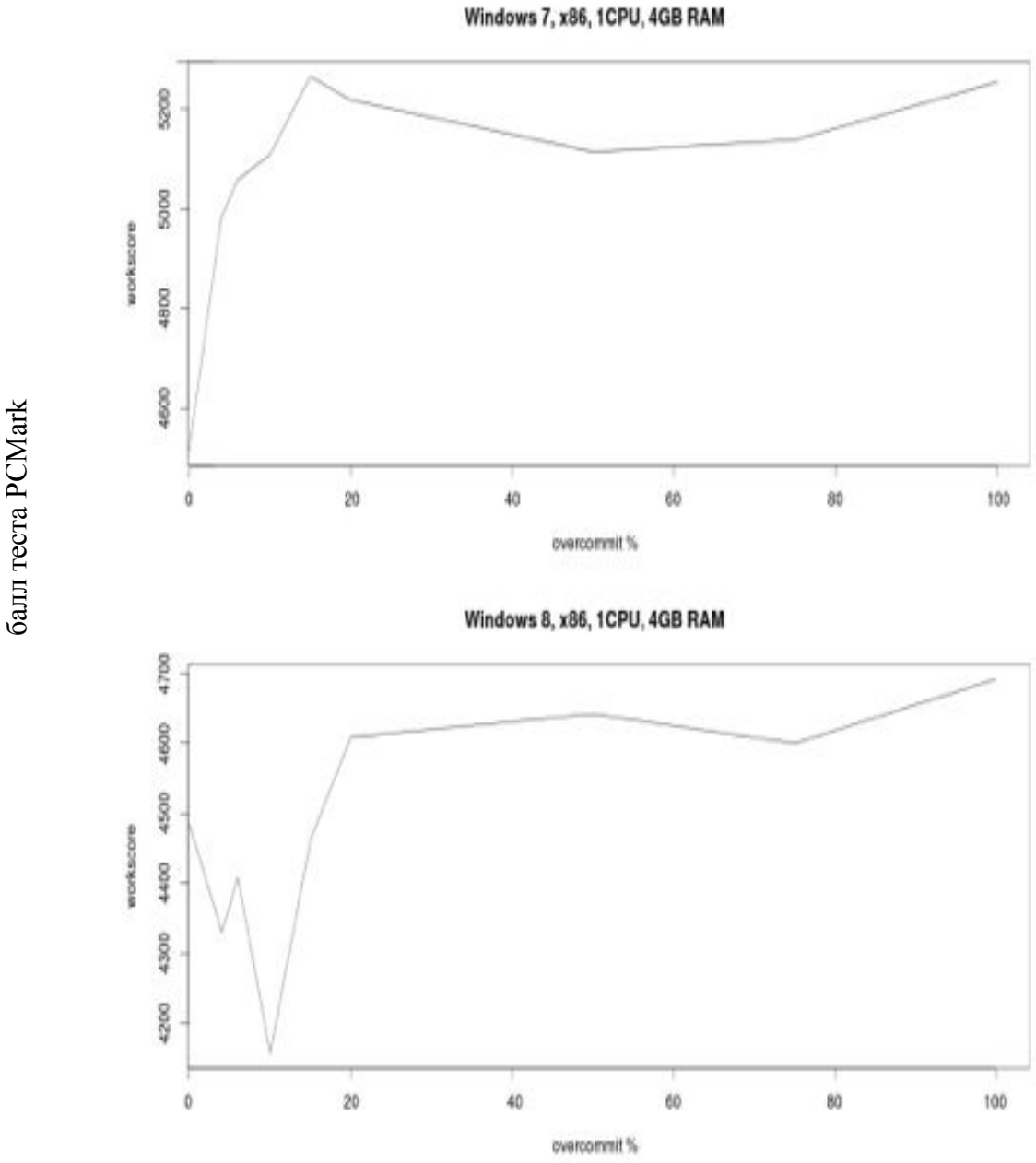
Рис. 11. Гостевые показатели Windows 7 и Windows 8

4.3. Корректировка оценки на размер кэшей

Дальнейшие практические исследования показали, что оценка не принимает во внимание размер кэшей гостевой операционной системы. Применение такого сырого результата для выставления размера фиктивно занятой памяти приведен к снижению производительности, так как оценка завышена. Мы обратили внимание, что размер системных кэшей не связан с текущей нагрузкой, но зависит от объема физической памяти, установленной в ОС. Такая корректировка была предпринята в соответствии с формулами:

$$WSS = CommitTotal - BalloonSize + \sigma * PhysicalTotal \quad (3)$$

где σ - это эмпирический коэффициент. Коэффициент будет меняться от политик управления памятью внутри гостевой ОС, которые отличаются для версий Windows 7 и Windows 8 [105]. Подбор коэффициента осуществлялся на известном тесте PCMark [106]. На рис.12 представлен график, где по вертикали представлен балл теста (чем больше, тем лучше), а по горизонтали отложена σ (в процентах)



балл теста PCMark

Рис. 12. Производительность в зависимости от эмпирического коэффициента кэша

Необходимым критерием эффективности алгоритма управлению фиктивно занятой памятью есть сохранение приемлемого уровня производительности. Для этого было проведено исследование на микротестах из подмножества, описанного в п. 3.2.

Тест	Windows 7, 1CPU, 4Gb	Windows 8, 1CPU, 4GB
busyloop_test	-2.0%	-2,0%
system_syscall_test	-0.1%	-0.3%
process_exec_test	-2.0%	-2.3%
thread_create_test	-2.0%	-1.8%
io_hdd_seq_rand_rd_test	-99.8%	-99.2%
virtalloc_test	-1.1%	-0.2%
mem_read_test	-1.5%	-5.4%
mem_write_test	-1.6%	-2.9%
mem_pf_read_test	-0.2%	-10.0%
mem_pf_write_test	+0.4%	-9.1%
mem_copy_test	-1.2%	-1.6%

Таблица 7. Результаты тестирования производительности

Как видно из результатов в таблице 8, обе системы испытывали серьезное падение производительности на дисковых операциях. Это могло быть связано с нехваткой в системе памяти на дисковые кэши [107]. Для преодоления указанных эффектов были исправлены системные настройки Windows:

1. Разрешен параметр LargeCacheFile для смены политики контроля кэша [108]
2. Установлены лимиты на файловый кэш.

Повторное тестирование производительности показало прирост, что подтверждает правильность гипотезы (таблица 9)

Test	#1. Windows 7 с balloon по сравнению к вм без balloon	#2. Windows 7 с balloon и контролем кэша по сравнению к вм#1	#3. Windows 8 с balloon по сравнению к вм без balloon	#2. Windows 8 с balloon и контролем кэша по сравнению к вм#3
busyloop_test	+1.1%	+3.2%	-1.6%	+0.4
system_syscall_test	-1.0%	-0.9%	+1.1%	+1.4%
process_exec_test	-3.3%	-1.4%	+2.7%	+5.1%
thread_create_test	+3.1%	+5.0%	+3.1%	+5.0%
io_hdd_seq_rand_rd_test	-99.8%	+32.0%	-98.8%	+53.1%
virtalloc_test	-0.4%	+0.7%	+0.2%	+0.4%
mem_read_test	-2.4%	-0.9%	-0.2%	+5.5%
mem_write_test	-2.4%	-0.8%	+6.1%	+9.2%
mem_pf_read_test	-0.3%	-0.0%	-4.9%	+5.7%
mem_pf_write_test	-0.1%	-0.6%	-4.0%	+5.7%
mem_copy_test	-0.9%	+0.3%	+7.7%	+9.4%

Таблица 8. Результаты тестирования производительности

Однако, падение производительности все еще заметно, хоть и не столь катастрофично. Кроме того, редактирование системных гостевых настроек может быть недостаточно хорошо воспринято пользователями.

Таким образом, хотя дополнительные коррекции расчетов на системные кэши и корректировка настроек системы улучшили общую производительность системы, предложенный метод вносит значительное снижение производительности при активном вводе/выводе. Как вывод из исследований, предпринятых в главе 4, для устранения влияния эмпирических допущений, повышения уровня производительности и улучшения адаптивных свойств оценки было принято решение уточнять оценку методами обучения с подкреплением, где вознаграждение поступало бы от виртуализационных счетчиков.

Глава 5. Корректировка оценки от гостевой статистики методами обучения с подкреплением

Как описывалось ранее, любая оценка рабочего набора гостевой ОС не является достаточно точной в силу синтетичности понятия размера рабочего набора. Для управления размером фиктивно занимаемой физической памяти мы хотим корректировать оценку, чтобы повысить производительность системы, в том числе и в момент возникновения новых нагрузок.

5.1. Описание алгоритма

Основная идея предлагаемого алгоритма – редактировать размер фиктивно занимаемой физической памяти (размер balloon) в соответствии с формулой

$$BalloonSize = PhysicalTotal - WSS - gap \quad (4)$$

где WSS – это оценка рабочего набора по формуле (2) (в качестве размера фиктивно занимаемой физической памяти (balloon) используется показатель от предыдущего расчета), PhysicalTotal – это назначенный объем памяти гостевой ОС, gap – это некая величина, смягчающая погрешность оценки, зазор. Причем завысить объем фиктивно занятой памяти значительно опаснее, чем занизить, то есть положительная ошибка существенно хуже отрицательной: в случае завышенного размера возможно резкое падение производительности гостевой ОС, а в случае заниженного - ожидается пропорциональное снижение сэкономленного ресурса (и соответственно снижение плотности размещения виртуальных сред). Автором были предложены к учету следующие факторы для формирования величины зазора (gap):

1. Колебание объема фиктивно занимаемой физической памяти (ballooned memory) негативно сказывается на производительности. Нужно по возможности сохранять его размер, пока вознаграждение от

альтернативного зазора (gap) не превысит проигрыш от изменения размера.

2. Величина зазора и скорость его изменения не должны выходить за определенные границы, иначе возрастают накладные расходы на содержание системы.
3. Возросшее число виртуализационных страничных промахов и операций ввода/вывода могут служить показателями неэффективности текущего зазора.

В соответствие с третьим предположением была сформирована следующая формула для расчета штрафа на изменение зазора

$$fine = (\Delta io - io\_thres) * io\_fine + (\Delta pgin - pgin\_thres) * pgin\_fine \quad (5)$$

где Δio, $\Delta pgin$ – это изменение числа операций ввода/вывода/числа виртуализационных страничных промахов за оцениваемый интервал, io_thres , $pgin\_thres$ - это пороговое значение, приемлемый уровень возрастания числа операций ввода/вывода / числа виртуализационных страничных промахов, io_fine , $pgin\_fine$ – штраф за возрастание/приз за снижение.

Обучение с подкреплением обеспечивает выбор лучшего из вариантов приращения gap в (100 – ε) случаев. В (ε) случаях приращение зазора выбирается произвольно.

Псевдокод алгоритма выбора зазора выглядит как

```
(1) gap_chng[GAP_TO_IDX(prev_gap_change)] -= fine();
(2) if( greedy_degree && (++iter)%greedy_degree == 0 )
(3)             gap = ALIGN_GAP(gap + rand64()%GAP_CHANGE_MAX);
(4) else
(5)   {
(6)       gap = ALIGN_GAP( ws - prev_ws + prev_gap );
```

```
(7)          gap_change = gap – prev_gap;
(8)          tgt_i = GAP_TO_IDX(gap_change); //target index
(9)         for( i = 0; i < GAP_CHANGE_ARRAY; i++ )
(10)              if( gap_chng[i] – gap_chng[tgt_i] > gap_chng_threshold )
(11)                  tgt_i = i;
(12)         if( tgt_i !=  GAP_TO_IDX(gap_change) )
(13)             gap = ALIGN_GAP( prev_gap + gap_chng[tgt_i] );
(14)     }
(15)     prev_gap_change = prev_gap - gap;
(16)     prev_gap = gap;
(17)     prev_ws = ws;
```

5.2. Описание программного комплекса

Программный комплекс выполнен как набор изменений к нескольким модулям системы, изолированных друг от друга разными адресными пространствами. Опишем задачи, реализуемые комплексом:

1. сбор выбранных параметров с гостевой операционной системы;
2. сбор выбранных параметров с виртуализационной системы;
3. получение оценки размера рабочего набора и ее корректировка в соответствии с алгоритмом:
4. выставление нового значения фиктивно занятой памяти.

5.2.1. Реализация в Parallels Server

Рис. 13. Реализация алгоритма в Parallels Server

На рис.13 представлено модульное видение реализации алгоритма. Зеленые прямоугольники справа представляют собой внесенные изменения. Оранжево-красные прямоугольники слева представляют собой отдельные модули виртуализационной системы. В агенте фиктивного занятия памяти мы производим сбор гостевой статистики и тут же мы оцениваем размера рабочего набора. Подсчитанный размер мы передаем через разделяемую структуру даных интерфейсаVirtIO [109] в монитор виртуальной машины. В мониторе виртуальной машины раз в секунду величина рекомендуемого размера фиктивно занятой памяти корректируется на основании оценки, полученной через VirtIO и виртуализационной статистики, доступной в модуле. В случае, если размер фиктивно занятой памяти изменился, выставляется флаг изменений, который

будет обработан гостевой ОС. Обновленный размер фиктивно занятой памяти также будет учитываться гипервизором для распределения памяти между виртуальными машинами. Со стороны командного и графического интерфейса добавлена поддержка на перевод работы алгоритма фиктивного занятия памяти в автоматический режим. Весь программный комплекс написан на языке С.

5.2.2. Реализация в qemu-kvm

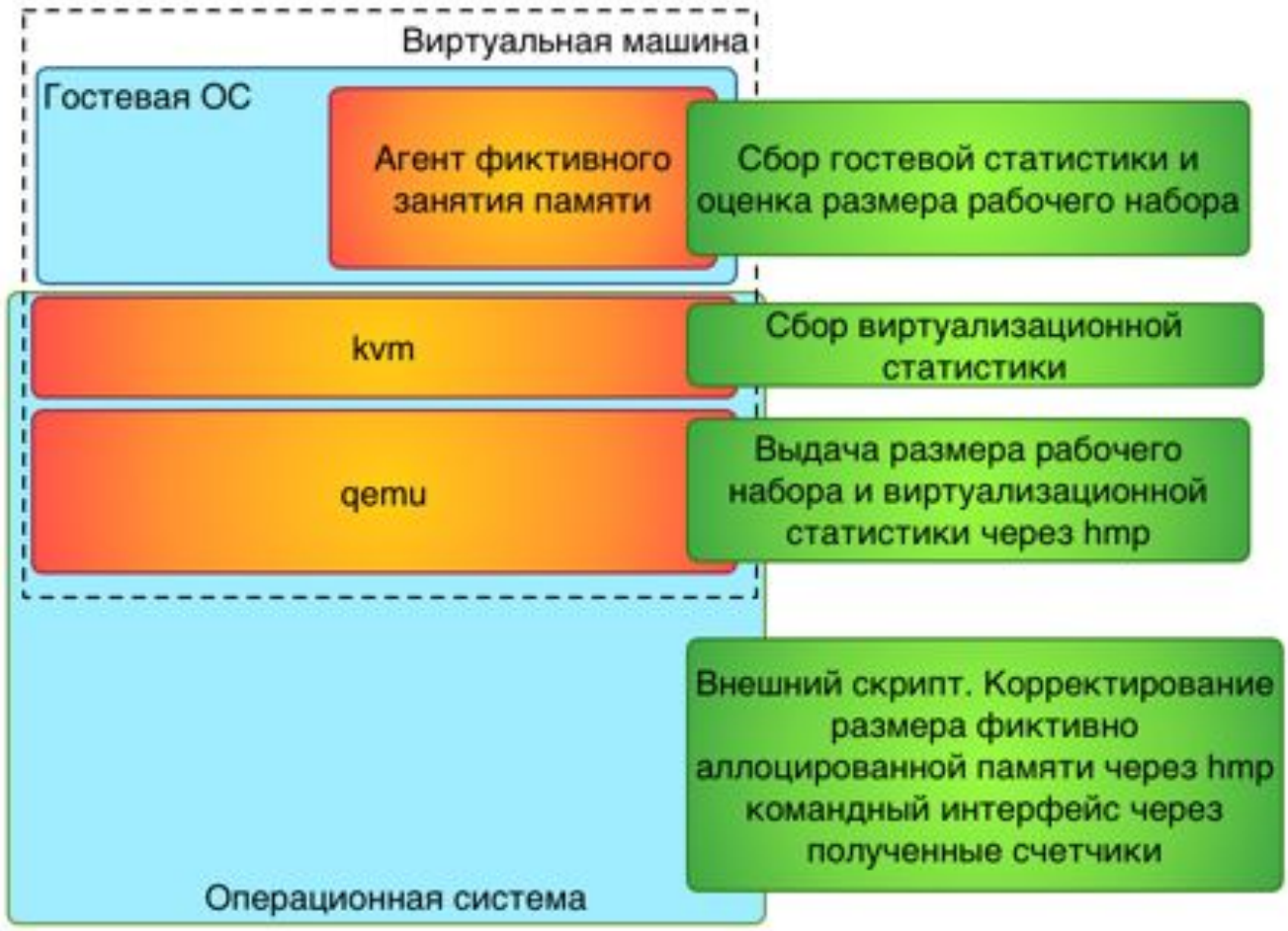


Рис. 14. Реализация алгоритма в qemu-kvm

На рис.14 представлено модульное видение реализации алгоритма в qemu-kvm. Зеленые прямоугольники справа представляют собой внесенные изменения. Оранжево-красные прямоугольники слева представляют собой отдельные модули виртуализационной системы.

Сбор гостевой статистики и оценка размера рабочего набора вставлялась в агент фиктивного занятия памяти (VirtIO balloon для Windows и Linux). Сбор виртуализационной статистики осуществлялся в KVM. Добавить там же корректировку размера фиктивно занятой памяти не представляется возможным, так как KVM не обладает знанием о virtIO устройстве. В qemu добавлено предоставление размера рабочего набора (посчитанного в гостевом агенте фиктивного занятия памяти и полученного через VirtIO) и виртуализационной статистики через стандартный программный интерфейс hmp. Внешний скрипт на языке Python управляет размером фиктивно занятой физической памяти, передавая команды qemu на основании полученных от него же данных. Не считая внешнего управляющего скрипта, весь программный комплекс написан на языке C.

5.3. Настройка параметров алгоритма

В алгоритме многое зависит от ε - степени «жадности» алгоритма (в псевдокоде *greedy_degree)*. Чем больше ε, тем быстрее алгоритм обучается, но тем хуже его производительность на время обучения. Значения ε подбирались экспериментальным путем на наборе тестов, в которые включались тесты, перечисленные в п. 3.2 и PCMark.

Этот набор тестов оказывается достаточно полным с точки зрения различных сценариев использования виртуальных машин, как мы можем отметить из нашего опыта работы с хостинг-провайдерами.

Тесты показали, что при использовании алгоритма существенно снизилось потребление памяти (рис. 15). Здесь эталон – виртуальная машина без фиктивного занятия памяти (без ballooning). «Жадный» алгоритм — это ε=2, «щедрый» алгоритм — ε=20, а фиксированный – это постоянное значение зазора без

адаптации (gap=64MB). Примечательно, что для разных параметров алгоритма и среднее потребление памяти и ее отклонения примерно одинаковы. Это обусловлено длительностью теста и разнообразием его нагрузок. Объем «сэкономленной» памяти на скромной конфигурации (1GB RAM) составил 40%, что представляется хорошим результатом.

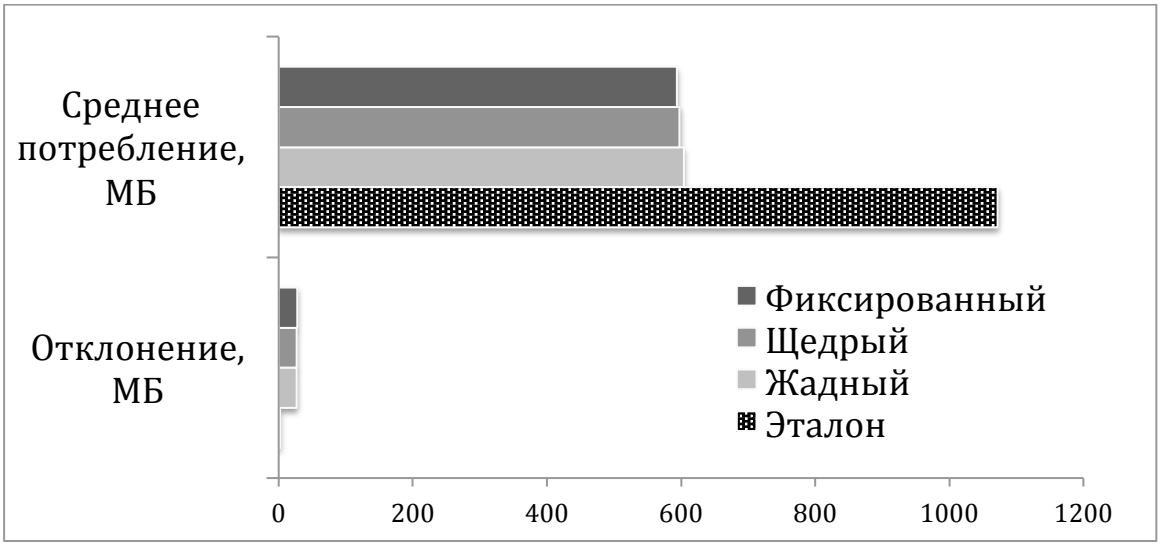


Рис. 15. Уровень потребления памяти

При этом снижение производительности незначительное, как видно из результатов тестов, представленных в таблице 10.

	Эталон	Жадный		Щедрый		Фиксированный	
Тест	**Балл**	**Балл**	**vs #1**	**Бадд**	**vs #1**	**Балл**	**vs #1**
sys_busyloop	252392	244313	**-3%**	259233	**+3%**	255197	**+1%**
sys_syscall	1538780	1534676	**0%**	1595706	**+4%**	1557842	**+1%**
sys_exchandler	128072	126602	**-1%**	135751	**+6%**	133692	**+4%**
cpu_cpuid-fast	222211	219701	**-1%**	225078	**+1%**	218373	**-2%**

cpu_cpuid-slow	1133195	1166001	+3%	1209299	+7%	1187391	+5%
cpu_cpuid-touch	799583	879250	+10%	897229	+12%	886960	+11%
sys_tlbflush	21371	21236	-1%	19417	-9%	21874	+2%
sys_registry	93171	90140	-3%	71582	-23%	89861	-4%
sys_messenger	145976	146663	+0%	123421	-15%	145189	-1%
time_msleep	529,376	534,879	+1%	554,344	+5%	524,484	-1%
sys_guest2host	133744	137208	+3%	130490	-2%	134038	+0%
sys_guest2vmap	108617	110439	2%	108049	-1%	107739	-1%
process_exec	364,449	364,199	0%	360,899	-1%	364,110	0%
process_ctxswch	240968	242521	+1%	238883	-1%	238816	-1%
thread_create	9951,74	9821,30	-1%	9701,72	-3%	9690,58	-3%
thread_ctxsw(1)	504603	505509	+0%	513614	+2%	518783	+3%
thread_ctxsw(2)	1151361	1170165	+2%	1152244	+0%	1151633	+0%
ext_cpu-irql	546753	537025	-2%	543258	-1%	511887	-6%
ext_cpu-cli	316306	319805	+1%	322145	+2%	290534	-8%
ext_cpu-lock	120723	120906	+0%	120081	-1%	115915	-4%
ext_cpu-invlpg	155058	154485	0%	155182	+0%	148456	-4%
ext_cpu-rdtsc	985564	980031	-1%	977803	-1%	954163	-3%
ext_cpu-mmio	434157	438238	+1%	445973	+3%	420119	-3%
ext_cpu-rdmsr	1462940	1465157	+0%	1477567	+1%	1360237	-7%
ext_cpu-inb	1299068	1294616	0%	1308078	+1%	1203080	-7%
ext_cpu-ins	509863	517752	+2%	520224	+2%	488015	-4%
mem_alloc-64M	375946	365414	-3%	373817	-1%	368146	-2%
mem_alloc-4K	536729	548355	+2%	558552	+4%	536443	0%

mem_read	7794,13	7776,06	**0%**	7807,76	**+0%**	7809,45	**+0%**
mem_write	6859,48	6887,64	**+0%**	6953,21	**+1%**	6065,83	**-12%**
mem_pf_read	567050	573753	**+1%**	561435	**-1%**	548744	**-3%**
mem_pf_write0	606331	607039	**+0%**	595421	**-2%**	580814	**-4%**
mem_copy	6464,35	6525,023	**+1%**	6510,458	**+1%**	5980,721	**-7%**
mem_copy-L1	23135	22968	**-1%**	23208	**+0%**	22522	**-3%**

Таблица 10. Тестирование производительности с разными параметрами алгоритма

В первом столбце представлен эталон – результаты тестирования без фиктивного занятия памяти (без ballooning). В втором столбце представлен результат «жадного» алгоритма ($\varepsilon = 2$). В третьем столбце результат «щедрого» алгоритма ($\varepsilon = 20$). В четвертом столбце результат алгоритма с фиксированным размером зазора (gap=64MB). Цифрами в «цветом» столбце указаны в процентах потери производительности по отношению к эталону. Светло-желтый означает паритет, зеленый – это прибавка производительности, бордово-кирпичный с белыми цифрами – это падение производительности. Из этих результатов видно, что жадный алгоритм практически не теряет производительности. В случае щедрого алгоритма возникают как внезапные улучшения, так и ухудшения производительности (и отклонения в результатах теста существенные). В случае фиксированного размера зазора имеем стабильную потерю производительности, что доказывает необходимость его динамической адаптации.

Еще одним полезным параметром алгоритма является устойчивость размера зазора по отношению к штрафу. Иными словами, насколько привлекательным для нас должен быть приз, чтобы мы изменили размер фиктивно занятой памяти с

последующими накладными расходами на выделение/освобождения страниц. Результаты представлены в таблице 11.

	threshold=50	threshold=5		threshold=500		
Тест	Результат	Результат	vs #1	Результат	vs #1	vs #2
mem_virtalloc-64M	328217	292091	-11%	359613	+10%	+23%
mem_virtalloc-4K	491371	484881	-1%	544989	+11%	+12%
mem_read	7338,2895	6779,5143	-8%	7558,7003	+3%	+11%
mem_write	6407,2787	5643,3749	-12%	6791,4921	+6%	+20%
mem_pgfault_read	527283	487671	-8%	540066	+2%	+11%
mem_pgfault_write	557680	522617	-6%	562313	+1%	+8%
mem_copy	6067,2118	5190,8652	-14%	6372,5031	+5%	+23%
mem_copy-L1cache	21439	19729	-8%	22753	+6%	+15%
system_busyloop	2349491	2309801	-2%	2344738	0%	+2%
system_syscall	1460111	1508436	+3%	1388513	-5%	-8%
system_exchandler	106086	124806	+18%	120006	+13%	-4%
cpu_cpuid-fast	2107187	2167575	+3%	2172306	+3%	+0%
cpu_cpuid-slow	1114448	1143829	+3%	1154582	+4%	+1%
cpu_cpuid-touch	836960	852677	+2%	784095	-6%	-8%
system_tlbflush	19536	19016	-3%	20056	+3%	+5%
system_registry	87839	84901	-3%	84355	-4%	-1%
system_messenger	133599	134070	+0%	146673	+10%	+9%
time_msleep	550,1142	545,9726	-1%	541,1678	-2%	-1%
system_guest2host	135348	133925	-1%	129126	-5%	-4%
system_guest2vmap	110076	109578	0%	108000	-2%	-1%
process_exec	346,3321	347,8284	+0%	358,6468	+4%	+3%
process_ctxswitch	260822	260722	0%	244878	-6%	-6%
thread_create	9488,7785	9360,08017	-1%	9663,918	+2%	+3%
thread_ctxswitch(1)	483078	476185	-1%	500168	+4%	+5%
thread_ctxswitch(2)	1085665	1001018	-8%	1127259	+4%	+13%

Таблица 11. Тестирование влияния параметра чувствительности (threshold)

При обучение с подкреплением важно найти компромисс между исследованием неизученных областей и применением имеющихся знаний. В какой-то момент многие из алгоритмов прекращают обучение с целью увеличения прибыли. Однако, в данном случае время жизни виртуальной машины оказывается незначительным (в соответствии со статистикой от хостинг-провайдеров), поэтому обучение небольшими темпами продолжается непрестанно.

5.4. Прикладное использование алгоритма

Алгоритм, реализованный в qemu-kvm, использовался на тестовом стенде Индустриального партнёра Акронис при экспериментальном исследовании распределенной системы хранения данных (СХД). На стенде моделировались наиболее важные тестовые сценарии, дающие исчерпывающее представление о характеристиках исследуемой СХД.

Экспериментальное исследование РСХД с избыточным резервированием представляло собой серию тестов, имитирующих отказ оборудования при заданной типовой нагрузке со стороны пользователей. Выполнялся прогон батареи тестов по штатному выдерживанию нагрузки, возникновениям сбоев дисков и серверных узлов, а также восстановлению хранимых данных после сбоя из резервных реплик. Полное описание экспериментов на стенде Индустриального партнёра ООО Акронис представлено в приложении Б.

Описанное в приложении стендовое тестирование РСХД требует значительных вычислительных мощностей для воспроизведения пользовательской нагрузки требуемой интенсивности. Поскольку исследование РСХД включает в себя и фактор масштабируемости, требуется задействовать

количество машин, превышающее физически имеющуюся аппаратную номенклатуру на стенде. Подобное требование можно удовлетворить с помощью развёртывания виртуализационных решений.

Средства виртуализации позволили, в частности, выполнить моделирование обращения пользователей к системе. Вместе с этим, точность и скорость проведения эксперимента напрямую определяется количеством развёрнутых виртуальных машин. Применение улучшенной экономичной версии qemu-kmv с фиктивным занятием памяти позволило увеличить количество одновременно работающих гостевых клиентских систем на 50%.

Реализация алгоритма фиктивного занятия памяти в программе Parallels Server также показывает потенциал повышения производительности. На рис. 16 приведены результаты теста СХД Parallels Cloud Storage по восстановлению одного терабайта данных после смоделированного сбоя системы.

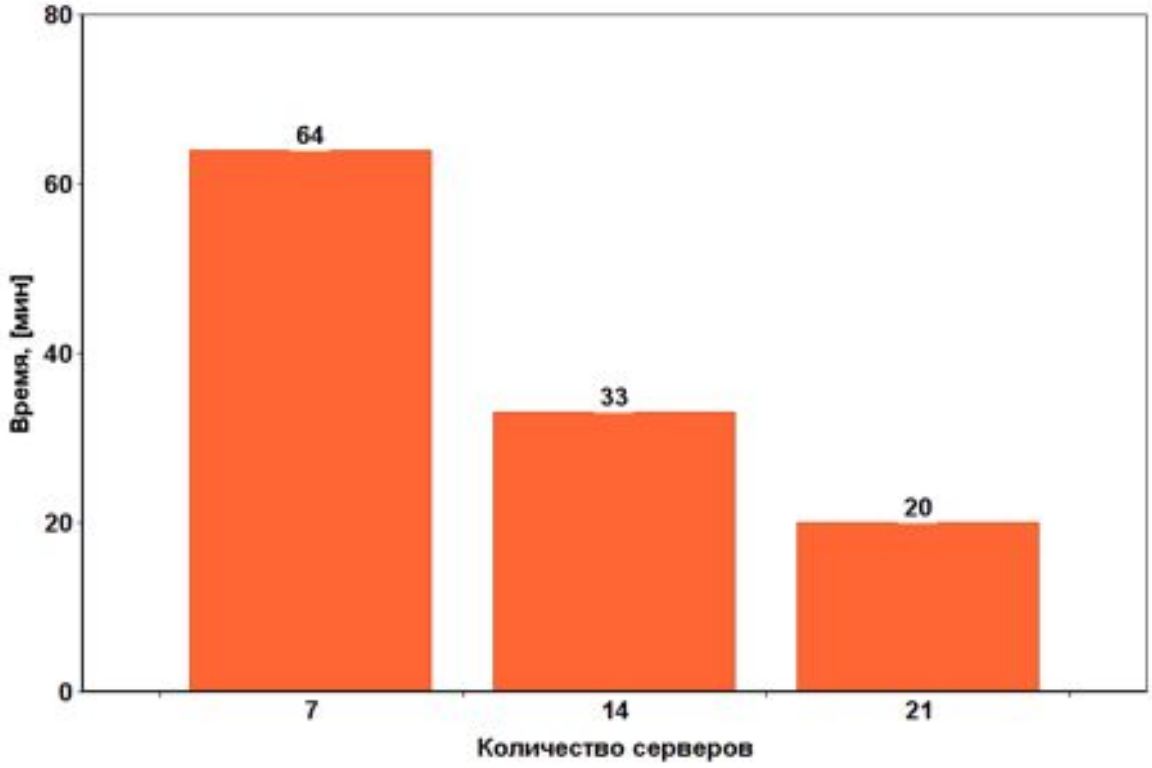


Рис. 16. Время восстановления одного терабайта данных в СХД Parallels Cloud Storage

Из графика видно, что удвоение количества машин позволяет сократить время восстановления данных в два раза. Это свидетельствует о значительной практической полезности предложенного алгоритма балансировки памяти виртуальных машин.

Заключение. Основные результаты и выводы диссертации

1. Проведен анализ существующих проблем и имеющихся решений в области управления памятью в гипервизорах

2. Выполнен сравнительный анализ виртуализационной и гостевой статистики с целью их применения для создания оценки рабочего набора

3. Совместно с Маркеевой Л.Б. предложена математическая модель оценки размера рабочего набора операционных систем семейства Windows на основании гостевых счетчиков.

4. Разработан метод корректировки оценки из п.3 на основании виртуализационной статистики методами обучения с подкреплением.

5. Реализован алгоритм управления размером balloon на основании оценки из п.3, скорректированной методом из п.4.

6. Проведен ряд экспериментов на реальных программных комплексах и на модельных тестов для настройки параметров модели и для проверки ее эффективности.

7. Эмпирически доказан выигрыш от реализованного алгоритма.

8. Реализация внедрена в программный комплекс Parallels Server.

9. Предложенный алгоритм также перенесен на систему с открытым исходным кодом Linux (расширения гостевой ОС KVM).

Благодарности

Хотелось бы выразить благодарности за большую помощь в работе над этой диссертационной работе прежде всего моим близким – маме Мининой И.А. и мужу Мелехову Г.В. – за их неустанную поддержку в моем труде. Огромная благодарность Минину П.В. за идею этой диссертационной работы, за его советы во время работы, за его корректуру и за помощь в презентовании полученного результата. Спасибо Воронцову В.В. (Вычислительный центр РАН) за его предложение испробовать обучение с подкреплением: без этого предложения результаты работы были бы значительно хуже. Огромная благодарность Винникову В.В. (компания Акронис) за его неустанную корректуру текста, поддержку и общие замечания. Хотелось бы поблагодарить Аветисяна А.И. и Зеленова С.В. из Института Системного Программирования РАН за их ответы на мои многочисленные вопросы, а также Козлова И.С. за его старания в поиске корреляцию между виртуализационными счетчиками и размером рабочего набора.

Выражаю благодарности своим коллегам по компании Parallels - Паршикову С.В. за долговременное сотрудничество в запуске теста vConsolidate, Овчинникову Ю.С. за помощь в устранении kernel panic-ов после перехода на новую версию PCS.

Мои благодарности Тормасову А.Г. за помощь в поиске информации и материальных ресурсов и за помощь в представлении работы широкому кругу специалистов, а также за общее руководство работой.

Список литературы

1. Microsoft Hyper-V https://technet.microsoft.com/library/cc816638(WS.10).aspx
2. Linux KVM http://www.linux-kvm.org/page/Main_Page
3. Apple “What’s new in OS X” https://developer.apple.com/library/mac/releasenotes/MacOSX/WhatsNewInOSX/Articles/MacOSX10_10.html
4. Intel virtualization technology http://www.intel.com/content/www/us/en/virtualization/virtualization-technology/intel-virtualization-technology.html
5. AMD virtualization http://www.amd.com/en-us/solutions/servers/virtualization
6. ARM virtualization extensions https://www.arm.com/products/processors/technologies/virtualization-extensions.php
7. NVIDIA Virtual GPU technology for Hardware Accelaration http://www.nvidia.com/object/grid-technology.html
8. Lowe, Scott D. "Best practices for oversubscription of CPU, memory and storage in vSphere virtual environments." Technical Whitepaper, Dell (2013).
9. Vijayaraghavan Soundararajan and Jennifer M. Anderson. 2010. The impact of management operations on the virtualized datacenter. SIGARCH Comput. Archit. News 38, 3 (June 2010), 326-337.
10. VMware White Papers “Business and Financial Benefits of Virtualization” https://www.vmware.com/files/pdf/cloud-journey/VMware-Business-Financial-Benefits-Virtualization-Whitepaper.pdf
11. VMware White Papers “The Operational Impact of Virtualization in the Datacenter” https://www.insight.com/content/dam/insight/en_US/pdfs/vmware/Operational-Impact-of-Virtualization-in-the-Datacenter-Opex-Savings-Whitepaper.pdf

12. Microsoft “How to Save on Power Consumption by Consolidating Servers Using Virtualization” https://technet.microsoft.com/en-us/virtualization/green_it_virtualization.aspx
13. VMware “How VMware Virtualization Right-sizes IT Infrastructure to Reduce Power Consumption” http://www.vmware.com/files/pdf/green_wp.pdf
14. Banerjee, Ishan, et al. "Memory overcommitment in the ESX server." VMware Technical Journal 2 (2013).
15. Gulati, Ajay, et al. "Vmware distributed resource management: Design, implementation, and lessons learned." VMware Technical Journal 1.1 (2012): 45-64.
16. Intel® 64 and IA-32 Architectures Software Developer Manuals http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html
17. Воробьева, А.Л. Стратегия виртуализации физической памяти, применяемые в виртуальных машинах. / А. Л. Воробьева // Процессы и методы обработки информации. сб. науч. тр. — М.: МФТИ, 2009.
18. Expansion of virtualized physical memory of virtual machine: патент US8135899 B1 США, МПК G 06 F12/00 / N. N. Dobrovolskiy, A. A. Omelyanchuk, A. B. Koryakin, A. L. Vorobyova, A. G. Tormasov, S. M. Beloussov; заявитель и патентообладатель Parallels IP Holdings GmbH — № US 13/084,262; заявл. 11.04.2011; опубл. 13.03.2012.
19. Воробьева, А.Л. Методика определения размера рабочего набора виртуальной машины. / А. Л. Воробьева // Математическое моделирование информационных систем. сб. науч. тр. — М.: МФТИ, 2012.
20. Воробьева, А.Л. Управление памятью в гипервизоре. Все о виртуализации памяти в Parallels. / А. Л. Воробьева // Разработка

высоконагруженных систем. сб. тр. конф. Highload++, Москва, 3–4 октября 2011. — М.: Изд-во Олега Бунина, 2012.

21. Melekhova, A. Machine Learning in Virtualization: Estimate a Virtual Machine's Working Set Size / A. Melekhova // Proceedings on 2013 IEEE Sixth International Conference on Cloud Computing, CLOUD 2013.
22. Бондарь, А.О. Энергосбережение изнутри: что в действительности могут измерить профилировщики / А. О. Бондарь, Д. К. Карпов, А. Л. Мелехова // RSDN Magazine, М.: 2013
23. Маркеева, Л. Б. Проверка гипотезы однородности виртуализационных событий, порожденных различными операционными системами / Л. Б. Маркеева, А. Л. Мелехова, А. Г. Тормасов // Труды МФТИ. — 2014. — Т. 6. — С. 57–64.
24. Бондарь, А. О. Алгоритмы решения задачи динамического управления питанием в облачной системе / А. О. Бондарь, Н. В. Ефанов, А. Л. Мелехова // Программная инженерия. — М. — 2015. — №4. — С. 20–30.
25. Melekhova, A. Estimating working set size by guest OS performance counters means / A. Melekhova, L. Markeeva // Proceedings CLOUD COMPUTING 2015, The Sixth International Conference on Cloud Computing, GRIDs, and Virtualization. — 2015. — С. 48.
26. Kudinova, M. CPU prediction models // M. Kudinova, A. Melekhova, A. Verinov // Proceedings of the 11th Central & Eastern European Software Engineering Conference in Russia. — 2015. — (готовится к печати)
27. Carl A.Waldspurger. “Memory resource management in vmware esx server.” SIGOPS Oper. Syst. Rev., 36(SI):181–194, 2002. ISSN 0163- 5980.
28. Dan Magenheimer. “Memory overcommit. . . without the commitment”, Oracle Corp, Extended Abstract for Xen Summit, June 2008 https://oss.oracle.com/projects/tmem/dist/documentation/papers/overcommit.pdf

29. Richard WM Jones, “Virtio balloon”, June 17, 2010 https://rwmj.wordpress.com/2010/07/17/virtio-balloon/
30. “Oracle VM Vitrtual Box. User manual”. Chapter 4 “Guest additions. Memory overcommitment” http://www.virtualbox.org/manual/ch04.html - id399892
31. “Implementing and configuring dynamic memory”. Microsoft, Oct 2010 http://download.microsoft.com/download/E/0/5/E05DF049-8220-4AEE-818B-786ADD9B434E/Implementing_and_Configuring_Dynamic_Memory.docx
32. VMware white papers. “Understanding Memory Resource Management in VMware® ESXTM Server” http://www.vmware.com/files/pdf/perf-vsphere-memory_management.pdf
33. KSM http://www.linux-kvm.com/content/using-ksm-kernel-samepage-merging-kvm
34. “LRU, метод вытеснения из кэша” http://habrahabr.ru/post/136758/
35. BELADY, L.A. 1966. A study of replacement algorithms for virtual storage computers. IBM Syst. J. 5, 2, 78-101.
36. Johnson, Theodore, and Dennis Shasha. "X3: A Low Overhead High Performance Buffer Management Replacement Algorithm." (1994).
37. O'Neil, Elizabeth J.; O'Neil, Patrick E.; Weikum, Gerhard (1993). "The LRU-K Page Replacement Algorithm for Database Disk Buffering".Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data. SIGMOD '93 (New York, NY, USA: ACM): 297–306
38. https://en.wikipedia.org/wiki/Page_replacement_algorithm
39. Tanenbaum, Andrew (2009). Modern Operating Systems Third Edition. pp 209 – 210
40. Nicola, V. F., A. Dan, and D. M. Dins, "Analysis of the Generalized Clock Buffer Replacement Scheme for Database Transaction Processing", IBM Research Report RC 17225, Yorktown Heights, NY, Sept. 1991.

41. "CLOCK-Pro: An Effective Improvement of the CLOCK Replacement" by Song Jiang, Feng Chen, and Xiaodong Zhang, 2005
42. WSCLOCK—a simple and effective algorithm for virtual memory management" by Richard W. Carr and John L. Hennessy, 1981
43. WSClock http://www.cs.nyu.edu/courses/spring09/V22.0202-002/wsclock-davis.html
44. Bansal, Sorav and Modha, Dharmendra S. (2004). "CAR: Clock with Adaptive Replacement". In Proceedings of the USENIX Conference on File and Storage Technologies (FAST). pp. 187–200
45. Peter M. Chen and Brian D. Noble. 2001. “When Virtual Is Better Than Real”. In Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HOTOS '01). IEEE Computer Society, Washington, DC, USA
46. P. J. Denning. 1980. Working Sets Past and Present. IEEE Trans. Softw. Eng. 6, 1 (January 1980), 64-84.
47. Richard W. Carr and John L. Hennessy. 1981. WSCLOCK—a simple and effective algorithm for virtual memory management. In Proceedings of the eighth ACM symposium on Operating systems principles (SOSP '81). ACM, New York, NY, USA, 87-95
48. Intel® 64 and IA-32 Architectures Software Developer’s Manual Volume 3C: System Programming Guide, Part 3 http://www.intel.ru/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3c-part-3-manual.pdf
49. Weiming Zhao, Zhenlin Wang, and Yingwei Luo. 2009. «Dynamic memory balancing for virtual machines.» SIGOPS Oper. Syst. Rev. 43, 3 (July 2009), 37-47
50. Weiming Zhao, Xinxin Jin, Zhenlin Wang, Xiaolin Wang, Yingwei Luo, and Xiaoming Li. 2011. «Low cost working set size tracking.» In Proceedings of

the 2011 USENIX conference on USENIX annual technical conference (USENIXATC'11)

51. P. Lu and K. Shen. Virtual machine memory access tracing with hypervisor exclusive cache. In USENIX ATC'07, pages 1–15, 2007.
52. W. Zhao and Z. Wang. Dynamic memory balancing for virtual machines. In VEE'09, 2009.
53. Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. 2014. Adaptive Resource Provisioning for Virtualized Servers Using Kalman Filters. ACM Trans. Auton. Adapt. Syst. 9, 2, Article 10 (July 2014), 35 pages
54. Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. 2009. Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman filters. In Proceedings of the 6th international conference on Autonomic computing (ICAC '09). ACM, New York, NY, USA, 117-126
55. Kalyvianaki, Evangelia, and Steven Hand. "Applying Kalman filters to dynamic resource provisioning of virtualized server applications." Proc. 3rd Int. Workshop Feedback Control Implementation and Design in Computing Systems and Networks (FeBid). 2008
56. Nguyen, Hiep, et al. "Agile: Elastic distributed resource scaling for infrastructure-as-a-service." Proc. of the USENIX International Conference on Automated Computing (ICAC'13). San Jose, CA. 2013.
57. Shen, Zhiming, et al. "Cloudscale: elastic resource scaling for multi-tenant cloud systems." Proceedings of the 2nd ACM Symposium on Cloud Computing. ACM, 2011.
58. Wei Xu, Xiaoyun Zhu, Sharad Singhal, and Zhikui Wang. 2006. Predictive control for dynamic resource allo- cation in enterprise data centers. In Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS'06). 115–126.

59. Gerald Tesauro, Nicholas K Jong, Rajarshi Das, and Mohamed N. Bennani. 2007. On the use of hybrid reinforcement learning for autonomic resource allocation. Cluster Computing 10, 3 (2007), 287– 299.

60. Jib Kundu, Raju Rangaswami, Ajay Gulati, Ming Zhao, and Kaushik Dutta. 2012. Modeling vir- tualized applications using machine learning techniques. In Proceedings of the 8th ACM SIG- PLAN/SIGOPS Conference on Virtual Execution Environments (VEE'12). ACM Press, New York, 3–12.

61. Jing Xu, Ming Zhao, Jose Fortes, Robert Carpenter, and Mazin Yousif. 2007. On the use of fuzzy modeling in virtualized data center management. In Proceedings of the International Conference on Autonomic Computing (ICAC'07). IEEE Computer Society, Washington, DC, 25.

62. B. Guenter, N. Jain, and C. Williams. Managing cost, performance, and reliability tradeoffs for energy-aware server provisioning. In Proc. INFOCOM, 2011.

63. S. Govindan, J. Choi, and et al. Statistical profiling-based techniques for effective power provisioning in data centers. In Proc. Eurosys, 2009.

64. J. Wildstrom, P. Stone, and E. Witchel. CARVE: A cognitive agent for resource value estimation. In ICAC, pages 182–191. IEEE Computer Society, 2008

65. S. Kundu, R. Rangaswami, K. Dutta, and M. Zhao. Application Performance Modeling in a Virtualized Environment. In Proc. of IEEE HPCA, January 2010.

66. P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Ander- sen. Fingerprinting the datacenter: Automated classification of per- formance crises. In EuroSys '10 Proceedings of the 5th European conference on Computer systems, pages 111–124, 2010.

67. J. Rao, X. Bu, C.-Z. Xu, L. Y. Wang, and G. G. Yin. VCONF: a reinforcement learning approach to virtual machines auto-configuration. In ICAC, pages 137–146. ACM, 2009.
68. Peter J. Denning. The working set model for program behavior. Commun. ACM 11, 5 (May 1968)
69. Р. Максудова, патч на снижение фиктивно занятой памяти в случае OOM https://lkml.org/lkml/2014/10/15/340
70. Auto-ballooning проект http://www.linux-kvm.org/page/Projects/auto-ballooning
71. Auto-ballooning презентация на KVM forum http://www.linux-kvm.org/images/5/58/Kvm-forum-2013-automatic-ballooning.pdf
72. VirtIO новый стандарт http://docs.oasis-open.org/virtio/virtio/v1.0/cs03/virtio-v1.0-cs03.pdf
73. http://www.vfrank.org/2013/09/18/understanding-vmware-ballooning/
74. http://kb.vmware.com/selfservice/microsites/search.do?language=en_US&cmd=displayKC&externalId=1003586
75. Xen self-ballooning patch set by Magenheimer http://old-list-archives.xenproject.org/xen-devel/2008-04/msg00567.html
76. Chiang, Jui-Hao, Han-Lin Li, and Tzi-cker Chiueh. "Working Set-based Physical Memory Ballooning." ICAC. 2013.
77. Kim, Jinchun, et al. "Dynamic Memory Pressure Aware Ballooning." Memory 20: 25. http://memsys.io/wp-content/uploads/2015/09/p103-kim.pdf
78. https://charbelnemnom.com/2014/01/understanding-dynamic-memory-in-hyper-v-2012r2-part-1/
79. https://technet.microsoft.com/en-us/library/hh831766.aspx
80. Free Memory on Linux http://blog.scoutapp.com/articles/2010/10/06/determining-free-memory-on-linux

81. http://www.cyberciti.biz/faq/linux-check-memory-usage/
82. http://blogs.msdn.com/b/tims/archive/2010/10/29/pdc10-mysteries-of-windows-memory-management-revealed-part-two.aspx
83. ProcFS https://www.kernel.org/doc/Documentation/filesystems/proc.txt
84. ProcFS http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html
85. Consuming Performance Data https://msdn.microsoft.com/en-us/library/windows/desktop/aa371903(v=vs.85).aspx
86. ZwQueryInformationProcess https://msdn.microsoft.com/en-us/library/windows/desktop/ms687420(v=vs.85).aspx
87. ZwQuerySystemInformation https://msdn.microsoft.com/en-us/library/windows/desktop/ms725506(v=vs.85).aspx
88. Многомерная линейная регрессия http://www.machinelearning.ru/wiki/index.php?title=Многомерная_линейная_регрессия
89. Strijov, V. “The search of a parametric regression model in an inductive-generated set”, Journal of Computational Technologies. 2007. No 1. P. 93-102
90. MVR Composer http://www.machinelearning.ru/wiki/index.php?title=MVR_Composer
91. Matlab http://www.mathworks.com/products/matlab/
92. “Redefining Server Performance Characterization for Virtualization Benchmarking”, Intel Technology Journal, Volume 10, Issue 03, Published August 10, 2006, 243-252
93. SPECjbb2005 http://www.spec.org/jbb2005/
94. SysBench http://sysbench.sourceforge.net/
95. WebBench http://cs.uccs.edu/~cs526/webbench/webbench.htm
96. Leo Breiman “Statistical Modeling: The Two Cultures”, Statistical Science 2001, Vol. 16, No. 3, 199–231

97. Критерий однородности http://www.machinelearning.ru/wiki/index.php?title=Критерии_однородности

98. Орлов, А.И. Прикладная статистика. Учебник/ А.И. Орлов - М.:Экзамен, 2004. - 656с.

99. Лемешко, Б.Ю. Мощность критериев согласия при близких альтернативах [Текст]/ Б.Ю. Лемешко, С.Б. Лемешко, С.Н. Постовалов// Измерительная техника. -2007. -No2. -С.22-27.

100. Проверка статистических гипотез http://www.machinelearning.ru/wii/index.php?title=Проверка_статистических_гипотез

101. Кобзарь, А.И. Прикладная математическая статистика. Для инженеров и научных работников/А.И. Кобзарь -М.:ФИЗМАТЛИТ, 2006 -816с

102. The R Project for Statistical Computing [Электронный ресурс] - Режим доступа: http://www.r-project.org/.

103. kSamples: K-Sample Rank Tests and their Combinations [Электронный ресурс] - Режим доступа: http://cran.r-project.org/web/packages/kSamples/index.html.

104. Sigel-Tukey: a Non-parametric test for equality in variability (R code) [Электронный ресурс] - Режим доступа: http://www.r-statistics.com/2010/02/siegel-tukey-a-non-parametric-test-for-equality-in-variability-r-code/.

105. Comparison Between Windows 7 and Windows 8 Memory Management System. http://www.askvg.com/comparison-between-windows-7-and-windows-8-memory-management-system/

106. PCMark http://www.futuremark.com/benchmarks/pcmark

107. File Cache Performance and tuning https://msdn.microsoft.com/en-us/library/bb742613.aspx

108. Things to consider before you enable System Cache mode in Windows XP https://support.microsoft.com/en-us/kb/895932

109. VirtIO http://www.linux-kvm.org/page/Virtio

110. VMware success stories (server consolidation case) http://www.vmware.com/a/customers/solution/1

Приложение А. Реализация алгоритма корректировки оценки

```
#ifndef PAGES_PER_MB
#define PAGES_PER_MB (SIZE_1MB/PAGE_SIZE)
#endif

/*
 * Balloon size = RAM - WS - GAP. Gap is an emprical value found by RL
means.
 * gap change price is the static array to estimate the best change of the
gap (+ or -)
 * gap change is the small array since balloon is inertial system and it
is not recommended to
 * inflate it or deflate for more then 128MB per second.
 */
#define GAP_ALIGN 4 /* common memory alignment */
#define GAP_CHANGE_MAX (128)
#define GAP_CHANGE_ARRAY (GAP_CHANGE_MAX/GAP_ALIGN*2+1)
#define GAP_TO_IDX(gap) (( (gap)+(int)GAP_CHANGE_MAX)/GAP_ALIGN)
#define GAP_FROM_IDX(i) ((int)(i)*GAP_ALIGN - (int)GAP_CHANGE_MAX)
static INT gapChangePrice[ GAP_CHANGE_ARRAY ];

UINT64 rand64()
{
    UINT64 rand;
    unsigned char ok = 0;

     asm volatile ("rdrand %0; setc %1"
          : "=r" (rand), "=qm" (ok));

    if( !ok )
     WRITE_TRACE( DBG_FATAL, "Failed to use rand!" );

    return rand;
```

```
}


/**
 * Put a fine or a prize for the previous change of the gap
 */
void FineFunction( INT fined_change )
{
    static UINT prev_pagein = 1000; /* initial value for starting delta */
    static UINT prev_io = 100; /* initial value for starting delta */
    UINT i = GAP_TO_IDX(fined_change);

    /* fines */
    static UINT io_fine = SfGetUINT32("kernel.balloon.gap_io_fine",15);
    static UINT pagein_fine =
SfGetUINT32("kernel.balloon.gap_pagein_fine",50);
    static UINT positive_prize =
SfGetUINT32("kernel.balloon.gap_positive_prize",5);

    UINT cur_pagein = MPERF_COUNT_GET(vmm_pagein);
    UINT cur_io = 0;
    UINT vcpu;
    for_each_vcpu( vcpu )
     cur_io += MPERF_COUNT_VCPU_GET(vcpu, vmm_io_op);

    if( i > GAP_CHANGE_ARRAY )
     Abort("Fine on gap outside interval %u(%d) > %u", i, fined_change,
GAP_CHANGE_ARRAY );

    if( (cur_io & 0xf) > (prev_io & 0xf) )
     gapChangePrice[ i ] -= io_fine;
    if( (cur_pagein & 0x3f) > (prev_pagein & 0x3f) )
     gapChangePrice[ i ] -= pagein_fine;
    if( cur_io <= prev_io && cur_pagein <= prev_pagein )
     gapChangePrice[ i ] += positive_prize;

    prev_io = cur_io;
    prev_pagein = cur_pagein;
}

/**
 * Choose gap in dependence on working set size
 */
UINT PciBalloonChooseGap(UINT ws)
{
    static INT prev_gap = 16; // 16MB is some empirical initial value. It
doesn't influence the final result
    static INT prev_gap_change = 0;
    static UINT prev_ws = 0;
    INT gap = 0;
    #define MIN_GAP 4
    /* keep gap aligned > MIN_GAP and < ws/2 */
    #define ALIGN_GAP(gap,ws) \
```

```
     MAX( (MIN( (INT)gap, (INT)(ws)/2 ) & ~(GAP_ALIGN-1)), MIN_GAP )

    /* do not overwrite forced value */
    if( (gap = (INT)SfGetUINT32( "kernel.balloon.gap", 0 )) )
     return gap;

    /* estimate the goodness of the previous gap */
    FineFunction( prev_gap_change );

    /* the degree of randomness. epsilon-greedy algorithm.
     each "greedy_degree" time the random result will be chosen
         0 means no training (100% greediness);
     1 means randomness
         2 is less random, etc
     think of "greedy_degree" as epsilon=100-100/greedy_degree
          (10 is 90% of greediness, 5 is 80%, 20 is 95%)*/
    static UINT greedy_degree = SfGetUINT32("kernel.balloon.gap_greedy",
15 );
    static UINT iter = 0;
    if( greedy_degree && (++iter)%greedy_degree == 0 )
    {
     gap = gap + char(rand64()%GAP_CHANGE_MAX);
     gap = ALIGN_GAP(gap, ws);
    }
    else
    {
     /* if we found a better solution, we avoid changing gap before
tolerance threshold is exceeded)*/
     UINT gap_change_threshold =
SfGetUINT32("kernel.balloon.gap_tolerance", 20 );

     /* we don't need too often changes. gap allows to compensate ws
fluctuation */
     gap = ALIGN_GAP( ((INT)ws - (INT)prev_ws + prev_gap), ws );

     /* keep gap change within (-GAP_CHANE_MAX;+GAP_CHANGE_MAX) limits */
     INT gap_change = MAX(MIN((gap - prev_gap), GAP_CHANGE_MAX),-
GAP_CHANGE_MAX);
     UINT gap_change_i = GAP_TO_IDX(gap_change);

     for( UINT i = 0; i < GAP_CHANGE_ARRAY; i++ )
     {
          if( gapChangePrice[i] > gapChangePrice[gap_change_i] &&
               (gapChangePrice[i] - gapChangePrice[gap_change_i]) >=
gap_change_threshold )
               gap_change_i = i;
     }

     if( GAP_FROM_IDX(gap_change_i) != gap_change )
     {
          /* keep gap below ws/2 */
          gap = ALIGN_GAP( (prev_gap + (INT)GAP_FROM_IDX(gap_change_i)),
ws );
```

```
      }
    }
    prev_gap_change = prev_gap - gap;
    prev_gap = gap;
    prev_ws = ws;
    return (UINT)gap;
}

void PciBalloonUpdateGuestUsed( UINT val )
{
    UINT gap;

    MPERF_COUNT_SET( vmm_guest_used, val );
    if( balloon_ctl->balloon_mode != BALLOON_GUEST_USED )
      return;

    /* align ws in pages at 4MB. this is default alignment of memory size
*/
    val &= ~(PAGES_PER_MB * GAP_ALIGN - 1);

    /* calculate the best fit size of balloon */
    gap = PciBalloonChooseGap( val/PAGES_PER_MB );
    UINT32 best_fit_sz = MonState.GuestPhyMem.uRamPages - val - gap;

    if( val > MonState.GuestPhyMem.uRamPages - gap )
      Abort("PANIC@215.83(%x,%x,%x)", MonState.GuestPhyMem.uRamPages, val,
gap );

    AtomicWrite( (UINT*)&balloon_ctl->balloon_target_size, best_fit_sz );
    UINT old = balloon_dev.state.target_pages;
    AtomicWrite( &balloon_dev.state.target_pages, best_fit_sz );

    MPERF_COUNT_SET( balloon_target, best_fit_sz );

    WRITE_TRACE( DBG_INFO, "[Balloon] set balloon guest used 0x%x ->
0x%x(0x%x), gap=%d", old, best_fit_sz, val, gap );
}
```

Приложение Б. Порядок проведения исследовательских испытаний на тестовом стенде ООО Акронис

Б.1. Планирование эксперимента

Представляемое планирование эксперимента основано на методах испытаний, которые являются актуальными разработками и воплощают в себе проверенные многолетней практикой организационно-технические решения компании Акронис по автоматизированному тестированию датацентров.

Планирование эксперимента определяется режимом работы используемого стенда и исследуемым типом форсированного сбоя комплексов СХД. Режим работы и форсирование сбоя осуществляются в автоматизированном режиме выделенным узлом СХД — оркестрантом, отвечающим за выполнение испытаний.

Функциональные возможности программного обеспечения должны обеспечивать масштабируемость процессов имитационного моделирования СХД, в том числе предусматривать возможность организации множества виртуальных узлов на одном физическом узле.

Проводимые опыты собраны в серии, каждая из которых описывается факторами, определяющими тип схемы хранения. Далее, опыты в каждой серии сгруппированы по преобладающему типу сбоя, однако в каждой отдельной группе опытов в силу стохастической природы возникновения ошибок возможны сбои других типов. Уровни факторов должны определять различные K/N схемы хранения.

Б.2. План эксперимента

Эксперимент должен состоять из набора тестов, позволяющих выявить отклики аппаратно-программного обеспечения на типовые факторы отказов. Ряд представленных тестов необходимо выполнять в перемежающейся циклической последовательности, поскольку результаты опытов одного теста являются

условиями проведения опытов другого теста. Ход эксперимента схематично показан на рис. Б.1.

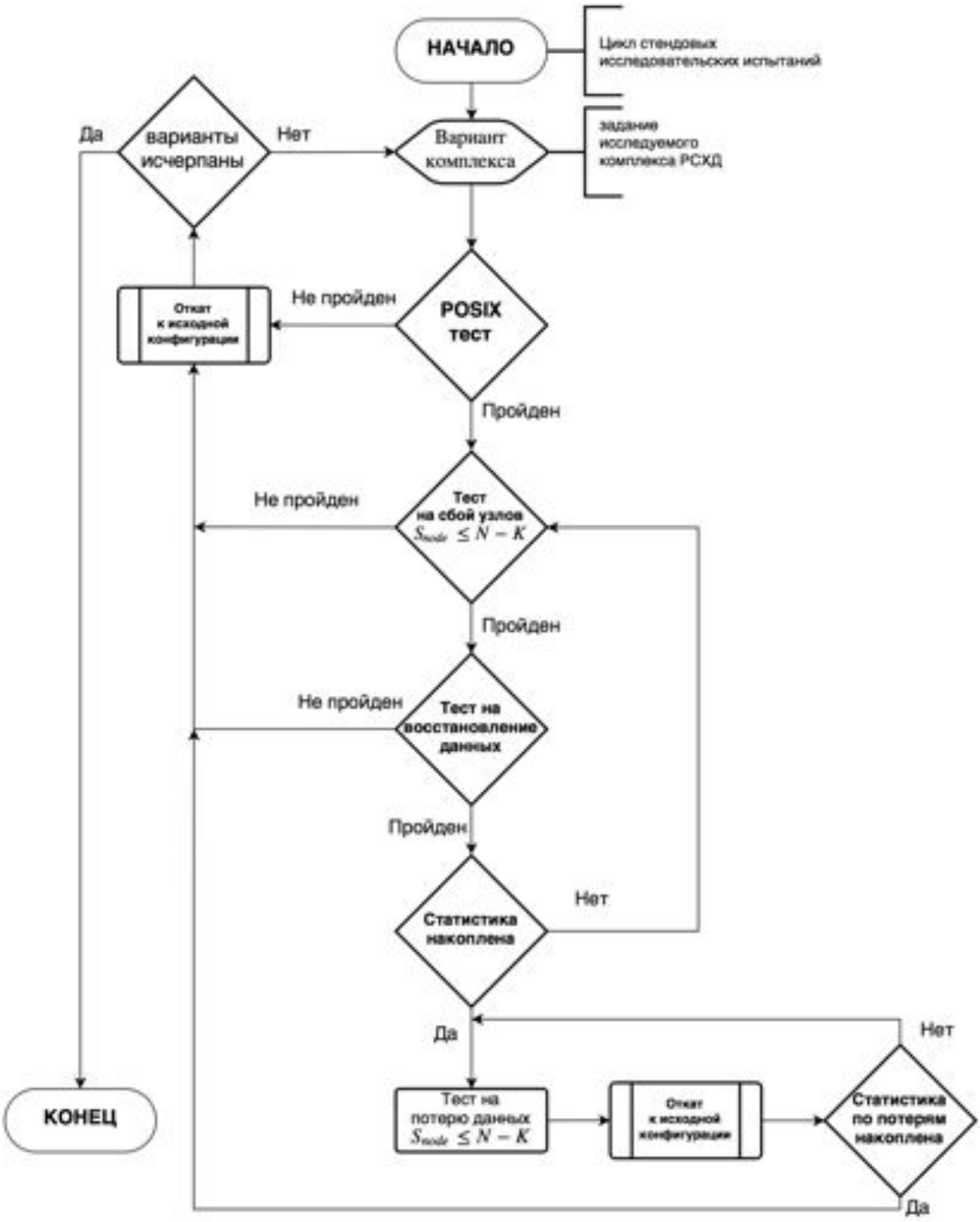


Рис.Б.1. Схема проведения серии экспериментов

Б.2.1. Функциональный тест

Функциональный тест должен осуществляться по протоколу POSIX Opensource test for posix compatibility (http://posixtest.sourceforge.net/) или эквивалентному ему. В ходе опытов испытательный стенд будет подвергаться штатной типовой нагрузке без форсированных сбоев. Серия опытов будет определяться следующими факторами и откликами.

Фактор: ω_{vol} — доля заполнения исходного объема, доступного используемому программному обеспечению. Размах варьирования фактора составляет 0.75, ($\omega_{vol} \in [0.05; 0.8]$).

Отклики представляют индикаторы posix-совместимости и отзывчивость хранилища на доступ к данным:

- I_{posix} — индикатор posix-совместимости, множество принимаемых значений $\{0; 1\}$ (невыполнение/выполнение);
- t_{read}, [с] — время чтения данных при установившейся скорости чтения;
- t_{write} [с] — время записи данных при установившейся скорости записи.

Ожидаемые отклики:

- выполнение posix-совместимости, $I_{posix} = 1$ для любых значений фактора $\forall\omega_{vol} \in [0.05; 0.8]$.

Ход проведения теста на испытательном стенде схематически приведен на рис. Б.2.

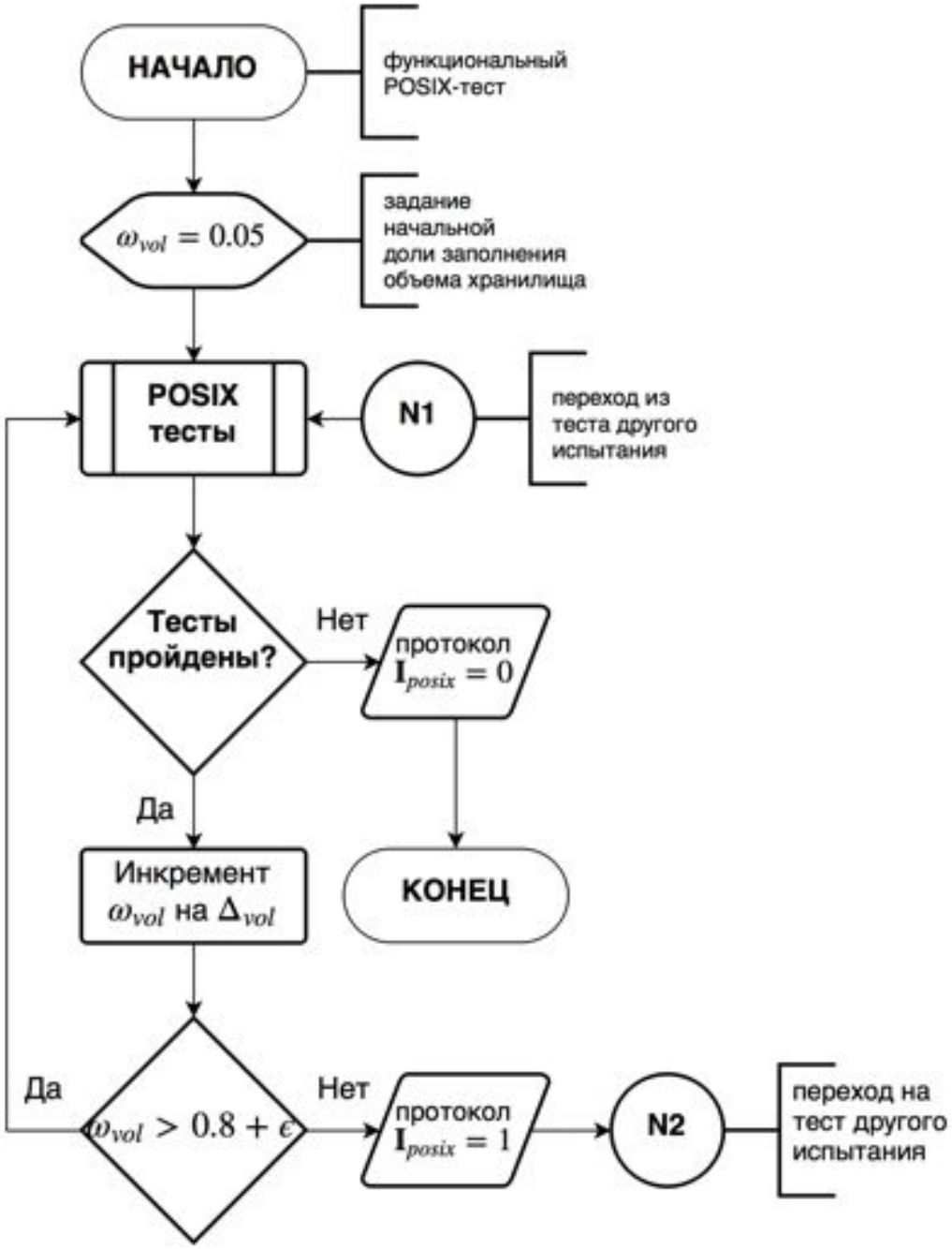


Рис. Б.2. Блок-схема проведения функционального теста на испытательном стенде

В.2.2.Тест на доступность данных без превышения лимита по количеству отказавших узлов

Тест на доступность данных в режимах запись/чтение осуществляется без превышения теоретического лимита по восстановимым потерям для узлов. Начальными состояниями данного теста являются итоговые состояния испытательного стенда и управляющего ПО при завершении опыта с фактором

$\omega_{vol} = 0.8$ из теста в п. Б.2.1. В ходе опытов испытательный стенд подвергается типовой нагрузке, тождественной функциональному тесту из п. Б.2.1. Серия опытов определяется следующими факторами и откликами.

Факторы:

- S_{node} — количество форсированно отказавших узлов хранения (до достижения лимита по потерям). Размах варьирования фактора равен $(N - K - 1)$: $S_{node} = 1, 2, \dots, (N - K)$; Выбор конкретных узлов для имитации форсированного отказа выполняется псевдослучайным образом в случайные моменты времени.
- Для данного теста доля заполнения исходного объема, доступного используемому программному обеспечению, фиксируется на уровне $\omega_{vol} = 0.8$.

Отклики представляют измеряемые потери данных и отзывчивость хранилища на доступ к данным:

- I_{loss} — индикатор потерь данных, множество принимаемых значений $\{0; 1\}$;
- t_{read}, [с] — время чтения данных при установившейся скорости чтения;
- t_{write} [с] — время записи данных при установившейся скорости записи.

Ожидаемые отклики:

- нулевой уровень потери данных I_{loss} для любых значений фактора S_{node}: $\mathrm{I}_{loss} \equiv 0, \forall S_{node} \in [1; (N - K)]$.

Каждый опыт считается успешно выполненным при осуществлении форсированного отказа согласно установленному фактору S_{node}.

Ход проведения теста на испытательном стенде схематически приведен на рис. Б.3.

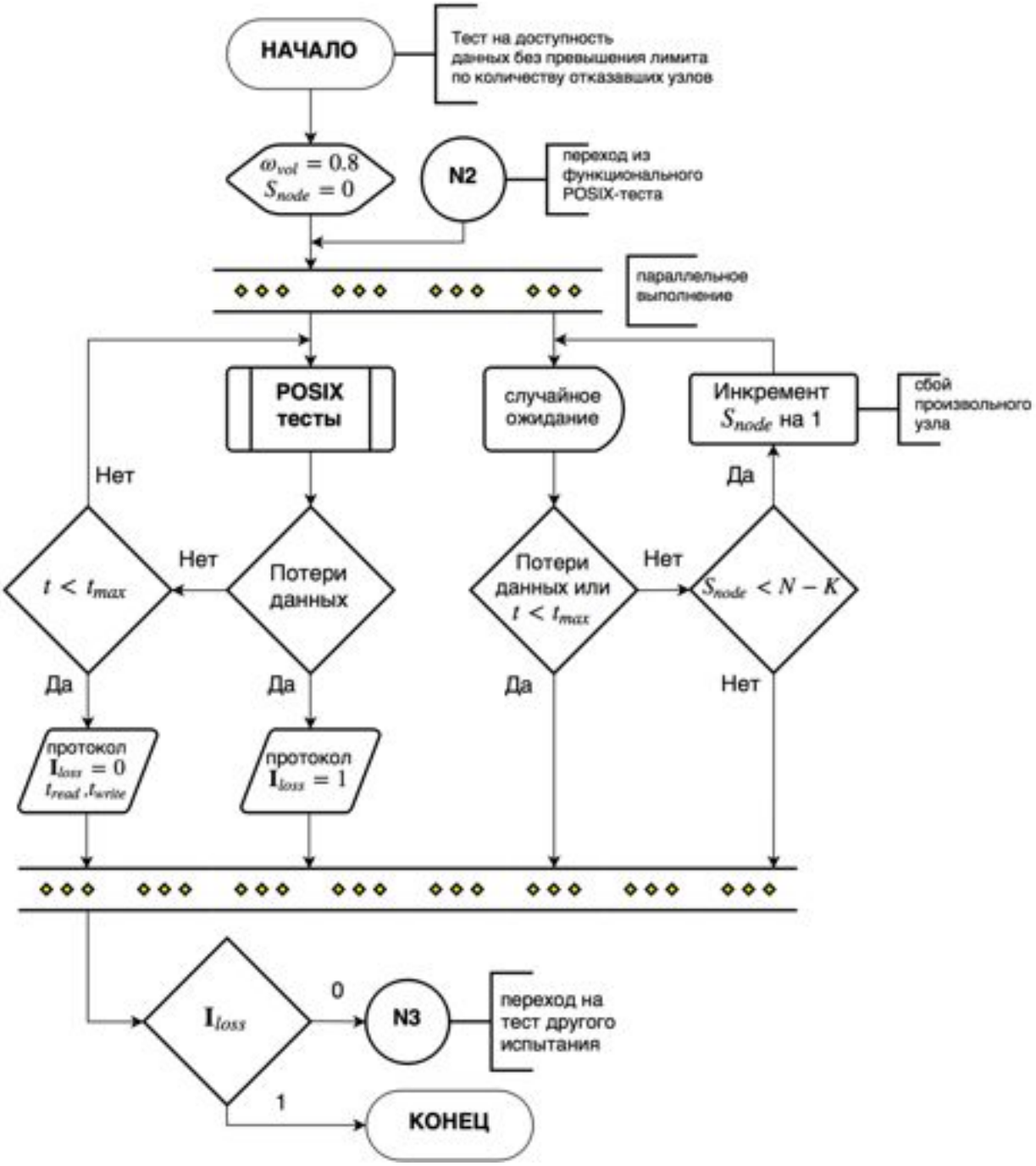


Рис. Б.3. Блок-схема проведения теста на доступность данных без превышения лимита по количеству отказавших узлов

Б.2.3. Тест по восстановлению данных после форсированного отказа узлов без превышения лимита

Начальными состояниями теста на восстанавливаемость данных являются итоговые состояния испытательного стенда и управляющего ПО при завершении опыта из теста в п. Б.2.2 В ходе опытов на испытательном стенде выполняется

поиск испорченных данных после форсированного отказа узла, включая сбой перезаписи. Серия опытов определяется следующими факторами и откликами.

Факторы:

- Для данного теста доля заполнения исходного объема, доступного используемому системному ПО, и количество отказавших узлов S_{node} полностью определяются результатами опыта из п. Б.2.2.

Отклики представляют измеряемые потери данных и отзывчивость хранилища на доступ к данным:

- $\mathrm{I}_{restore}$ — индикатор восстановления данных, множество принимаемых значений $\{0; 1\}$;
- t_{before}, [с] — время до начала восстановления данных;
- $t_{restore}$ [с] — полное время восстановления данных.

Ожидаемые отклики:

- полное восстановление данных $\mathrm{I}_{restore} = 1$.

Каждый опыт считается успешно выполненным при достижении полного восстановления данных после форсированного отказа с фактором S_{node}.

Ход проведения теста на испытательном стенде схематически приведен на рис. Б.4.

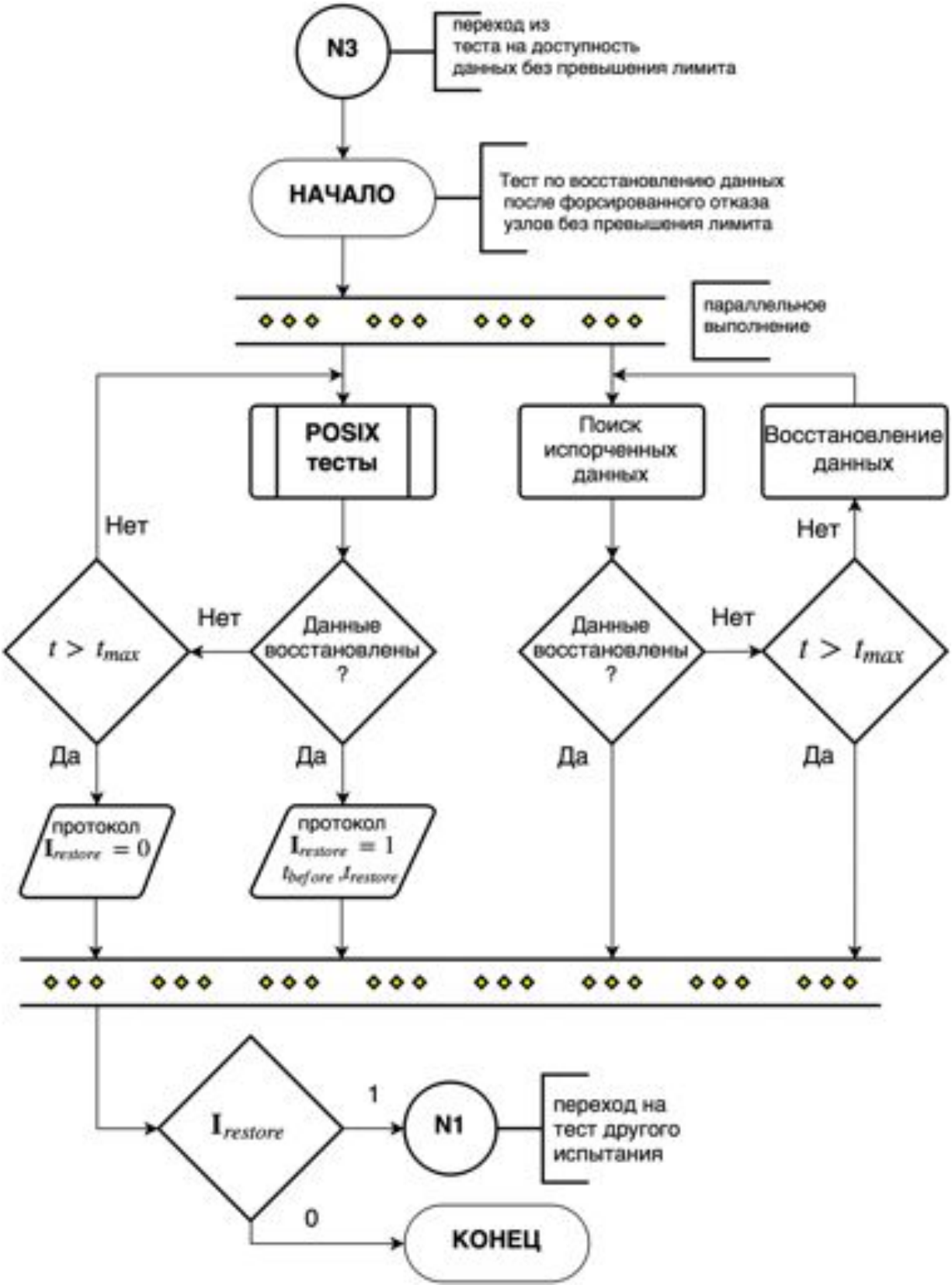


Рис. Б.4. Блок-схема проведения теста по восстановлению данных после форсированного отказа узлов без превышения лимита

Б.2.4. Тест на уровень потерь данных при превышении лимита по количеству отказавших узлов

Тест на уровень потерь данных осуществляется при превышении на единицу теоретического лимита по количеству потерь для узлов. В ходе опытов испытательный стенд подвергается типовой нагрузке, тождественной функциональному тесту из п. Б.2.1. Серия опытов определяется следующими факторами и откликами.

Факторы:

- избыточность хранения $\vartheta_{node} = (N - K)/K$, задаётся K/N схемой хранения;
- Для данного теста количество форсированно отказавших узлов хранения фиксируется на уровне $S_{node} = N - K + 1$.

Отклики представляют измеряемые потери данных:

- V_{loss} — чистый объем потери данных, принимает значения в диапазоне от 0 байт до полного объема хранимых данных;
- $H = (h_1, h_2, h_3, h_4)^{\mathrm{T}}$ — гистограмма количества h_i невосстановимо утраченных файлов с типовыми диапазонами размеров: 1 КБ — 1 МБ, 1 МБ — 10 МБ, 10 МБ — 100 МБ, 100 МБ — 1 ГБ;
- ω_{loss} — процент потерь от общего объема данных, множество принимаемых значений — диапазон рациональных чисел $[0; 1]$.

Ожидаемый отклик: снижение уровня потерь данных при увеличении фактора избыточности хранения. Ход проведения теста на испытательном стенде схематически приведен на рис. Б.5.

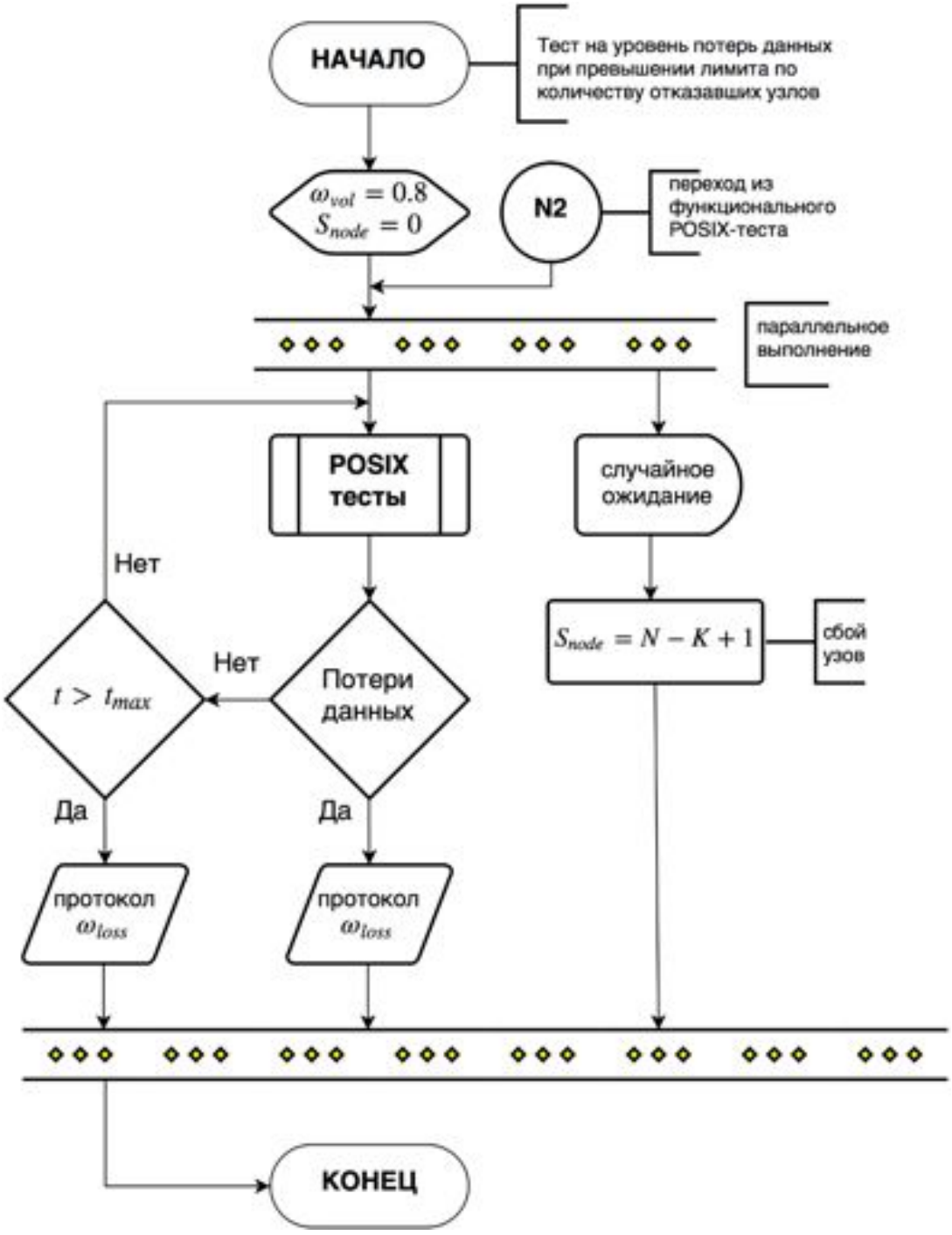


Рис. Б.5. Блок-схема проведения теста на уровень потерь данных при превышении лимита по количеству отказавших узлов

Б.2.5. Тест на доступность данных без превышения лимита по количеству отказавших дисков

Тест на доступность данных в режимах запись/чтение осуществляется без превышения теоретического лимита по восстановимым потерям для дисков, отказавших в разных узлах. В ходе опытов испытательный стенд подвергается

типовой нагрузке, тождественной функциональному тесту из п. Б.2.1. Серия опытов определяется следующими факторами и откликами.

Факторы:

- S_{disk} — количество форсированно отказавших дисков хранения в разных узлах (до достижения лимита по потерям). Размах варьирования фактора равен $(N - K - 1)$: $S_{disk} = 1, 2, \dots, (N - K)$;
- Для данного теста доля заполнения исходного объема, доступного используемому системному ПО, фиксируется на уровне $\omega_{vol} = 0.8$.

Отклики представляют измеряемые потери данных и отзывчивость хранилища на доступ к данным:

- I_{loss} — индикатор потерь данных, множество принимаемых значений $\{0; 1\}$;
- t_{read}, [с] — время чтения данных при установившейся скорости чтения;
- t_{write} [с] — время записи данных при установившейся скорости записи.

Ожидаемые отклики:

- нулевая потеря данных I_{loss} для любых значений фактора S_{disk}: $\mathrm{I}_{loss} \equiv 0$, $\forall S_{disk} \in [1; (N - K)]$.

Ожидаемые отклики:

- нулевая потеря данных I_{loss} для любых значений фактора S_{disk}: $\mathrm{I}_{loss} \equiv 0$, $\forall S_{disk} \in [1; (N - K)]$.

Ход проведения теста на испытательном стенде схематически приведен на рис. Б.6.

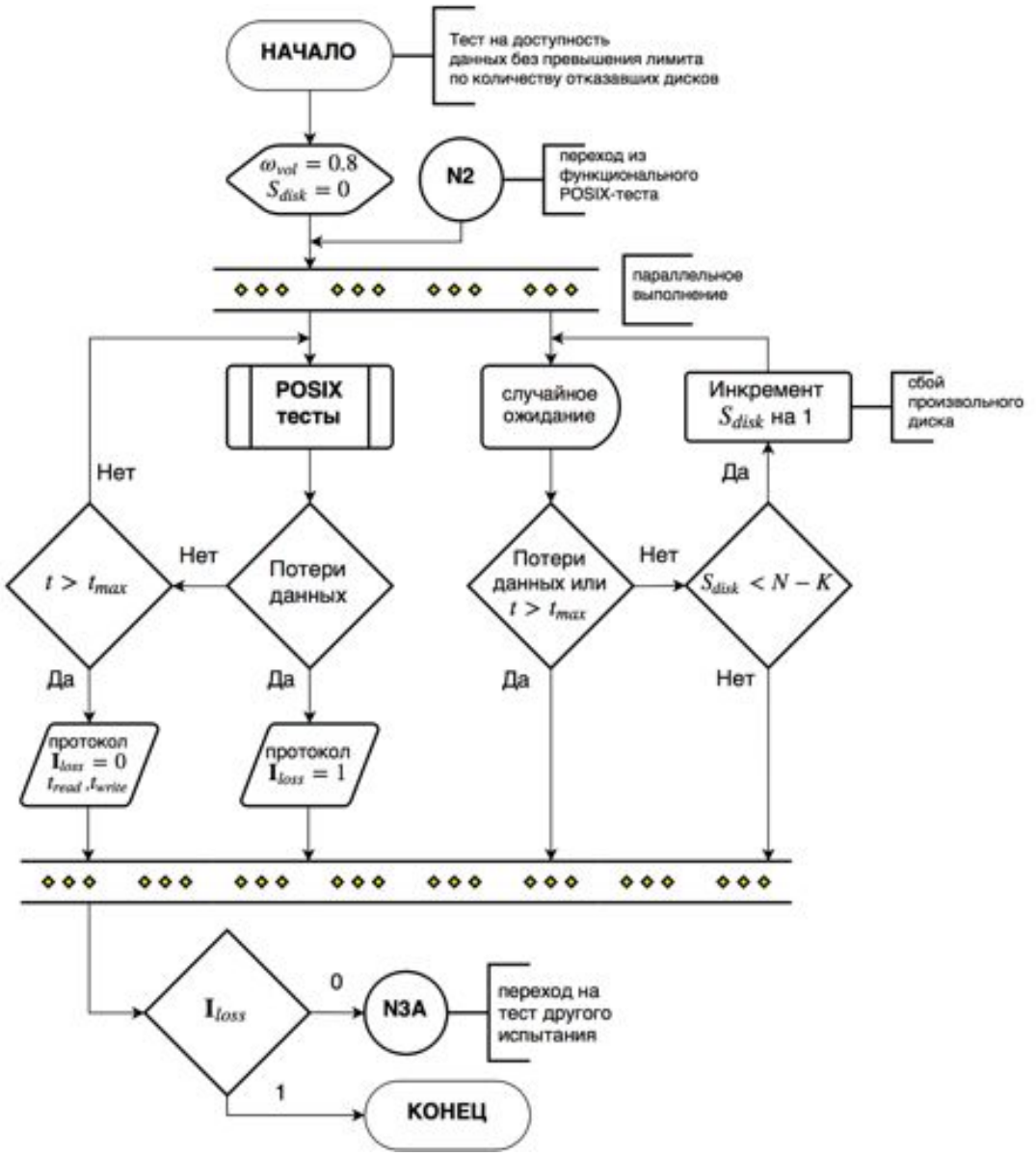


Рис. Б.6. Блок-схема проведения теста на доступность данных без превышения лимита по количеству отказавших дисков

Б.2.6. Тест по восстановлению данных после форсированного отказа дисков без превышения лимита

Начальными состояниями теста на восстановимость данных являются итоговые состояния испытательного стенда и управляющего ПО при завершении опыта из теста в п. Б.2.5. В ходе опытов на испытательном стенде выполняется поиск испорченных данных после форсированного отказа диска, включая сбой перезаписи. Серия опытов определяется следующими факторами и откликами.

Факторы:

- Для данного теста доля заполнения исходного объема, доступного используемому системному ПО, и количество отказавших дисков S_{disk} полностью определяются результатами опыта из п. Б.2.5.

Отклики представляют измеряемые потери данных и отзывчивость хранилища на доступ к данным:

- $\mathrm{I}_{restore}$ — индикатор восстановления данных, множество принимаемых значений $\{0; 1\}$;
- t_{before}, [с] — время до начала восстановления данных;
- $t_{restore}$ [с] — полное время восстановления данных.

Ожидаемые отклики:

- Полное восстановление данных $\mathrm{I}_{restore} = 1$.

Каждый опыт считается успешно выполненным при достижении полного восстановления данных после форсированного отказа с фактором S_{disk}. Ход проведения теста на испытательном стенде схематически приведен на рис. Б.7.

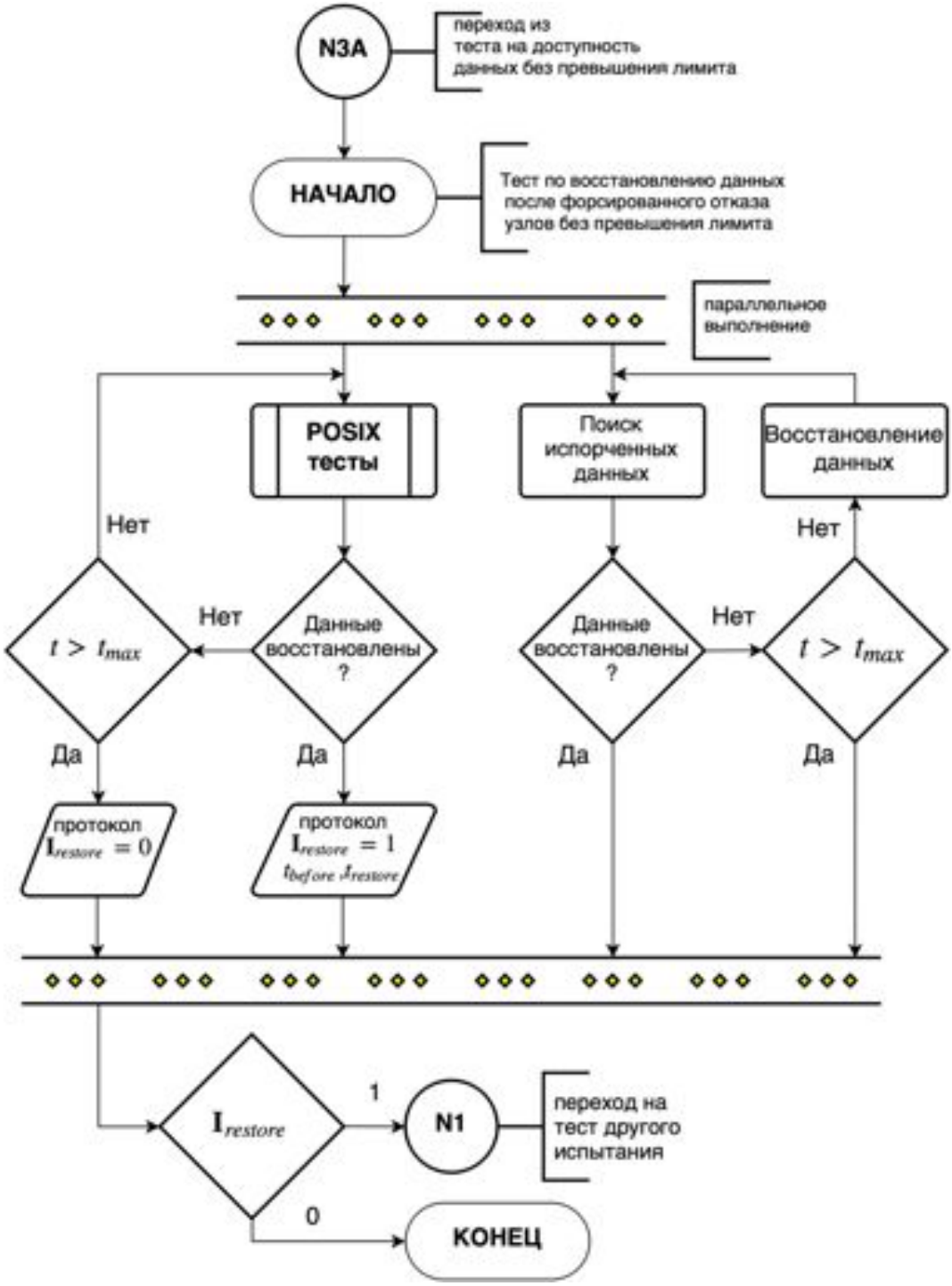


Рис. Б.7. Блок-схема проведения теста по восстановлению данных после форсированного отказа дисков без превышения лимита

Б.2.7. Тест на уровень потерь данных при превышении лимита по количеству отказавших дисков

Тест на уровень потерь данных осуществляется при превышении на единицу теоретического лимита по количеству потерь для дисков. В ходе опытов

испытательный стенд подвергается типовой нагрузке, тождественной функциональному тесту из п. Б.2.1. Серия опытов определяется следующими факторами и откликами.

Факторы:

- избыточность хранения $\vartheta_{disk} = (N - K)/K$, задаётся K/N схемой хранения;
- Для данного теста количество форсированно отказавших узлов хранения фиксируется на уровне $S_{disk} = N - K + 1$.

Отклики представляют измеряемые потери данных:

- V_{loss} — чистый объем потери данных, принимает значения в диапазоне от 0 байт до полного объема хранимых данных;
- $H = (h_1, h_2, h_3, h_4)^{\mathrm{T}}$ — гистограмма количества h_i невосстановимо утраченных файлов с типовыми диапазонами размеров: 1 КБ — 1 МБ, 1 МБ — 10 МБ, 10 МБ — 100 МБ, 100 МБ — 1 ГБ;
- ω_{loss} — процент потерь от общего объема данных, множество принимаемых значений — диапазон рациональных чисел $[0; 1]$.

Ожидаемый отклик: снижение уровня потерь данных при увеличении фактора избыточности хранения. Ход проведения теста на испытательном стенде схематически приведен на рис. Б.8.

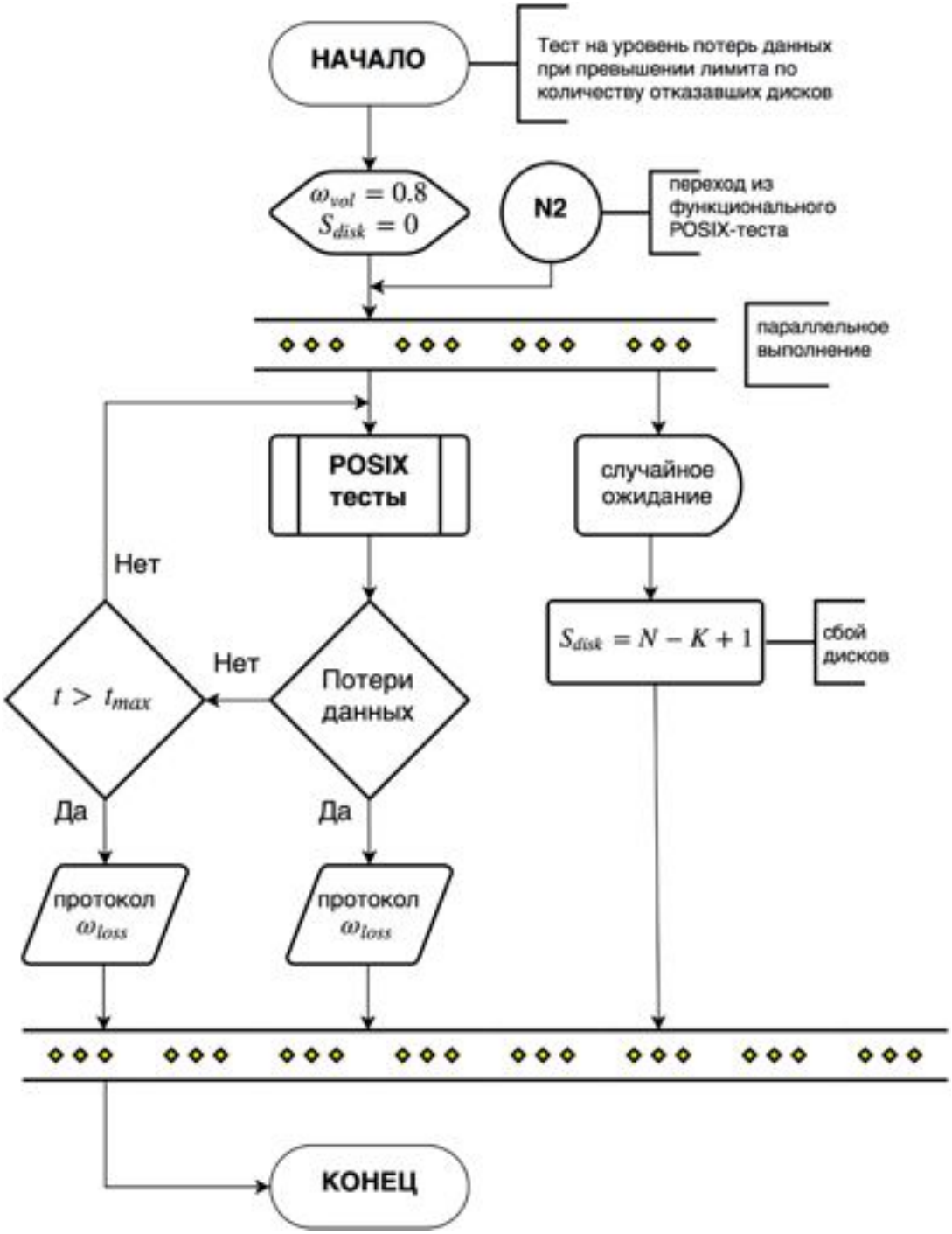


Рис. Б.8. Блок-схема проведения теста на уровень потерь данных при превышении лимита по количеству отказавших дисков

Б.2.8.Тест на доступность данных при сетевых сбоях

Тест на доступность данных в режимах запись/чтение осуществляется при сетевых сбоях заданной временной продолжительностью. В ходе опытов испытательный стенд подвергается типовой нагрузке, тождественной

функциональному тесту из п. Б.2.1. Серия опытов определяется следующими факторами и откликами.

Факторы:

- t_{delay}, [с] — интервал времени форсированного отказа единичного сегмента сети.
- Для данного теста доля заполнения исходного объема, доступного используемому системному ПО, фиксируется на уровне $\omega_{vol} = 0.8$.

Отклики представляют показатель целостности кластера:

- I_{excess}, — индикатор выполнения избыточного восстановления.

Ожидаемые отклики:

- отсутствие избыточных восстановлений $\mathrm{I}_{excess} \equiv 0$.

Ход проведения теста на испытательном стенде схематически приведен на рис. Б.9.

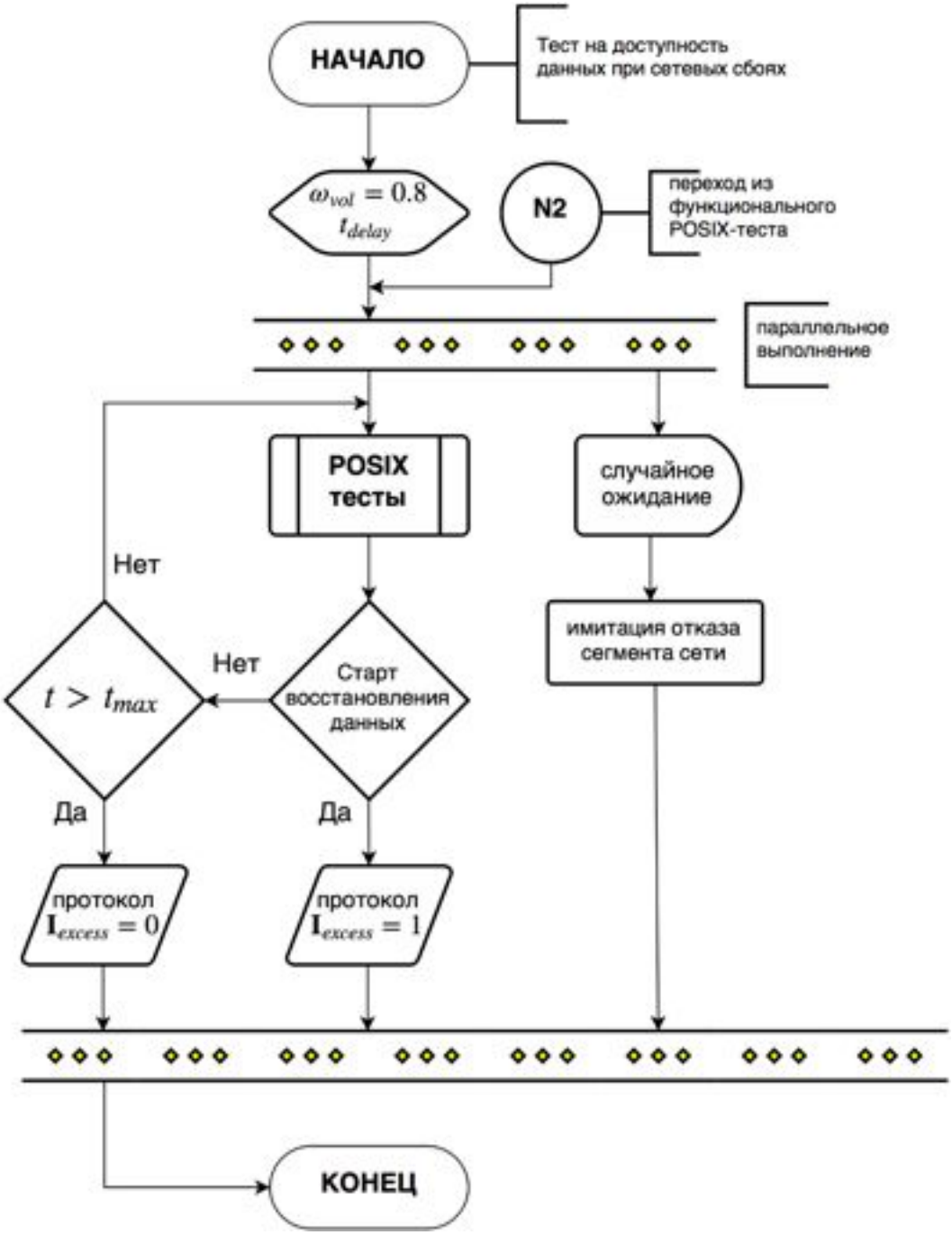


Рис. Б.9. Блок-схема проведения теста на доступность данных при сетевых сбоях