Моделирование и анализ данных
2025, *15*(4), 58—70.
https://doi.org/10.17759/mda.2025150404
ISSN: 2219-3758 (печатный)
ISSN: 2311-9454 (online)

Modelling and Data Analysis
2025, *15*(4), 58—70.
https://doi.org/10.17759/mda.2025150404
ISSN: 2219-3758 (print)
ISSN: 2311-9454 (online)

Научная статья | Original paper

УДК 004.451

Разработка тестов создания виртуальных машин с различными параметрами виртуального процессора

О.Р. Лисовский
Московский авиационный институт (национальный исследовательский университет)
Москва, Российская Федерация
✉ Lisoleg555@yandex.ru

Резюме

Контекст и актуальность. Гипервизоры являются основой современных облачных сервисов. В этой области активно разрабатываются новые решения, которые требуют тщательного тестирования. Особенно важна стабильная работа ключевого компонента — виртуальной машины. Из-за обилия настроек тестирование может занять продолжительный период времени, поэтому требуется не только максимальное тестовое покрытие, но и минимальное количество тестовых сценариев. **Цель.** Выявить и показать независимость минимального количества тестов для проверки создания виртуальной машины с различными параметрами виртуального процессора. **Гипотеза.** Количество независимых тестов, необходимое для проведения исчерпывающего тестирования создания виртуальной машины с разными параметрами процессора строго ограничено, и эти ограничения можно выявить с помощью классов эквивалентности. **Методы и материалы.** Рассматривались все основные параметры настройки виртуального процессора при создании виртуальной машины. Для каждого из них подбирались оптимальные классы эквивалентности. Далее брались тесты по граничным значениям классов и по одному тесту из середины класса. После выявления всех уникальных тестов для рассматриваемого параметра проводился поиск пересечений с ранее выявленными тестами, а дубликаты вычёркивались из конечного списка. Отдельно помечались тесты, которые могут отсутствовать из-за особенностей реализации гипервизора. **Результаты.** Результатом является набор тестов с одним параметром, меняющимся в зависимости от количества процессоров, которое может эмулировать гипервизор. Все тесты независимы, что позволяет производить их параллельный запуск. **Выводы.** Показано, что для одного из разделов тестирования гипервизора — создания виртуальной машины с различными параметрами виртуального процессора — можно при помощи классов эквивалентности вывести строгий набор независимых тестов. Рекомендуется таким же образом разработать тесты для прочих параметров создаваемой виртуальной машины.

Ключевые слова: тестирование, виртуальный процессор, классы эквивалентности, гипервизор, оптимизация

Для цитирования: Лисовский, О.Р. (2025). Разработка тестов создания виртуальных машин с различными параметрами виртуального процессора. *Моделирование и анализ данных, 15*(4)*,* 58—70. https://doi.org/10.17759/mda.2025150404

Development tests of creating virtual machines with different virtual processor parameters

O.R. Lisovskiy
Moscow Aviation Institute (National Research University)
Moscow, Russian Federation
✉ Lisoleg555@yandex.ru

Abstract

Context and relevance. Hypervisors are the foundation of modern cloud services. New solutions are being actively developed in this area and require thorough testing. Due to the abundance of settings, testing can take a long period of time, so not only maximum test coverage but also a minimum number of test scenarios is required. **Objective** Identify and demonstrate the independence of the minimum number of tests for verifying the creation of virtual machine with different virtual processor parameters. **Hypothesis.** The number of independent tests required to perform exhaustive testing of virtual machine creation with different processor parameters is strictly limited, and these limitations can be identified using equivalence classes. **Methods and materials.** All key virtual processor configuration parameters of created virtual machine were considered. Optimal equivalence classes were selected for each. Then, tests were taken at the boundary values of each class, along with one test from the middle of each class. After identifying all unique tests for the parameter in question, a search was conducted for intersections with previously identified tests, and duplicates were removed from the final list. Tests that may be missing due to specific hypervisor implementation features were marked separately. **Results.** The result is a set of tests with a single parameter that varies depending on the number of processors the hypervisor can emulate. All tests are independent, allowing them to be run in parallel. **Conclusions.** It has been shown that for one section of hypervisor testing — creating a virtual machine with various virtual processor parameters — it is possible to derive a clear set of independent tests using equivalence classes.

It is recommended to develop tests for other parameters of creating virtual machine in a similar way.

Keywords: testing, virtual processor, equivalence classes, hypervisor, optimization

For citation: Lisovskiy, O.R. (2025). Development tests of creating virtual machines with different virtual processor parameters. *Modelling and Data Analysis, 15*(4)*,* 58—70. (In Russ.). https://doi.org/10.17759/mda.2025150404

Введение

Для обеспечения масштабируемости и надежности современной инфраструктуры облачных сервисов активно применяются гипервизоры с прямым доступом к аппаратным ресурсам серверов (Portnoy, 2012). Гипервизор содержит спектр параметров для виртуальных процессоров (Brown et al., 2019; Hagen, 2008; Ivanov, 2017), что позволяет создать виртуальную машину для практически любой прикладной задачи, при этом не делая всю систему более уязвимой к атакам из вне (Olzak et al., 2010). Однако каждый такой параметр увеличивает общую сложность программы (Jorgensen, DeVries, 2021) и повышает вероятность её нештатной работы на различных этапах (создание виртуальной машины, редактирование её параметров, получение описания виртуальной машины и даже её удаление). Любые ошибки в процессе работы гипервизора могут обернуться значительными потерями времени и финансов. Единственный способ предотвращения ошибок гипервизоров в целом и работы виртуальных процессоров в частности — тщательное тестирование программы. Из-за большого количества разработок программного обеспечения в сфере виртуализации (в том числе для импортозамещения) (Сдобникова, 2025; ТОП гипервизоров 2025…, 2025) проблема тестирования гипервизоров является весьма актуальной. Для увеличения эффективности и надёжности тестирования необходимо минимизировать количество тестов, требующих ручного прогона, и максимизировать количество автотестов — тестов, которые смогут выполняться в автоматическом режиме и по указанному расписанию. Это позволит как значительно ускорить процесс тестирования продукта: человек, очевидно, выполняет все пункты, указанные в протоколе тестирования, медленнее чем компьютер. Кроме того, это позволит минимизировать человеческий фактор в процессе тестирования. В работе рассматривается автоматизированное тестирование компонента «Создание виртуальной машины», в частности создание виртуальной машины с различными параметрами виртуального процессора. Кроме того, приводится список с минимально необходимым количеством тестов создания виртуальных машин с различными параметрами виртуального процессора и учтены все возможные дублирования тесовых сценариев. Минимальное количество получается с помощью разбиения на классы эквивалентности, что позволит сократить большую группу тестов до малой, в том числе единичного теста (Bhat, Quadri, 2015).

Предварительные условия тестирования

Предварительно необходимо определить зависимость создания виртуальных машин от других глобальных компонентов гипервизора, так как они могу заблокировать эту функцию. Существует зависимость от функции установки гипервизора, так как без его инсталляции вообще невозможно произвести тестирование виртуальной машины в любом виде. Поэтому установка гипервизора должна показать успешный результат тестирования. Также в случае гипервизоров первого типа (Nirek, 2021; Portnoy, 2012), предназначенных для работы с тысячами виртуальных машин у крупных компаний, будет существовать прямая зависимость от компонента «Хранилища», так как виртуальная машина будет требовать хранилища для размещения и корректной работы этого хранилища для создания файлов конфигурации, виртуальной оперативной памяти, файлов жёстких дисков и т. д. В случае, если гипервизор взаимодействует с хранилищем через операционную систему (Kipper, Barrett, 2010; Nirek, 2021), данная зависимость опускается, так как операционные системы семейств Linux, Windows, Mac OS и прочие имеют достаточно надёжные системы хранения данных, способные сами по себе выполнять запросы гипервизора. Ошибки могут возникать только со стороны действий виртуальной машины.

В рамках самого компонента «Создание виртуальной машины» необходимо убедиться, что различные конфигурации процессора никак не зависят от других настроек виртуальной машин: оперативная память, жёсткие диски, сеть и т. д. Полностью гарантировать независимость компонентов невозможно, но так как тестирование всех возможных комбинаций настроек виртуальной машины занимает огромное количество времени, необходимо выделить в отдельную категорию тестирование потенциально мешающих друг другу настроек (например добавление процессорных ядер и оперативной памяти без выключения виртуальный машины). Для всех остальных случаев необходимо первоначально проверить создание виртуальной машины со стандартными настройками всех компонентов, кроме жёсткого диска. В качестве жёсткого диска должен использоваться диск с уже установленной операционной системой, так она будет служить подтверждением, что различные настройки процессора корректно работают. Если виртуальная машина успешно создалась, и операционная система этой машины работает в штатном режиме, то во всех тестах компонента «Создание виртуальной машины с разными параметрами процессора» будет принято считать, что они независимы от настроек других компонентов, и во всех тестах будут использоваться стандартные предлагаемые мастером создания виртуальной машины параметры, кроме фиксированного жёсткого диска. Если же было выявлено, что с какими-то из стандартных параметров виртуальная машина не создаётся, то необходимо выбрать один из двух вариантов действий. Если исправление ошибки у отдела разработки будет занимать немного времени (до трёх часов), допускается полная остановка прогона автотестов для создания виртуальной машины. Но если ошибка не будет исправлена быстро или оценка работоспособности различных параметров процессора виртуальной машины обладает повышенным приоритетом, то необходимо подготовить сценарий, определяющий причину проблемы (к примеру, посредством анализа лог-файла или уже имеющегося отчёта об ошибке) с последующей итерационной заменой в коде автотестов значений требуемой переменной до успешного создания виртуальной машины. Но если ошибка заключается в использовании диска с уже установленной операционной системой, то тестирование компонента будет целесообразнее прекратить, так как не будут получены данные о процессорах со стороны операционной системы виртуальной машины, что сделает результаты теста потенциально недействительными.

Для гарантии надёжности тестов необходимо обеспечить их независимость (Crispin, Gregory, 2008). Для этого каждый тест должен работать со своей собственной виртуальной машиной, повторное использование одной машины для тестов на её создания невозможно: она уже создана. Чтобы обеспечить экономию места хранилища, каждая такая машина создаётся на этапе подготовки теста и удаляется на этапе завершения теста. На этапе подготовки копируется диск с установленной операционной системой, так как одновременное использование несколькими виртуальными машинами операционной системы на одном диске неминуемо приведёт к сбоям используемой операционной системы или порче этого диска.

Благодаря всем этим предварительным условиям можно обеспечить параллельное выполнение автотестов. Количество потоков ограничено либо количеством тестов, либо возможностями тестовой среды, либо возможностями гипервизора к параллельной обработке запросов (например, искусственное ограничение на 10 одновременно создаваемых машин).

Далее необходимо обозначить параметры виртуального процессора, требующие проверки: количество процессорных ядер, распределение ядер по сокетам, резервирование частоты процессора, модели процессоров и добавление процессорных ядер без остановки работы машины. Все перечисленные параметры проверяются через мониторинг гипервизора и через средства мониторинга операционной системы, установленной на виртуальную машину.

Работа сконцентрирована на разработке положительных тестов и негативных тестах, которые используют корректный тип данных вводимых параметров. Тесты, которые используют некорректный тип данных (Kaner, Falk, Nguyen, 1999), объединены в единый класс эквивалентности, и подробно рассматриваться будут только два таких теста из-за возможных уникальных ошибок. Остальные будут только обозначены и в итоговое значение количества тестов включены не будут. В рамках каждого класса необходимо проверять оба граничных значения класса и одно неграничное значение (Gupta, Choudhary, 2024). Если в нескольких классах имеется идентичное граничное значение, то рассматривать его необходимо единожды, чтобы не создавать дублирующие тесты.

Тестирование создания виртуальных машин с различным количеством процессорных ядер

Одним из наиболее часто используемых параметров виртуального процессора при создании виртуальной машины является количество виртуальных вычислительных ядер. Они регулируют возможности виртуальной машины в области параллельных вычислений (Kusnetzky, 2011; Portnoy, 2012).

Первоначально необходимо разбить множество возможных значений виртуальных процессоров на классы эквивалентности и выделить границы. Таких классов несколько:

1. $\left[1;\min\left(n_1,n_2\right)\right]\cap\mathbb{N}$;
2. $\left(\max\left(n_1,n_2\right);+\infty\right)\cap\mathbb{N}$;
3. $(n_1,n_2)\cap\mathbb{N}$, если $n_1<n_2$;
4. Неверный тип данных — отрицательные числа, ноль, дробные числа, строковые значения и т. д.

Здесь n_1 — максимально возможное количество ядер, обрабатываемых гипервизором, а n_2 — максимально возможное количество ядер, обрабатываемых физическим хостом. Рассматривается именно такое соотношение, так как если $n_1 \geq n_2$, виртуальная машина точно не может быть создана из-за физических ограничений, и это является составляющей второго класса эквивалентности, в то время как при выбранном соотношении виртуальная машина потенциально может создаться, но не будет соответствовать существующим ограничениям.

Оба этих значения могут, в зависимости от реализации эмуляции ядра процессора, соответствовать как числу физических ядер процессора хоста, что сделает проверку достаточно простой, так и количеству потоков, что усложнит тесты и сделает их менее объективными, так как придётся использовать близкие к максимальным значения, ввиду постоянно меняющегося количества процессов внутри хоста. Отсюда можно выделить следующий ряд тестов, где k_1 — количество ядер на момент создания:

1. $k_1=1$ — проверка создания виртуальной машины с минимальным количеством ядер.
2. $k_1\in\left(1;\min\left(n_1,n_2\right)\right)\cap\mathbb{N}$ — проверка создания виртуальной машины с наиболее часто используемым k_1 .
3. $k_1=n_1$ — проверка создания виртуальной машины с предельно установленным количеством ядер.
4. $k_1\in\left(n_1;n_2\right)\cap\mathbb{N}$ — проверка невозможности создания виртуальной машины, которая может быть физически создана, но не соответствует программным ограничениям.
5. $k_1=n_2$ — проверка создания виртуальной машины с предельным количеством физических ядер на физическом хосте.
6. $k_1>\max\left(n_1,n_2\right)\cap\mathbb{N}$ — проверка невозможности создания виртуальной машины, которая превышает физические и программные ограничения.
7. $k_1=0$ (первый тест с неверным типом данных) — проверка невозможности создания виртуальной машины, без процессорных ядер.
8. Следующие тесты для неверных типов данных — различные тесты на обработку неверных типов данных.

Если не установлен максимум для гипервизора или он гарантированно выше максимума хоста, то третий класс эквивалентности вместе с тестами 3 и 4 опускается.

Тестирование создания виртуальных машин с различным количеством процессорных сокетов

Количество сокетов позволяет определить, будет виртуальная машина просто обладать многоядерным процессором или будет многопроцессорной (Patterson, Hennessy, 2018). Данная настройка нужна для специфических задач виртуальной машины, требующих параллельных вычислений.

Существует одна основная реализация распределения ядер по сокетам при создании виртуальной машины: рассчитанные хостом делители количества ядер (Hagen, 2008). Ручной ввод количества сокетов с последующим равномерным распределением ядер и ручной ввод количества сокетов и ядер в каждом из них практически не встречается из-за сложности реализации, сложности пользования и малой востребованности.

Рассмотрим три основных класса эквивалентности:

1. Один сокет с $m \in \left[1; \min\left(n_1, n_2\right)\right] \cap \mathbb{N}$ ядер;
2. s сокетов с $m: \begin{cases} m \equiv 0 \left(\operatorname{mod} s\right) \\ m \in \left[1; \min\left(n_1, n_2\right)\right] \cap \mathbb{N} \end{cases}$ ядер суммарно;
3. $\min\left(n_1, n_2\right)$ сокетов с одним ядром;

Здесь n_1 и n_2 — ранее введённые максимально возможные количества ядер. Рассматривается именно минимум, так как промежуток между минимумом и максимумом проверяется в предыдущем разделе, что позволяет отсеять ненужные тесты и сократить время прогона.

Неверный тип данных не рассматривается, так как при автоматическом расчёте всех вариантов ввод некорректных данных крайне затруднён или вовсе невозможен.

На основе имеющихся классов эквивалентности можно составить следующий ряд тестов:

1. Один сокет с одним ядром — проверка создания виртуально машины с минимальным количеством ядер и сокетов.
2. Один сокет со стандартным количеством ядер по условию первого класса эквивалентности — проверка создания виртуальной машины с наиболее часто используемыми параметрами.
3. Один сокет с количеством ядер, соответствующим $\min\left(n_1, n_2\right)$ — проверка создания виртуальной машины с одним сокетом с предельно возможным количеством ядер.
4. s сокетов по одному ядру — проверка создания виртуальной машины с несколькими сокетами с минимальным количеством ядер.
5. s сокетов со стандартным количеством ядер по условию второго класса эквивалентности — проверка создания виртуальной машины с несколькими сокетами с наиболее часто используемым количеством ядер на сокет.
6. s сокетов, задействующих все ядра физического хоста — проверка создания виртуальной машины с несколькими сокетами с максимально возможным количеством ядер.
7. Максимальное количество сокетов, равное допустимому количеству ядер физического хоста — проверка создания виртуальной машины с максимальным количеством сокетов.

Имея списки тестов для ядер и сокетов, можно провести ряд оптимизаций. Тест № 1 для ядер можно провести только при наличии одного сокета, поэтому его можно объединить с тестом № 1 для сокетов. Поскольку в тесте № 2 для ядер количество сокетов не существенно, его можно объединить с тестом № 2 для сокетов. Поскольку для теста № 3 для сокетов требуется максимально допустимое число ядер, при $n_1 < n_2$ он может быть объединён с тестом № 3 для ядер, а при обратном значении неравенства — с тестом № 5 для ядер.

Тестирование создания виртуальных машин с различным резервированием частоты процессора

Резервирование минимальной частоты не даёт опуститься значению частоты виртуального процессора ниже заданного значения. Это необходимо для гарантированного обеспечения виртуальной машины вычислительными ресурсами при возможной конкуренции с другими машинами (McAdams, 2015).

Выделим четыре основных класса эквивалентности:

1. $[0; n_3)$;
2. $[n_3; n_4]$;;
3. $(n_4; +\infty)$.
4. Неверный тип данных — отрицательные числа, строковые значения и т. д.

Здесь n_3 — минимальное значение резервирования частоты виртуального процессора, n_4 — максимально возможное значение резервирования частоты, обрабатываемое гипервизором. Поскольку физический процессор формально не имеет физических ограничений на значение частоты, гипервизор получает такое искусственное ограничение. Из имеющихся классов можно составить следующие тесты, где k_2 — резервирование частоты:

1. $k_2 = 0$ — проверка невозможности создания виртуальной машины без резервирования частоты процессора.
2. $k_2 \in (0; n_3)$ — проверка невозможности создания виртуальной машины без резервирования частоты процессора ниже минимальной.
3. $k_2 = n_3$ — проверка создания виртуальной машины с минимальным резервированием частоты процессора.
4. $k_2 \in (n_3; n_4)$ — проверка создания виртуальной машины с резервированием частоты процессора в интервале допустимости.
5. $k_2 = n_4$ — проверка создания виртуальной машины с максимальным резервированием частоты процессора.
6. $k_2 > n_4$ — проверка невозможности создания виртуальной машины с резервированием частоты процессора в интервале допустимости.
7. Тесты для неверных типов данных — различные тесты на обработку неверных типов данных.

Проведём следующую оптимизацию: объединим тест № 4 с тестом № 2 из раздела тестирование создания виртуальных машин с различным количеством процессорных ядер, так как в тесте со стандартными параметрами предполагается, что все возможные настройки виртуального процессора будут стандартными. В случае если тест останется, произойдёт дублирование тестов, что негативно скажется на скорости выполнения тестов.

Стоит отметить, что возможна следующая реализация гипервизора — минимальное резервирование (n_3) равно 0. В таком случае тесты № 1 и № 2 данного раздела опускаются, так как полуинтервал первого класса эквивалентности полностью входит в отрезок второго.

Тестирование создания виртуальных машин с различными моделями процессора

Создание виртуальных машин с различными моделями процессора необходимо для работы с эмулируемыми процессорами. Здесь каждый эмулируемый процессор считается классом эквивалентности. Количество эмулируемых процессоров для каждого продукта своя, поэтому количество необходимых проверок будет m. Однако проверка данного раздела осложнена тем, что одни физические процессоры не могут эмулировать другие (Ivanov, 2017; Portnoy, 2012; Syrewicze, Siddaway, 2018), что влечёт неизбежное увеличение количества проверок в два раза для проверки корректного поведения при запросе создания виртуальной машины с неподдерживаемой моделью процессора. Это обстоятельство также увеличивает затраты на тестовый стенд, так как необходимы как машины способные эмулировать все указанные в документации процессоры, так и машины, неспособные на это. Может потребоваться проверка ещё двух дополнительных случаев — эмулирование модели процессора хоста и прямой проброс процессора в виртуальную машину (Brown et al., 2019).

Проведём небольшую оптимизацию: одна из моделей виртуального процессора будет включаться в тесты ранее описанных разделов, поэтому уникальных тестов будет на один меньше. Итого по данному разделу может понадобиться от $2m-1$ до $2m+1$ тестов.

Тестирование создания виртуальных машин с добавлением процессорных ядер без выключения виртуальной машины

Добавление процессорных ядер без выключения виртуальной машины — особая настройка процессора, которая позволяет регулировать количество ядер в процессе исполнения работы виртуальной машины. Это позволяет не прерывать работу служб, установленных или активно пользующихся ресурсами машины, и увеличить количество выделяемых процессорных ресурсов (Brown et al., 2019; Chisnall, 2008). После включения этой настройки может появиться дополнительное требование: вписать максимальное значение виртуальных ядер, которое может быть после добавления (Brown et al., 2019).

Для настройки имеются следующие классы эквивалентности:
1. Настройка выключена;
2. $\left[1;n_5\right)\cap\mathbb{N}$, при $n_5>2$;
3. $\left[n_5;\min\left(n_1,n_2\right)\right]\cap\mathbb{N}$;
4. $\left(\min\left(n_1,n_2\right);+\infty\right)\cap\mathbb{N}$;
5. Неверный тип данных — отрицательные числа, ноль, дробные числа, строковые значения и т. д.

Здесь n_5 — текущее значение количества процессорных ядер виртуальной машины. В четвёртом классе эквивалентности рассматривается именно минимум n_1 и n_2, так как промежуток между минимумом и максимумом, а также промежуток больше максимума заведомо должны привезти к ошибке создания виртуальной машины из-за превышения порога допустимого значения. Это позволяет отсеять ненужные тесты и сократить время прогона. Из имеющихся классов можно составить следующие тесты, где k_1 — количество ядер на момент создания, а k_3 — количество максимальных ядер после добавления:
1. Настройка выключена — проверка создания виртуальной машины без возможности добавления процессорных ядер без выключения.
2. $k_3=1$, $k_1>2$ — проверка невозможности создания виртуальной машины, если максимальное количество добавленных ядер меньше стартового и равно минимально возможному.
3. $k_3\in\left(1;k_1\right)\cap\mathbb{N}$ — проверка невозможности создания виртуальной машины, если максимальное количество добавленных ядер меньше стартового.
4. $k_3=k_1$ — проверка создания виртуальной машины с максимумом ядер равным текущему значению ядер.
5. $k_3\in\left(k_1;\min\left(n_1,n_2\right)\right)\cap\mathbb{N}$ — проверка создания виртуальной машины с максимумом ядер, превосходящим текущее значение ядер, но меньшее, чем максимальное.
6. $k_3=\min\left(n_1,n_2\right)$, $k_1<\min\left(n_1,n_2\right)$ — проверка создания виртуальной машины с максимально возможным количеством доступных для добавления ядер и текущим количеством меньшим максимума.
7. $k_3=0$ (второй тест с неверным типом данных) — проверка невозможности создания виртуальной машины, если максимальное количество добавленных ядер равно нулю.

Следующие тесты для неверных типов данных — различные тесты на обработку неверных типов данных.

Поскольку одна из указанных настроек (наиболее вероятно — самая первая из-за своей простоты) будет включаться в тесты ранее описанных разделов, уникальных тестов в данном разделе на один меньше. Однако возможна и реализация без ограничения максимального количества добавленных ядер, так как максимум изначально будет равен всем доступным процессорным ядрам, и её проверка ложится преимущественно на тесты изменения виртуальной машины. В таком случае количество уникальных тестов становится равным единице.

Заключение

В результате представленного исследования были выявлены классы эквивалентности для всех основных настроек процессора виртуальной машины при её создании. Из этих классов составлен минимальный набор тестов для каждой отдельной настройки. Значительное пересечение (3 теста) было выявлено только у настроек количества процессорных ядер (в зависимости от реализации 6 или 8 тестов) и количества сокетов (всего 7 тестов). При учёте объединения получается 10 или 12 тестов. Для различных значений частоты процессора было выявлено, в зависимости от реализации, пять или семь тестовых случаев, а с учётом пересекающегося значения с другими тестами — четыре или шесть. Для различных моделей процессора уникальное количество тестов от $2m-1$ до $2m+1$. Для настройки добавления ядер без остановки виртуальной машины было выявлено два или восемь тестов, с учётом пересекающейся настройки от одного до семи уникальных тестов. Из этого следует, что гипервизор с самым малым набором настроек потребует $14+2m$ уникальных тестов, а гипервизор с самым широким спектром настроек — $26+2m$. Эти же числа показывают максимальное количество автотестов, которые могут быть запущены одновременно. Эти тесты позволяют покрыть всевозможные классы эквивалентности настроек без лишних проверок, обеспечивая максимальное тестовое покрытие при минимальных временных затратах, что очень важно для активно разрабатывающегося гипервизора, который нуждается в постоянном тестировании из-за большой частоты обновлений.

Из-за большого удобства и простоты использования данного метода написания тестовых сценариев его резонно рассматривать для написания тестовых сценариев для прочих крупных компонентов настроек при создании виртуальной машины.

Список источников / References

1. Сдобникова, И. (2025). Могут ли российские платформы виртуализации заменить Microsoft, Citrix и VMware в 2025 году. (In Russ.). URL: https://www.anti-malware.ru/analytics/Technology_Analysis/Russian-Virtualization-platforms-2025 (дата обращения: 14.08.2025).
Sdobnikova, I. (2025). Can Russian virtualization platforms replace Microsoft, Citrix, and VMware in 2025. URL: https://www.anti-malware.ru/analytics/Technology_Analysis/Russian-Virtualization-platforms-2025 (viewed: 14.08.2025).
2. ТОП гипервизоров 2025: чем заменить VMware в корпоративной среде. (2025). URL: https://softlist.biz/company/news/top_gipervizorov_2025_chem_zamenit_vmware_v_korporativnoy_srede/ (дата обращения: 15.08.2025).
TOP Hypervisors 2025: What to Replace VMware in a Corporate Environment. (2025). (In Russ.). URL: https://softlist.biz/company/news/top_gipervizorov_2025_chem_zamenit_vmware_v_korporativnoy_srede/ (viewed: 15.08.2025).
3. Bhat, A., Quadri, S.M.K. (2015). Equivalence class partitioning and boundary value analysis — A review. In: 2015 2nd International Conference on Computing for Sustainable Development (INDIACom) (pp. 1557—1562). New Delhi: IEEE.
4. Brown, M., Cartwright, H., Gavanda, M., Mauro, A., Novak, K., Valsecchi, P. (2019). The Complete VMware vSphere Guide. Birmingham: Packt Publishing.

5. Chisnall, D. (2008). The Definitive Guide to the Xen Hypervisor. Boston: Pearson Education.
6. Crispin, L., Gregory, J. (2008). Agile Testing: A Practical Guide for Testers and Agile Teams. Crawfordsville: Addison-Wesley Professional.
7. Gupta, S., Choudhary, J. (2024). An efficient test suit reduction methodology for regression testing. Indonesian Journal of Electrical Engineering and Computer Science, 34(2), 1336—1343. https://doi.org/10.11591/ijeecs.v34.i2.pp1336—1343
8. Hagen, W. (2008). Professional Xen® Virtualization. Indianapolis: Wiley Publishing.
9. Ivanov, K. (2017). KVM Virtualization Cookbook: Learn how to use KVM effectively in production. Birmingham: Packt Publishing.
10. Jorgensen, P.C., DeVries, B. (2021). Software Testing: A Craftsman's Approach. Fifth Edition. Boca Raton: CRC Press.
11. Kaner, C., Falk, J., Nguyen, H.Q. (1999). Testing Computer Software. New York: John Wiley & Sons.
12. Kipper, G., Barrett, D. (2010). Virtualization and Forensics: A Digital Forensic Investigator's Guide to Virtual Environments. Burlington: Syngress.
13. Kusnetzky, D. (2011). Virtualization: A Manager's Guide. Sebastopol: O'Reilly Media.
14. Nirek, R. (2021). Comparative Study of Hypervisor Technologies for Running Enterprise Applications on Linux VMs. Journal of Scientific and Engineering Research, 8(3), 278—284. https://doi.org/10.13140/RG.2.2.22497.52320
15. McAdams, S. (2015). Virtualization Components of the Modern Hypervisor: UNF Graduate Theses and Dissertations. University of North Florida. Jacksonville.
16. Olzak, T., Sabovik, J., Boomer, J., Keefer, R.M. (2010). Microsoft Virtualization: Master Microsoft Server, Desktop, Application, and Presentation Virtualization. Burlington: Syngress.
17. Patterson, A.D., Hennessy, J.L. (2018). Computer Organization and Design RISC-V Edition: The Hardware Software Interface. Cambridge: Morgan Kaufmann.
18. Portnoy, M. (2012). Virtualization Essentials. Indianapolis: John Wiley & Sons.
19. Syrewicze, A., Siddaway R. (2018). Pro Microsoft Hyper-V 2019: Practical Guidance and Hands-On Labs. Berkeley: Apress Berkeley.

Информация об авторах

Лисовский Олег Романович, аспирант кафедры «Вычислительная математика и программирование», Московский авиационный институт (национальный исследовательский университет), Москва, Российская Федерация, ORCID: https://orcid.org/0009-0000-7768-9182, e-mail: Lisoleg555@yandex.ru

Information about the authors

Oleg R. Lisovskiy, postgraduate student, Department of Computational Mathematics and Programming, Moscow Aviation Institute (National Research University), Moscow, Russian Federation, ORCID: https://orcid.org/0009-0000-7768-9182, e-mail: Lisoleg555@yandex.ru

Вклад авторов

Все авторы приняли участие в обсуждении результатов и согласовали окончательный текст рукописи.

Contribution of the authors

All authors participated in the discussion of the results and approved the final text of the manuscript.

Конфликт интересов

Авторы заявляют об отсутствии конфликта интересов.

Conflict of interest

The authors declare no conflict of interest.

Поступила в редакцию 09.10.2025
Поступила после рецензирования 25.10.2025
Принята к публикации 05.11.2025
Опубликована 28.12.2025

Received 2025.10.09
Revised 2025.10.25
Accepted 2025.11.05
Published 2025.12.28