Санкт-Петербургский Государственный Университет

Математико-Механический факультет

Кафедра системного программирования

Безопасное рабочее пространство пользователя

Multi-Cloud Desktop

Конфигурация системы.
Построение протокола взаимодействия компонентов

Курсовая работа студента 445 группы

Серко Сергея Анатольевича

Научный руководитель.. . . . . Баклановский М.В.

старший преподаватель

кафедры системного программирования

Санкт-Петербург

2013

Содержание

1. Введение

В наше время во многих сферах жизнедеятельности человека важную роль играют различные информационные системы. В течение последних 50 лет мир IT очень бурно развивался. Однако есть технологии, которые уже более полувека считаются основополагающими, и без них невозможно представить ни одну современную систему. Виртуализация – одна из таких концепций, которая лежит в основе подавляющего большинства компьютерных систем.

1.1 Базовые понятия

В частном случае, виртуализация – это совокупность технологий, для организации выполнения приложений в изолированной среде так, что:

- приложения не влияют на стабильность работы других приложений за пределами данной среды;
- приложения получают доступ к ресурсам системы так, будто они работают с ними напрямую;
- на одном ПК могут одновременно выполняться несколько групп приложений в отдельных средах. [31]

«Автономные» среды из определения – виртуальные машины (ВМ). Программы, контролирующие их работу – гипервизоры, или менеджеры ВМ (Virtual Machine Manager, VMM). Каждая ВМ содержит основные устройства, присущие обычному компьютеру: процессоры, память, жесткие диски, устройства I/O. При работе виртуальных машин необходимо управлять доступом к ЦПУ и выполнением команд, проводить переадресацию из памяти ВМ в основную память, перенаправлять потоки ввода/вывода. Для реализации подобных действий используются сложные программные и аппаратные технологии.

Не стоит забывать о проблеме депривилегизации при работе с виртуальным процессором. В интеловской архитектуре в ring 0 работает основная ОС либо менеджер виртуальных машин [12]. Т.о. нужно правильно обрабатывать инструкции гостевых ОС (Guest OS), не имеющих необходимых привилегий. Решать эту проблему можно двумя путями.

1. *Бинарная трансляция* состоит в том, что VMM перехватывает инструкции гостевых операционных систем и корректно исполняет их. Это очень ресурсоемкая методика, однако плюс заключается в ее универсальности.

2. *Паравиртуализация* заключается во внесении изменений в ядро Guest OS. В результате проведенных модификаций системные вызовы гостевой ОС направляются в гипервизор из чего следует очевидное ограничение по применению метода: он возможен только для opensource операционных систем.

В начале 80-х гг. Р.Голдбергом были описаны 2 типа гипервизоров: native и hosted [2]. Native-VMM являются самым нижним уровнем ПО в системе. Например: Xen, KVM, Microsoft Hyper-V, VMware ESXi. Hosted-VMM работают, как обычные приложения хостовой операционной системы (Host-OS). Яркими представителями таких гипервизоров считаются VMware Workstation, Parallels Desktop и Oracle VirtualBox.

Позже классификация VMM, предложенная Голдбергом, была уточнена компанией Intel (см. Рис. 1): hosted был поделен на host-based и containers, а native – на hypervisor и micro-hypervisor.

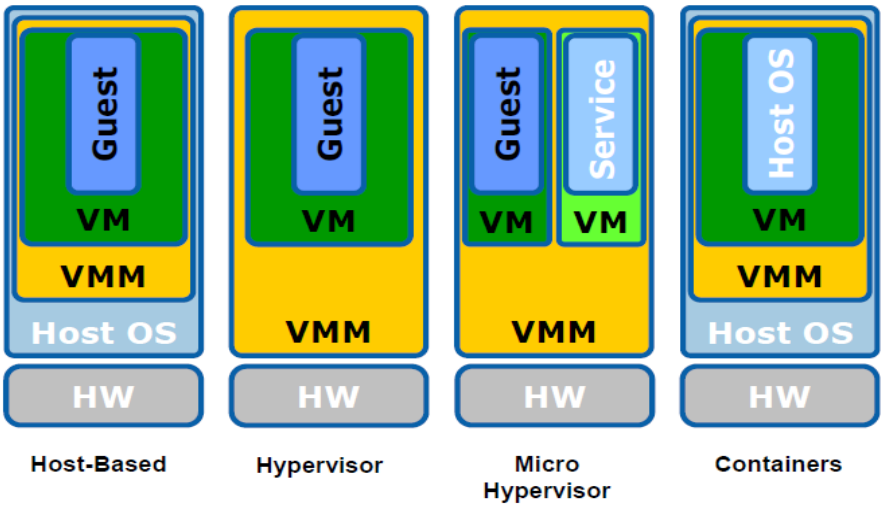

Рис. 1

Hypervisor VMM представляет собой специальную ОС с драйверами для всех устройств. Возможна поддержка разных гостевых ОС; каждая виртуальная машина имеет полный набор виртуальных устройств. В качестве ограничений следует отметить, что при данном подходе для VMM необходима дополнительная память, и им поддерживается ограниченное число устройств. Пример продукта: VMware ESX Server.

Micro-hypervisor VMM представляет собой специальную ОС; работу с устройствами осуществляет привилегированная VM. Данный подход позволяет запускать разные гостевые операционные системы и работать с широким набором устройств, но придется пожертвовать производительностью из-за взаимодействия с сервисной ОС. Примеры соответствующих продуктов: XEN, KVM, Citrix XenServer.

Host-Based гипервизор запускается в среде базовой ОС и использует ее ресурсы. Виртуальная машина имеет полный набор виртуальных устройств. В такой системе возможна поддержка разных гостевых ОС, однако резко снижается производительность из-за активного расходования ресурсов системы. Этот метод вполне универсален. Примерами существующих систем служат: VMware Server, Parallels Server.

Гипервизор, реализующий механизм *containers*, «клонирует» базовую ОС. VMM разделяет ресурсы между виртуальными машинами – изолированными контейнерами. При таком подходе нет необходимости в виртуализации большинства устройств и расход памяти на виртуальную среду сравнительно небольшой. Существенный минус подобных менеджеров виртуальных машин заключается в том, что все ВМ выполняют одну ОС. Пример существующих решений: Parallels Virtuozzo Containers [31].

1.2 Эволюция виртуализации

Впервые виртуализация была использована в мэйнфреймах IBM 360/65. В своей первой реализации она позволяла лишь распределять память: половину для работы в режиме IBM 360 и половину для эмуляции IBM 7080. Логика использования памяти переключалась специальным софтом, называемым «гипервизор» [10].

Всем известная концепция виртуальной памяти была разработана в 1956 году в Берлинском Техническом Университете. В 1966 году в системе IBM System 360/67 было реализовано виртуальное адресное пространство. Память была разбита на страницы по 4 KB, сгруппированные в сегменты по 1 MB. Страницы динамически сопоставлялись реальной памяти [26].

Следующей вехой в развитии технологии виртуализации является SIMMON (SIMulation MONitor) – программный продукт, разработанный IBM в конце 60х годов для тестирования ПО. Это был гипервизор, поддерживавший всего одну виртуальную машину, внутри которой могла исполняться лишь одна тестируемая программа. Несмотря на это ограничение, долгие годы SIMMON был популярен среди программистов благодаря своей возможности динамического подключения инструментов тестирования [23].

Параллельно с SIMMON в IBM была разработана CP-40 – первая операционная система, в которой была реализована полная виртуализация. Данная ОС предоставляла среду виртуализации, поддерживавшую окружение целевой компьютерной системы S/360-40. Таким образом другие S/360-подобные системы могли быть установлены,

протестированы и использованы так, будто они работают на отдельной машине. CP-40 поддерживала до 14 одновременно работающих виртуальных машин. Каждая виртуальная машина работала в особом режиме: привилегированные инструкции вроде операций ввода/вывода вызывали исключения, которые затем отлавливались специальной контролирующей программой и эмулировались [11].

Очередной шаг в развитии виртуализации был также сделан компанией IBM, сотрудниками которой был создан продукт PR/SM (Processor Resource/System Manager) – гипервизор, позволяющий нескольким логическим компонентам LPAR (Logical Partition Access Resources) разделять такие физические ресурсы, как CPU, устройства хранения прямого доступа и пр. Впервые система PR/SM была представлена в 1985 году в системе IBM System z [25].

В 1994 году в MIT была введена новая концепция виртуализации – экзоядро. В рамках этого подхода операционная система реализовывала лишь примитивные механизмы работы с ресурсами компьютера, а весь прочий функционал для запуска приложений и предоставления им соответствующих абстракций ресурсов выносился в специальные библиотеки – libOS'ы. Таким образом, можно было запускать приложения, написанные под различные ОС, на одной машине с использованием соответствующих библиотек [15].

В конце 1990-х/начале 2000-х началось бурное развитие области виртуализации десктопов: свои решения представили VMware (Virtual Station и Workstation), Oracle (Virtual Box), Microsoft (Virtual PC, Hyper-V), Parallels (Virtuozzo Containers, Desktop), Xen. Такие системы позволяют на одной физической машине запускать сколько угодно различных операционных систем (их количество лимитировано лишь количеством доступных ресурсов).

В 2005-2006 гг. Intel и AMD сделали акцент на аппаратной виртуализации, анонсировав технологии Intel-VT и AMD-V, позволяющие эффективно виртуализировать ПО непосредственно с использованием «железа» [13, 1].

В последние годы существует тенденция перехода к мультихостовой серверной архитектуре компьютерных систем. Суть так называемой облачной инфраструктуры заключается в виртуализации имеющихся распределенных ресурсов и предоставлении их конечному пользователю в виде независимых абстракций. Например, всем известные SaaS, DaaS, IaaS, PaaS (software, desktop, infrastructure, platform as a service соответственно)

предоставляют клиентам логические абстракции процессора, памяти, файлового хранилища, приложений и т.д.

1.3 Современное применение виртуализации

В современном мире виртуализация стала неотъемлемой частью IT-индустрии. Основные области применения технологий виртуализации:

- Консолидация серверов

  Идея заключается в том, что с использованием виртуализации можно использовать машины более эффективно. Нет необходимости для двух разных систем держать два разных физических сервера: можно установить специальное ПО и запустить на одном сервере обе системы. Это позволит снизить затраты на оборудование, электропитание, оптимизировать затраты на поддержку и организовать гетерогенные системы.
- Изоляция сервисов

  Данный подход позволяет приложениям работать на одном физическом хосте, но в разных изолированных средах. Это сводит на нет риск краха системы, вызванный аварийным завершением работы любого из изолированных процессов.
- Тестирование ПО

  Как следствие предыдущего пункта можно выделить задачу тестирования программного обеспечения. Инженеры могут запускать потенциально опасные программы в виртуальных средах, не боясь за судьбу системы в целом. Даже если виртуальная машина «упадет», на работе системы это никак не скажется.
- Отказоустойчивость

  Отказоустойчивость – одно из важнейших применений технологий виртуализации, поэтому ее стоит рассмотреть подробнее [20, 31].

1.4 Отказоустойчивость

Отказоустойчивость – это способность системы продолжать корректно работать при отказе (механической или алгоритмической причине ошибки) одной или нескольких подсистем, от которых она зависит [34]. Отказы могут представлять собой ошибки программного и аппаратного обеспечения, непредусмотренные сценарии использования (ошибки пользователя), ошибки каналов связи и др. Зачастую отказы анализируют, измеряя следующие параметры:

- продолжительность – постоянный ли это отказ или временный;
- масштабность – локальность/глобальность отказа;
- значимость – зависит от числа логических ошибок, вызванных отказом.

Отказоустойчивость идет бок о бок с высокой доступностью – методом проектирования системы, позволяющим достигать высокого уровня доступности системы в течение какого-либо промежутка времени. Доступность обычно выражается в процентах uptime (беспрерывного времени доступности пользователям) в год. Всем хорошо известное обозначение высокой степени доступности «пять девяток» означает, что система доступна пользователю 99.999% рабочего времени (т.е. примерно 5 секунд в год система «не откликается» пользователю).

Отказоустойчивая система должна корректно продолжать работу даже в экстремальных ситуациях, руководствуясь следующим абстрактным алгоритмом.

1. *Обнаружение ошибок*. Механизм, способный определить, когда произойдет ошибка, и тем самым позволяющий системе предотвратить ее. В идеале, механизм обнаружения ошибок должен опираться только на спецификацию системы и быть полностью независимым от самой системы.
2. *Обработка отказов*. Детектированная ошибка является лишь симптомом отказа и не обязательно определяет его. Задача нахождения отказа и его устранения в большинстве случаев довольно сложна. Вместо полного устранения отказа можно его игнорировать и продолжать работу, частично устранив повреждения. Относительно конкретной компоненты системы существуют две тривиальные стратегии устранения отказа: замена или реконфигурация.
3. *Оценка повреждений*. Обычно происходит на этапе обработки отказа либо восстановления системы.
4. *Восстановление системы*.

- Backward – стратегия, основанная на восстановлении системы к состоянию, зафиксированному точкой восстановления. Точки восстановления могут быть получены, используя разные техники…
- Forward – стратегия, основанная на попытке продолжить работу системы с ошибочного состояния. Например, используя различные обработчики ошибок.

Есть два принципиально разных пути достижения отказоустойчивости. Первый – за счет использования избыточности системы. Отказоустойчивости можно добиться, внедряя в систему дополнительные компоненты и особые алгоритмы работы, позволяющие системе переживать ошибочные состояния. Степень отказоустойчивости определяется тем, с каким успехом выявляются и восстанавливаются ошибочные состояния. Однако избыточность невыгодна в первую очередь по экономическим причинам: необходимо «с запасом» покупать дорогостоящее оборудование [8].

Второй способ подразумевает использование виртуализации, а именно гипервизора – программного продукта, позволяющего нескольким ОС независимо работать на одном компьютере в изолированных средах и не влиять друг на друга в случае отказов. Таким образом, эта концепция позволит снизить время недоступности системы и поможет существенно сэкономить на оборудовании. Можно развить идею отказоустойчивой виртуализации и применить технику миграции: перенос последнего корректного состояния системы на другую машину и возобновление работы без существенного (заметного пользователю) простоя (см. Рис. 2).

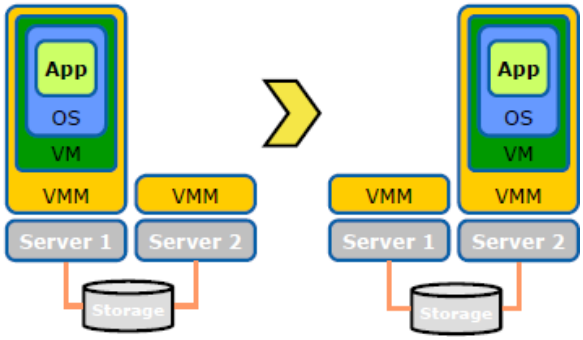

Рис. 2

2. Предварительная постановка задачи

2.1 Отказоустойчивая виртуализация на основе гипервизора Xen

2.1.1 Предпосылки использования виртуализации

Как было указано ранее, для достижения устойчивости компьютерной системы к программным и аппаратным сбоям можно воспользоваться двумя способами. Первый способ эксплуатирует принцип избыточности. Такой способ сложен и дорог в реализации: требует первоклассной квалификации программистов и инженеров, а также закупки качественных, дорогих архитектурных компонентов. Второй способ подразумевает применение концепции виртуализации. Именно он и был рассмотрен нами при реализации отказоустойчивой системы. В качестве организатора виртуальной среды нами был выбран гипервизор Xen. Для создания отказоустойчивой системы с его помощью необходимо наличие в гипервизоре следующих дополнительных модулей:

- DRBD (Distributed Replicated Block Device)
- Remus

2.1.2 Результаты

В результате на нелегком пути к отказоустойчивой системе было преодолено множество трудностей, связанных с администрированием системы. Собрана система, состоящая из двух идентичных хостов и основанная на гипервизоре Xen; в качестве Domain0 выбрана ОС OpenSUSE, т.к. в ней существует встроенный функционал для установки Xen, и нет необходимости ручного процесса установки через терминал. В качестве Dom0 были опробованы также такие ОС, как: Debian, Ubuntu, Arch Linux. На первых двух система не заработала после продолжительных попыток ее оживить, однако на Arch Linux после аккуратной ручной настройки она успешно запустилась. Были сконфигурированы модули Remus и DRBD: расстановка контрольных точек производилась с частотой 40 раз в миллисекунду, определены 2 одинаковых DRBD-устройства для репликации размером по 40 GB. В качестве гостевых доменов использовались виртуальные машины Ubuntu и Windows XP. Было показано, что настройки рассматриваемого VMM позволяют сконфигурировать сетевую топологию таким образом, что удаленный клиент практически не заметит выхода из строя одного из хостов (на эхо-запросы к виртуальным машинам не перестают приходить эхо-ответы вне

зависимости от работоспособности одного из хостов). Таким образом, была подтверждена успешная миграция этих доменов, как запланированная со стороны администратора, так и экстренная в контексте технологии отказоустойчивости (системам посылался сигнал аварийного завершения, после которого с минимальными задержками их работа возобновлялась на запасном хосте).

На системе был поставлен ряд экспериментов с целью определения зависимости ее производительности от использования технологии отказоустойчивости. В частности, на примере запущенной небольшой базы MySQL, выполняющей вставку N записей в таблицу, были получены следующие результаты: производительность приложений, полностью работающих в памяти, снизилась в среднем на 28%.

Домен	**Dom0**	**DomU**
Время без отказоустойчивости	9,5 ± 1,9 сек.	9,6 ± 1,9 сек.
Время с отказоустойчивостью	12,2 ± 2,1 сек.	138,3 ± 24,1 сек.

Таб. 1

В первом столбце показаны времена вставки в таблицу N записей в рамках домена 0, а во втором – в рамках гостевой ОС. Строки означают, была ли включена технология отказоустойчивости.

Далее рассмотрим подробнее ключевой элемент системы – гипервизор Xen, так как он играет ключевую роль в дальнейших исследованиях [32, 33].

2.2 Xen

В качестве менеджера виртуальных машин был выбран представитель класса micro-hypervisor – Xen. Данный продукт распространяется по лицензии GNU General Public License, имеет открытый исходный код и пользуется немалой популярностью. Xen, в силу своих особенностей установки, требует наличия на обеих физических машинах предустановленной ОС Unix-семейства [6].

2.2.1 Xen vs. KVM

Вообще, на данный момент существует два популярных активно развивающихся гипервизора с открытым кодом: Xen hypervisor и Linux Kernel-based Virtual Machine (KVM). Наш выбор менеджера виртуальных машин был обусловлен следующими преимуществами Xen по сравнению с KVM:

- Xen – «тонкий» гипервизор

  KVM использует ядро Linux целиком – со всеми драйверами и интерфейсами, что размывает границу между кодом ядра и кодом, который обрабатывает события виртуальных машин. В Xen-е весь управляющий виртуальными машинами код сосредоточен в Domain 0, что позволяет упростить аудит исполняемого гипервизором кода – кристально ясно, какой код принадлежит менеджеру виртуальных машин.

- Наличие PV драйверов

  KVM использует полную виртуализацию для всех своих виртуальных машин. Это означает, что у каждой его виртуальной машины должен быть соответствующий эмулятор I/O операций.

  В свою очередь Xen позволяет создавать виртуальные машины как в режиме полной виртуализации, так и в режиме паравиртуализации, в котором за счет особой переконфигурации ядра гостевой системы нет необходимости в сложном, уязвимом для атак эмуляторе I/O.

- Возможность создавать домены драйверов

  Xen позволяет выделять «песочницы» для отдельных драйверов, таким образом даже в случае критической ошибки драйвера система защищена от аварийного завершения. Такие песочницы – специальные виртуальные машины, работающие в режиме паравиртуализации, которые имеют доступ к определенным PCI устройствам с помощью Intel VT-d. Это свойство позволяет вывести безопасность исполнения соответствующего кода на более высокий уровень [24].

2.2.2 Архитектура Xen

На представленной ниже диаграмме показана базовая организация основных компонент системы Xen.

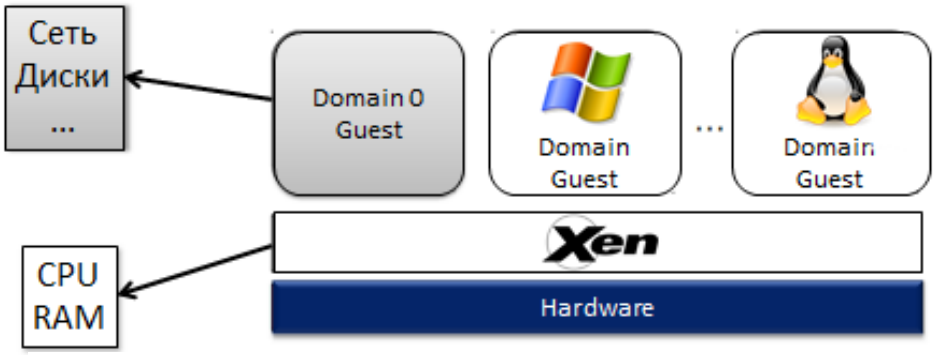

Рис. 3

Гипервизор Xen – базовый уровень абстракции софта, самая нижняя прослойка между аппаратными ресурсами и гостевыми ОС, которая отвечает за распределение ресурсов CPU и памяти среди виртуальных машин. Он не только предоставляет абстракции ресурсов, но и контролирует работу ВМ, разделяющих единую исполняемую среду. Однако не все управляющие функции сосредоточены в гипервизоре – Xen не управляет сетью и остальным I/O. Для этих функций выделяется отдельная сервисная операционная система – Domain 0.

Domain 0 – это ОС с модифицированным ядром Linux, имеющая специальные права доступа к физическим устройствам I/O и каналам связи с другими гостевыми ОС (Domain U PV/HVM). Для работы Xen требуется, чтобы Domain 0 был запущен перед запуском любой из гостевых ОС.

Domain U не имеют прямого доступа к аппаратным ресурсам – все операции ввода/вывода делегируются Domain 0, поэтому такие гостевые системы и называются Unprivileged.

Паравиртуализированные ВМ, запущенные на Xen, называются Domain U PV и являются модифицированными ОС семейств Linux, Solaris, FreeBSD и других UNIX-подобных. ВМ, основанные на динамической (бинарной) трансляции, – Domain U HVM (fully virtualized machine). На них без модификации ядра запускаются ОС семейства Windows (т.к. изменение их ядра не представляется возможным) или любые другие ОС.

Виртуальная машина Domain U PV сконфигурирована так, что её запросы к сети и жесткому диску напрямую отправляются в Domain 0. К тому же она «видит» присутствие

других ВМ. В свою очередь Domain U HVM не в курсе, что делит ресурсы с другими ВМ – её запросы перехватываются Domain 0.

Domain 0 содержит 2 драйвера для работы с сетью и жесткими дискам Domain U: Network Backend Driver и Block Backend Driver. Network Backend Driver общается непосредственно с сетевой картой, чтобы обработать все запросы от Domain U. Block Backend Driver общается с диском для чтения/записи информации Domain U.

Domain U PV содержит 2 драйвера для сети и дисков соответственно: PV Network Driver и PV Block Driver. Domain U HVM не располагает PV-драйверами, вместо этого для каждой из них создается специальный демон в Domain 0, называющийся Qemu-dm. Qemu-dm перехватывает обращения к сети и дискам виртуальных машин Domain U HVM.

Проиллюстрируем взаимодействие Domain 0 и Domain U примером записи данных Domain U PV на диск.

Domain U PV Block Driver получает запрос на запись данных на диск и пишет через Xen hypervisor в соответствующую локальную память, разделяемую с Domain 0. Существует «событийный канал» между Domain 0 и Domain U PV, который позволяет им общаться посредством асинхронных междоменных прерываний в Xen hypervisor. Domain 0 получает прерывание от Xen hypervisor, побуждающее PV Block Backend Driver к доступу к локальной разделяемой системной памяти для чтения записанного блока данных, пришедшего от Domain U PV. Затем эти данные пишутся на жесткий диск [9].

2.2.3 Установка Xen

Xen инсталлируется не на «чистую» машину. Для того, чтобы развернуть Xen, необходима установленная и настроенная операционная система, которая может работать в Domain 0. Многие ошибочно полагают, что Xen является гипервизором хостового типа, т.к. процесс его инсталляции осуществляется в рамках предустановленной ОС. Однако Xen – представитель класса native-гипервизоров: во время установки он модифицирует ядро имеющейся ОС и «подлезает» под нее, превращая ее в Domain 0 и становясь таким образом самым низким уровнем ПО в системе. При старте системы сначала запускается гипервизор, а затем исходная операционная система.

Итак, для преобразования ОС в Domain 0 нужно установить гипервизор, ядро, портированное на Xen и способное работать в домене 0, утилиты Xen, необходимые для управления другими доменами, а также изменить конфигурацию загрузчика.

По завершению работ с Domain 0 можно заняться подготовкой гостевых систем – Domain U. Для этого необходимо завести дисковый раздел или файл, который будет использоваться в качестве диска виртуальной машины. На следующем шаге на эти виртуальные диски необходимо инсталлировать гостевую ОС, а так же создать в Domain 0 конфигурационный файл виртуальной машины.

Запуск и управление виртуальными машинами осуществляется через специальные утилиты Xen, установленные в Domain 0. Главным интерфейсом по управлению гостевыми доменами является программа xm. С её помощью можно создавать, приостанавливать и останавливать домены, а также просматривать список доменов, управлять количеством и привязкой виртуальных процессоров, подключать и отключать виртуальные устройства. Все операции xm выполняются с помощью специального управляющего демона xend, запускаемого во время загрузки машины.

2.3 Дальнейшие пути развития

После построения отказоустойчивой системы на основе гипервизора Xen мы решили провести обзор возможных направлений развития данной тематики. Хотелось сохранить полученные результаты и применить уже имеющийся опыт работы с Xen в будущих задачах.

Среди возможных путей развития явно были выделены следующие.

2.3.1 Тестирование отказоустойчивой системы

Эта ветвь подразумевала под собой тестирование отказоустойчивой системы под различными нагрузками и в различных «экстремальных» условиях (выключение электричества на одном из хостов, искусственный вывод одной или нескольких компонент системы из строя, искусственная генерация программных ошибок и прочее). Планировалось создать детальный отчет о том, какие ошибки «переживаются» системой наиболее болезненно, оценить, насколько велик риск полного или частичного краха системы в результате последовательных ошибок, каковы вероятность и длительность пребывания хостов в неконсистентном состоянии и прочее.

2.3.2 Профайлинг Remus

Данная задача заключалась в детальном понимании и доработке механизма расстановки контрольных точек Remus и соответствующего алгоритма восстановления работоспособности системы. Среди приоритетных направлений было отмечено совершенствование процесса непосредственной передачи чек-поинтов, а именно внесение улучшенных методов сжатия передаваемых данных для ускорения инкрементальной репликации. В качестве результата ожидался сравнительный анализ реализаций Remus с различными вариантами сжатия. Однако после погружения в данную тематику было обнаружено исследование канадских коллег из UBC (University of British Columbia), которые рассмотрели различные алгоритмы сжатия информации и сравнили их реализации применительно к Remus [22].

2.3.3 Автомасштабирование ресурсов

Идея заключалась в том, чтобы реализовать некую дополнительную логику над Xen, которая смогла бы управлять доступными ресурсами и адекватно перераспределять нагрузку на отказоустойчивую систему в целом. Цель – реализовать балансировщик нагрузки, способный при создании новых или миграции существующих ВМ распределить их на определенные хосты, благодаря чему эффективнее распределялись бы ресурсы системы.

В качестве возможной альтернативы допускалась и задача реализации «кластера» из построенной отказоустойчивой системы. Основная мысль заключалась в том, чтобы использовать вычислительные мощности простаивающей backup-машины для запуска требуемых ВМ и, соответственно, реализовать двусторонний механизм контрольных точек.

2.3.4 Qubes-OS

Qubes-OS – операционная система с открытым исходным кодом, созданная, чтобы предоставить высокую степень защиты при работе с различными приложениями. Qubes-OS основана на Xen, X Window System и поддерживает работу с большинством Linux-приложений и драйверов. Первый релиз был выпущен в сентябре 2012 года. Второй релиз сейчас находится в разработке: планируется поддержка виртуальных машин Windows [19].

Общая идея Qubes-OS заключается в изоляции приложений друг от друга и запуске их в специальных виртуальных машинах (AppVM). Пользователю предоставлена возможность создавать множество доменов и определять их уровни доверия/безопасности. Эти домены представляют собой легковесные виртуальные машины, запущенные под гипервизором Xen. Например, пользователь может создать:

- «Random»-домен

  Такой домен может быть использован для обычного интернет-серфинга, чтения новостей, просмотра видео на YouTube и т.п. Также можно установить сюда приложения из небезопасных источников. Как только эта виртуальная среда начинает показывать признаки заражения, пользователь может быть спокоен за состояние других доменов, а этот домен просто перезапустить. Qubes вернет его в последнее корректное состояние.

- «Social»-домен

  Может быть использован для просмотра e-mail, общения в соц.сетях и работы с другими приложениями, связанными с интернет-профилем пользователя.

- «Shopping»-домен

  Может пригодиться для покупок в интернете с использованием кредитной карты и т.п. (если пользователь пользуется несколькими картами, имеет смысл создать несколько доменов для работы с ними).

- «Bank»-домен

  Может использоваться для работы с банками через интернет посредством защищенных соединений и пр.

- Прочие домены.

Основная цель – обеспечить строгое разделение между доменами так, что в случае атаки на приложения в рамках одного домена другие остаются в целости и сохранности. Однако Qubes-OS не предоставляет защиту приложений друг от друга в рамках одного домена. В рассматриваемой операционной системе реализованы механизмы, облегчающие ее использование, например: интеграция GUI приложений на одном рабочем столе, благодаря чему создается иллюзия обычного локального исполнения, защищенный процесс копирования/вставки между доменами, защищенная передача файлов между доменами и прочее. Также для удобства запущенные приложения при отображении на

рабочем столе окаймляются рамочками разных цветов, что позволяет контролировать уровень безопасности соответствующего домена.

Основной архитектурной идеей является строгое разделение различного функционала системы и помещение соответствующих процессов в изолированные домены (см. Рис. 4).

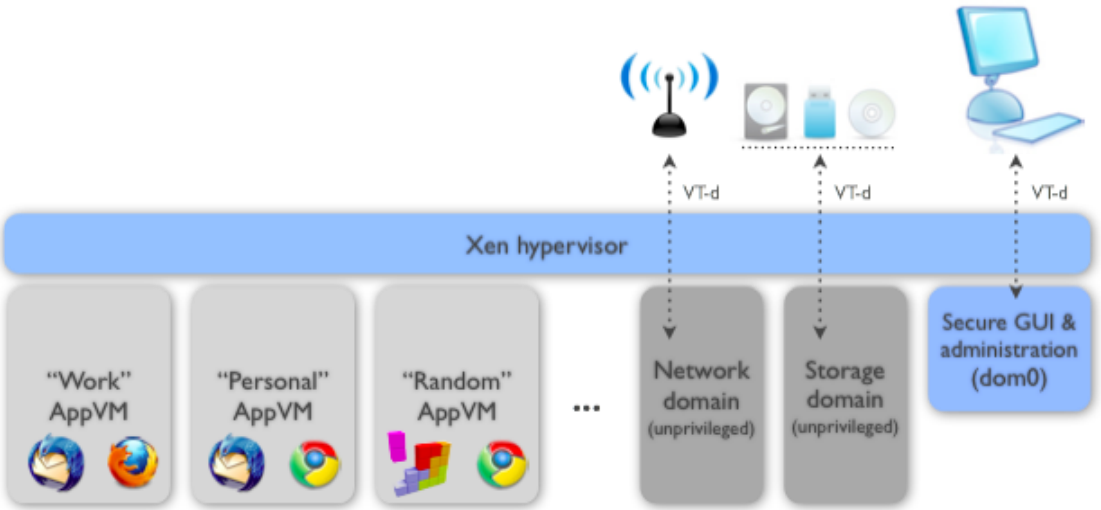

Рис. 4

Таким образом, весь сетевой код изолирован в непривилегированную виртуальную машину и работает с помощью IOMMU/VT-d; USB-стек и драйверы находятся также в отдельном домене. В привилегированном Dom0 (в терминологии Xen) нет никакого сетевого кода, присутствуют лишь графическая подсистема, отвечающая за вывод приложений на экран, и необходимый административный функционал. Это единственная виртуальная машина, имеющая прямой доступ к графическому устройству и устройствам ввода (клавиатуре, мышки). В Dom0 работают XenStore (стандартная административная утилита Xen для управления системы в целом), X-server (клиентская часть системы X Window System, обеспечивающей протокол для построения графического интерфейса пользователя), который отображает рабочий стол пользователю, и оконный менеджер (Window manager), позволяющий запускать и останавливать приложения и управлять их окнами.

Детали реализации компонентов Qubes-OS подробно описаны в документации [24], и нет смысла дублировать их здесь. Для нас же важная сама концепция построения подобной архитектуры системы, основанная на разделении привилегий запущенных процессов и их изоляции.

Мы выделили два основных пути дальнейшего развития.

Первый связан с безопасностью Qubes-OS и исследованием потенциальных векторов атак. Например, можно заниматься поиском уязвимостей в гипервизоре. Это самый привилегированный элемент системы, поэтому любые дыры в его безопасности критичны. Возможны уязвимости и в XenStore-демоне (в Dom0), воспользовавшись которыми, злоумышленник сможет захватить управление всеми виртуальными машинами. Также стоит уделить внимание попыткам получить контроль над GUI-демоном в Dom0, использовать уязвимости сетевого кода системы в соответствующем домене, считать общую память из непривилегированного домена, эмулировать (для пользователя) привилегированное приложение из непривилегированного, например, с целью получения пар логин/пароль, и т.п.[24].

Второй путь развития – модификация архитектуры Qubes-OS и построение ее распределенного аналога. Т.е. позволить Qubes-OS работать сразу на нескольких серверах с сохранением основной концепции изоляции приложений в доменах и организовать балансировку нагрузки и прочие «облачные» алгоритмы.

Однако идею разделения приложений по безопасным доменам с учетом условно неограниченных ресурсов и возможностей облаков можно усовершенствовать, особенно в контексте задачи, о которой пойдет речь далее.

3. Проблематика и постановка задачи обеспечения безопасного рабочего пространства пользователя

3.1 Общее описание проблемы

При работе в среде с высокими требованиями к безопасности необходимо свести к минимуму возможность внедрения вредоносного ПО и, соответственно, вероятность кражи конфиденциальных сведений. Если пользователь хранит или обрабатывает секретную информацию локально, то любая брешь в защите может привести к ее потере. Основными источниками угроз являются запускаемые на локальной машине приложения и сеть Интернет.

Сейчас описанная проблема решается с использованием физически разных компьютеров: одного для взаимодействия непосредственно с важными данными, а другого – для запуска необходимых второстепенных приложений и выхода в Интернет. Такой способ очень неудобен, поэтому предлагается работать со всеми приложениями и данными в рамках одной машины, причем делать это безопасно. Для этого необходимо изолировать компьютер пользователя от доступа к сети Интернет, переместить важные данные в отдельное хранилище с повышенной степенью защиты, отказаться от исполнения приложений локально и рассредоточить их по удаленным серверам в соответствии с требуемым уровнем безопасности. Клиенту будут передаваться только состояния приложений.

3.2 Идея решения

Первый вполне разумный вариант, который приходит в голову, – это применение виртуальных машин. Можно запускать приложения, необходимые пользователю, в различных виртуальных «песочницах» (идея Qubes-OS, описанная ранее), что обезопасит нас, на первый взгляд, от возможных атак. Однако существуют техники, позволяющие понять, находитесь ли вы в рамках некоторой виртуальной машины, и преодолеть этот виртуальный «барьер» [3,7,14,16].

Но можно усовершенствовать эту мысль, допустив рассредоточение виртуальных машин серверам и построив систему, получившую название Multi-Cloud Desktop.

3.3 Multi-Cloud Desktop

Клиент, используя свой локальный компьютер, работает с необходимыми приложениями: он видит привычные окна запущенных программ, взаимодействует с ними через клавиатуру/мышь... Однако ключевое отличие от работы с обычной системой заключается в том, что пользователю предоставляется лишь иллюзия локальной работы. А именно, пользователь видит лишь «мгновенные снимки» приложения, которое на самом деле функционирует в удаленном облаке, состоящем из специальных серверов.

На локальном компьютере с высокой степенью защиты (как программной, так и аппаратной) функционирует только тонкий клиент, реализующий протокол общения с серверами приложений. Машина изолирована от сети Интернет и не имеет сторонних программ. По требованию пользователя приложения запускаются в облаках, которые могут располагаться внутри организации и тоже не иметь выхода в Интернет. Клиент получает видео-поток, отображающий состояние запущенной программы, и демонстрирует его пользователю. Пользователь, в свою очередь, видит приложение так, будто оно запущенно локально. В качестве элементов взаимодействия с приложением на соответствующий сервер отправляются сигналы нажатий клавиш клавиатуры и мыши, а также относительные координаты курсора. Там они обрабатываются, и обновленное состояние приложения передается пользователю.

Каждое приложение в облаке запускается в отдельной изолированной среде, которая создается с использованием технологии виртуализации. Такой подход совершенствует уровень безопасности и позволяет построить отказоустойчивую систему. Важной особенностью является возможность одновременной работы с приложениями для различных операционных систем (Linux, Windows, Mac OS...) на одном компьютере.

Серверы приложений в облаках могут иметь различные уровни безопасности. Наиболее защищенным должен быть сервер для запуска приложений, обрабатывающих секретные данные: желательно запретить ему доступ в Интернет и ограничить к нему физический доступ сотрудников организации. На рисунке 5 представлена общая схема описанной выше архитектуры.

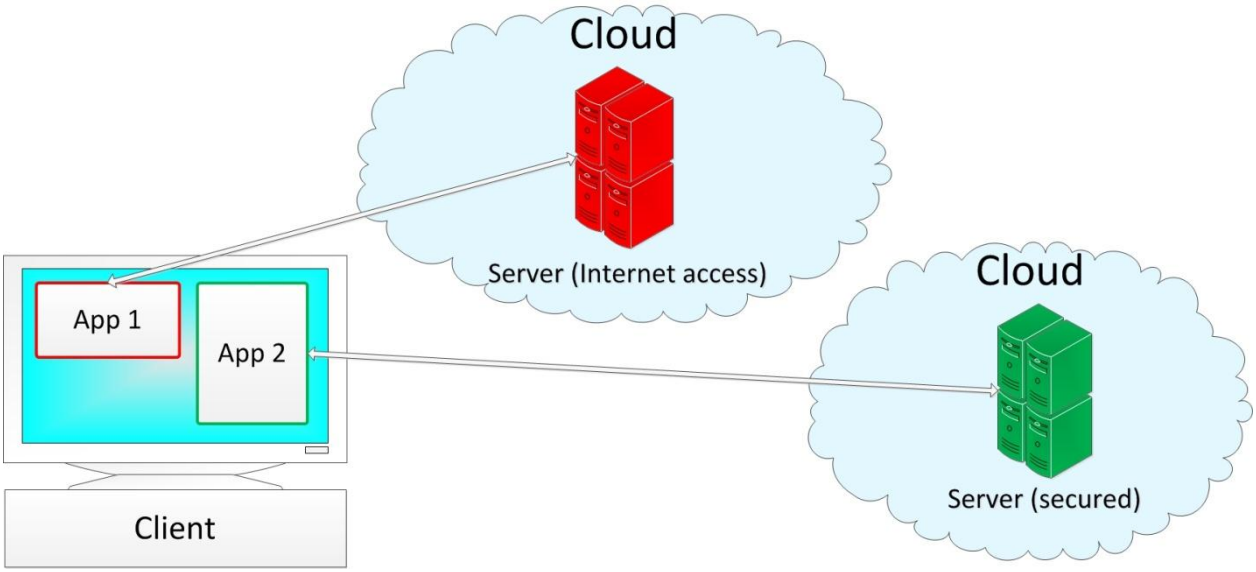

Рис. 5

3.4 Обзор существующих решений

Перед выполнением работ был проведен анализ рынка. В результате долгого поиска не было обнаружено решения, полностью удовлетворяющего нашим целям. Самым близким по функциональности оказался продукт Citrix XenApp.

Citrix XenApp – коммерческое решение, работающее на основе виртуализации, имеющее возможность централизованного управления и реализующее доставку Windows-приложений как сервиса (SaaS) по запросу на любое пользовательское устройство. Для доставки приложений используется проприетарный протокол ICA.

Ниже приведены одни из наиболее интересных функций [4].

- Подписки приложений

  Администраторы публикуют доступные для пользователя приложения, пользователи могут подписаться на использование этих приложений посредством drag’n’drop интерфейса.
- Динамическая доставка

  Пользователь может закачать себе локальную версию приложения. Если на клиентском устройстве не хватает ресурсов для работы приложения (например, заканчивается оперативная память), оно откатывается на сервер, чтобы не было заметно падения производительности.

- Доставка на любое устройство/через любое соединение

  Citrix XenApp может доставлять приложения на операционные системы семейств Windows, Linux, Mac, iOS посредством соединений любого типа: WiFi, Ethernet, 3G, YOTA…
- Поддержка MS Remote Desktop Client
- Высокая доступность

  Доступность приложений оценивается «пятью девятками» (99,999% uptime)

Архитектура и схема работы XenApp представлена ниже [21].

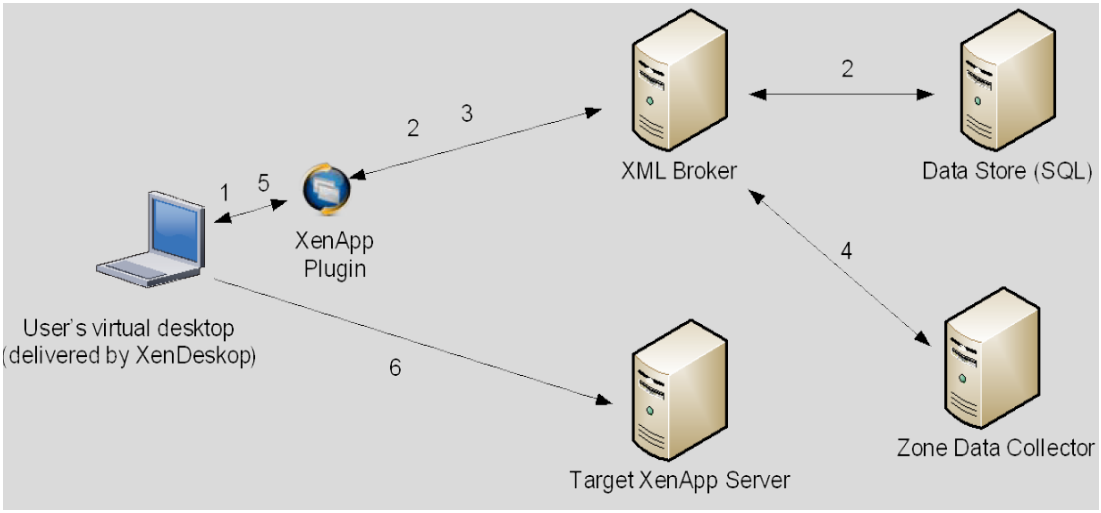

Рис. 6

1. Пользователь с помощью виртуального рабочего стола, предоставляемого XenDetktop, получает доступ к XenApp Plug-in. Этот плагин используется вместе с соответствующим веб-интерфейсом на сервере.

2. Веб интерфейс XenApp Plug-in обращается к XML broker для определения списка доступных приложений. Специальный сервис на стороне XML broker обращается к локальной памяти и определяет набор приложений пользователя. XML broker формирует ответ и передает его обратно.

3. Пользователь кликает на иконку приложения, в результате чего сайт XenApp Plug-In отравляет запрос XML broker, с целью узнать адрес сервера приложений, способного обеспечить пользователя требуемым ПО.

4. Информация об адресе сервера приложений хранится на специальном сервере – Zone Data Collector. XML broker получает необходимые данные от ZDC и переправляет их на сайт XenApp Plug-in

5. Посредством веб-интерфейса XenApp Plug-In передает информацию о выбранном сервере приложений на устройство пользователя в виде ICA-файла.

6. На пользовательском компьютере запускается ICA-файл, привязанный к соответствующему серверу приложений.

3.5 Описание основных технологий

Для построения системы Multi-Cloud Desktop предлагается использовать гипервизор Xen с открытым исходным кодом, а точнее его «облачную версию» Xen Cloud Platform.

3.5.1 Xen Cloud Platform

XCP представляет собой обычный Xen с предустановленной Domain 0 системой (CentOS 5), но с некоторыми дополнительными административными утилитами, организующими работу облака.

В первую очередь XCP состоит из хостов – серверов, которые занимаются виртуализацией. Хосты могут быть объединены в специальные структуры, внутри которых может осуществляться миграция и, за счет этого, балансировка нагрузки, отказоустойчивость и прочее. Такие объединения называются пулами, каждый хост может относиться только к одному пулу. У пулов есть состояние – это аспекты конфигурации самого пула, хостов, SR-ов (Storage Repository), виртуальных машин и др. Состояние пула хранится на каждом из его хостов. Операции в пуле может выполнять лишь один хост, называемый мастером пула (Pool Master). Именно через мастера осуществляется управление виртуальными машинами, принадлежащими пулу, миграцией и всем остальным. Master принимает команды извне и отсылает их Slave-ам (так называются оставшиеся хосты, которые не являются Master), на которых те уже и исполняются.

Запускаемые виртуальные машины хранятся в специальных Storage Repository (SR). Стоит заметить, что SR – лишь абстракция, не имеющая физического воплощения. Метод доступа к SR определяется в структуре Physical Block Device (PBD). PDB может быть смонтированным NFS-хранилищем, подключенным iSCSI-устройством, локальным жёстким диском, отдельным разделом на этом диске и т.д. Хосты осуществляют подключение к SR с помощью драйвера, называемого Storage Manager (SM). Он устанавливает соединение с SR и выдает хосту PBD.

Виртуальные машины тесно связаны с двумя сущностями: Virtual Disk Image (VDI) – отвечает за хранение информации, в самой простой реализации это побайтовая копия диска; Virtual Block Device (VBD) – отвечает за способ доступа к информации, является мостиком между VDI и диском в виртуальной машине, по сути – блочное устройство в гостевой виртуальной машине. Если первый может существовать без ВМ, то второй, являясь её свойством, нет.

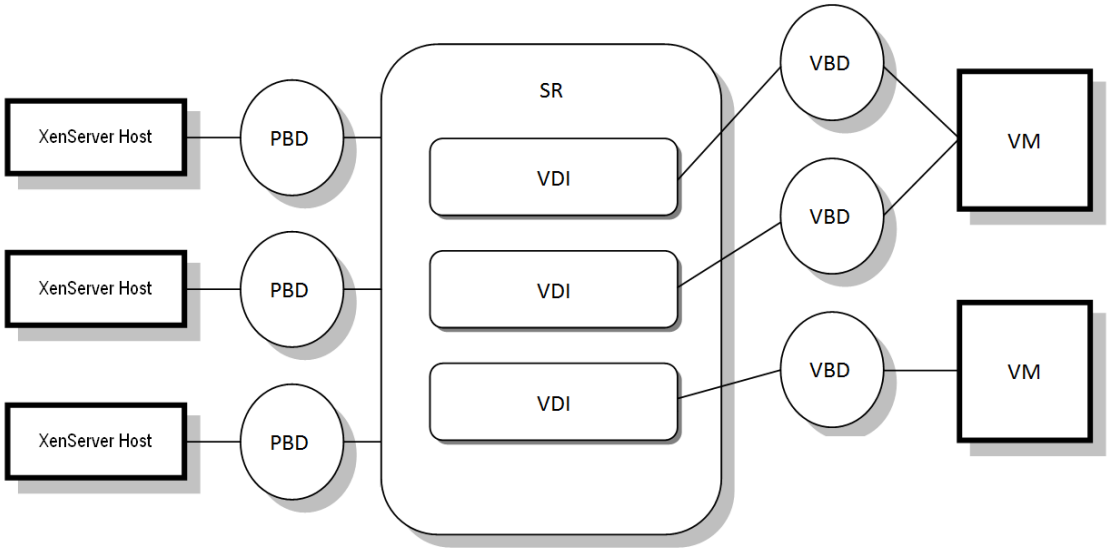

Рис. 7

Рассмотрим основные компоненты XCP:

- xen

Основой всей облачной инфраструктуры является гипервизор Xen. Он управляет виртуальными машинами, распределением ресурсов – всем тем, что находится в ведении менеджеров виртуальных машин.

- xapi

xapi – это мозг XCP. Основной сервис, командующий гипервизором, осуществляющий миграцию и обрабатывающий вызовы API. Контролирует работу виртуальных машин. Именно эта часть XCP отправляет информацию мастеру пула и принимает от него команды. Это могут быть команды как в отношении виртуальных машин, так и в отношении конфигурации хоста: настройка сети, перезагрузка/выключение, монтирование блочных устройств, управление пользователями.

- xe

Мощнейшая консольная утилита для управления облаком – обертка над xapi. Её особенностью является то, что она способна работать по сети. Подключается к мастеру пула с помощью API и после авторизации позволяет управлять пулом почти так же, как бы это делалось локально.

- stunnel

Эта часть системы используется для организации защищенных соединений между Master и Slave, а также между Slave и Slave хостами.

- sqeezed

Данный компонент XCP является демоном, предоставляющим динамическое регулирование памяти. Новым доменам этот демон выделяет максимум доступной памяти, а в случае потребности гипервизора в памяти, уменьшает объем доступной памяти для всех виртуальных машин. Стоит заметить, что администратор может определить две величины: memory_dynamic_max (> 0) и memory_dynamic_min (< физически доступного объема оперативной памяти), определяющие дополнительные рамки на выделение памяти. В результате, при каждом изменении памяти ВМ проверяется формула

$0 \leq \text{memory\_dynamic\_min} \leq \text{target} \leq \text{memory\_dynamic\_max} \leq \text{max RAM}$,

где target – желаемый объём памяти [36].

3.5.2 X11 (X Window System)

X Window System – это открытая кроссплатформенная оконная система, то есть “серверно-клиентское” программное обеспечение, которое позволяет управлять оконным графическим интерфейсом пользователя в распределенных сетях.

X11 – протокол позволяющий запускать приложения на удаленных компьютерах (серверах) с помощью X-Client и отображать эти приложения на локальном компьютере пользователя (клиенте) с помощью X-Server.

Рассмотрим место X Window System в MCD на следующем примере:

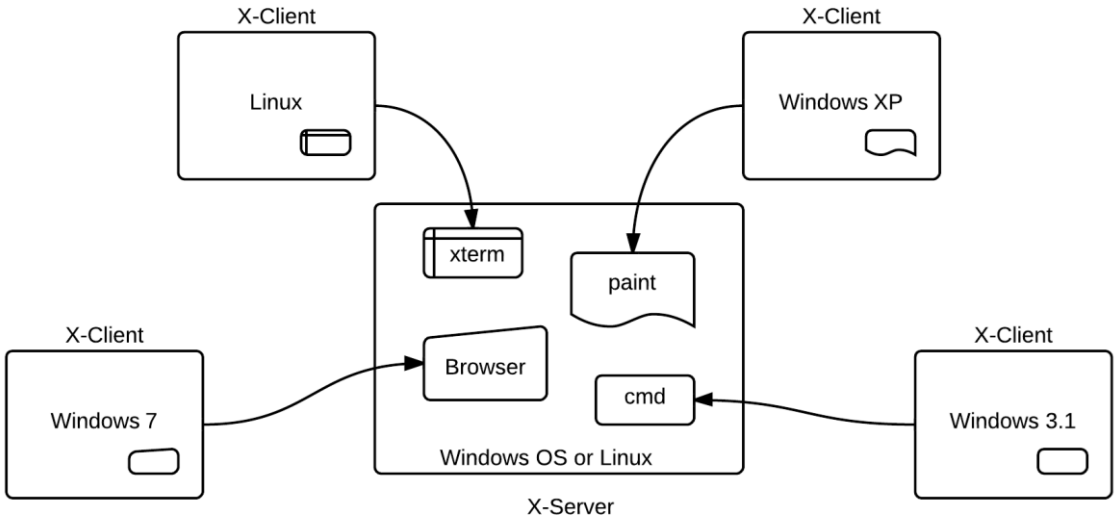

Рис. 8

В нашей системе X11 играет роль протокола доставки приложений. На клиентской машине стоит X-Server, который занимается формированием окон, а также обработкой событий мыши и клавиатуры и их передачей на X-Client. Серверная часть X Window System выполняется в нашем облаке на виртуальной машине, из которой мы хотим запустить приложение. На стороне X-Client мы запускаем необходимое приложение, транслируем видеопоток его фреймов на X-Server и пробрасываем события клавиатуры и мыши в буфер сообщений этого приложения. После исполнения приложением событий, пользователю приходит новый фрейм.

На рисунке 8 в качестве передаваемых приложений выступают Linux-овый xterm, Paint из Windows XP, командная строка Windows 3.1, и некоторый абстрактный браузер Windows 7. Важно отметить, что может осуществляться доставка любых приложений на клиентскую машину с любой ОС.

На текущий момент по протоколу X11 возможно передавать окна с Linux на Linux (на нем основана графическая подсистема), с Linux на Windows (существуют X-серверы для Windows, например, MobaXTerm). Однако не реализованы варианты X11, заключающиеся в передаче окон из MS Windows т.к. в основе этой

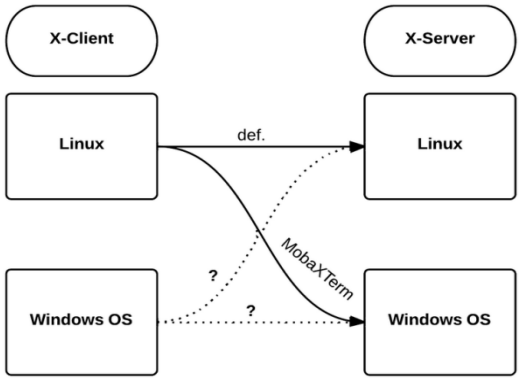

Рис. 9

операционной системы лежит другой механизм отрисовки графики приложений (см. Рис. 9) [35].

Поэтому для создания единой системы передачи графики приложений необходимо реализовать недостающие ветви протокола.

3.6 Постановка задачи

3.6.1 Основные задачи

На этапе планирования работ по созданию системы Multi-Cloud Desktop было выделено три ключевых задачи, без решения которых реализация проекта не представляется возможной.

3.6.1.1 Конфигурация тестовой системы

Первостепенной задачей является создание и настройка требуемой облачной инфраструктуры. В результате работ, проделанных в этом направлении, была собрана тестовая система следующей минимальной конфигурации:

Тестовый стенд состоит из двух идентичных хостов с предустановленным Xen Cloud Platform 1.6. Хосты соединены по сети Ethernet и объединены в пул. Для корректной работы системы необходимо один из хостов пула объявит мастером пула – именно он будет управлять виртуальными машинами этого пула, осуществлять миграцию, сбор статистики и др. Управление осуществляется с помощью утилиты xe – как локально, так и по сети. Каждому из хостов соответствует сконфигурированный локальный Storage Repository, в котором хранятся образы гостевых систем хоста. Установка ОС в непривилегированные домены осуществляется либо с установочного диска, вставленного в привод дисков хоста, либо по сети.

3.6.1.2 Автозапуск приложений по требованию

XCP может легко управлять запуском виртуальных машин, однако залезть внутрь гостевой ОС для запуска какого-либо приложения по требованию пользователя он не умеет. Поэтому было решено реализовать модуль системы, который бы занимался внедрением автозапуска необходимого приложения в образ операционной системы, предварительно укомплектованных некоторым требуемым набором ПО. Таким образом для запуска приложения, нам достаточно будет запустить модифицированную виртуальную машину.

Место модуля в системе таково: пользователь отправляет запрос на запуск приложения в облако, где осуществляется поиск удовлетворяющей виртуальной машины, в которую внедряется автозапуск нужного приложения.

Различные методы внедрения автозапуска и наша реализация этого модуля могут быть найдены в соответствующем отчете [30].

3.6.1.3 Передача графики. Модификация протокола X11

Важной частью MCD является подсистема доставки окон. Было принято решение использовать для этих целей протокол X11, однако он обладает одним существенным не достатком – не умеет доставлять приложения ОС Windows. Поэтому была поставлена задача реализации требуемого функционала X Window System для операционной системы Windows.

3.6.1.4 Протокол управления MCD

Имея все необходимые модули в наличие, нужно уметь еще правильно ими воспользоваться для корректной работы системы в целом. Таким образом, требовалось спроектировать протокол взаимодействия клиента и сервера, а также внутри серверное взаимодействие его компонентов.

3.6.2 Конкретная задача

В данной курсовой работе рассматривается решение задачи проектирования системы Multi-Cloud Desktop и протокола её управления.

4. Решение задачи создания общей архитектуры протокола управления MCD

4.1 Определение

Протокол передачи данных – набор соглашений интерфейса логического уровня, которые определяют обмен данными между различными программами.

4.2 Ключевые особенности построения системы

Рассмотрим подробнее архитектурные особенности системы.

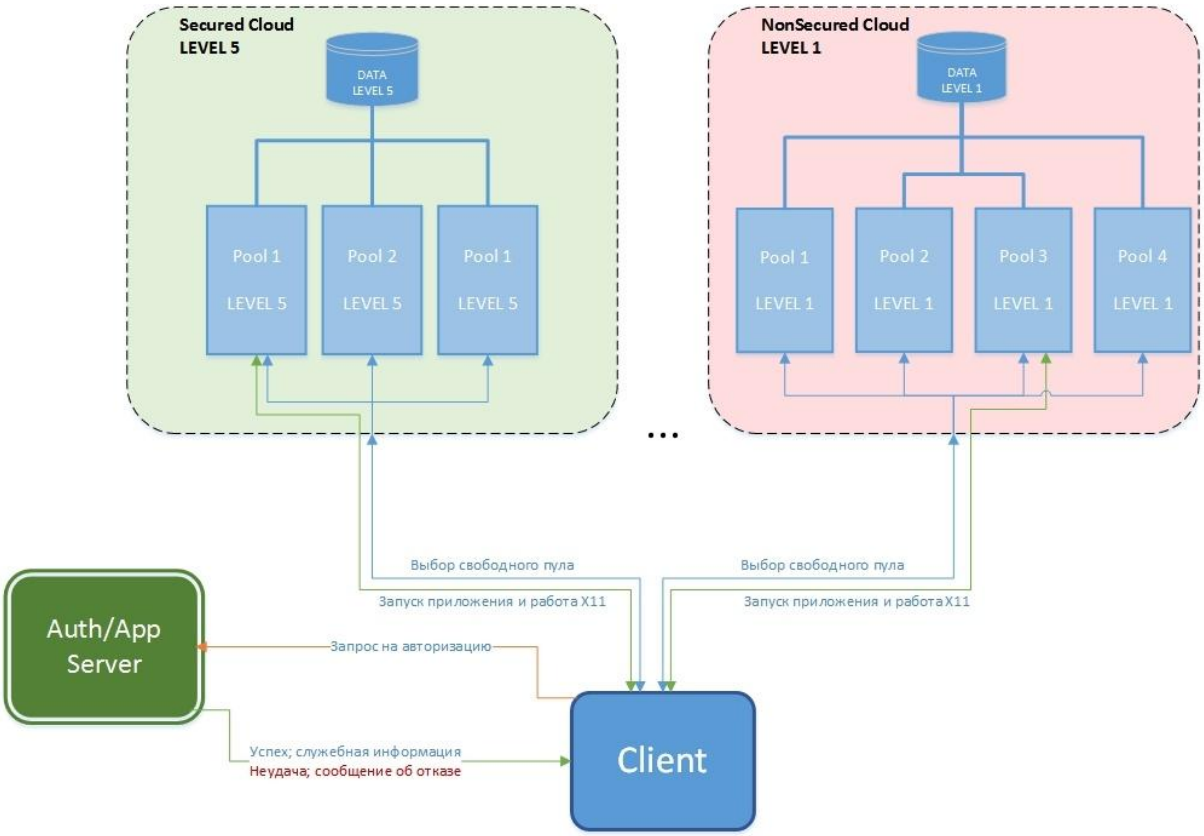

Рис. 4

Как уже упоминалось ранее, основная идея заключалась в максимальной изоляции виртуальных машин и, соответственно, приложений между собой. В целях безопасности организуются физически разные облака, отличающиеся уровнями доступа (на рисунке условно названы LEVEL и пронумерованы, меньшее значение соответствует более низкому уровню безопасности). Каждое облако может состоять из нескольких пулов (логически объединенных хостов). Для одного уровня доступа представляется целесообразным создать сетевое хранилище данных.

Для хранения идентификационных данных пользователей, данных о приложениях и прочей служебной информации используется специальный сервер (Auth/App Server). Он

контролирует процедуру авторизации пользователей и выдает доступные им идентификаторы приложений.

Из-за повышенных требований к безопасности использованные виртуальные машины могут удаляться каждый раз по окончании сеанса работы пользователя с приложением. Однако возможен и сценарий, когда ВМ будут сохраняться в специально отведенном месте для дальнейшего использования тем же пользователем.

Если приложению, которое использует пользователь, необходимо сохранить что-то на диск (например, закладки браузера), нельзя позволять скидывать данные в локальный диск виртуальной машины, т.к. он может удаляться по завершении каждого сеанса работы. Но решение такой проблемы может быть найдено, к примеру, следующим образом. На сетевом хранилище соответствующего уровня безопасности выделяется место для каждого пользователя, и этот «кусочек» присоединяется в новую ВМ при ее старте. Все пути сохранения данных направляются в этот примонтированный диск. Таким образом, после завершения работы с приложением и уничтожения виртуальной машины постоянные данные пользователя сохранятся в хранилище.

Перед тем, как описывать протокол взаимодействия клиента и системы, необходимо отметить, что все взаимодействие реализуется с помощью механизмов шифрования, например протоколов TLS (SSL).

4.3 Абстрактный протокол

Взаимодействие пользователя с системой логически делится на два ключевых этапа: авторизация и работа с приложением. Рассмотрим схему взаимодействия компонент системы во время каждого из них.

4.3.1 Авторизация и получение списка доступных приложений

Авторизация пользователя осуществляется, например, с помощью логина и пароля. Введенные данные отправляются на специально выделенный сервер (Auth/App Server, рис. 10), на котором производится проверка идентификационных данных и, в случае успешного входа в систему, поиск прав доступа данного пользователя. В случае некорректных входных данных, пользователю отправляется сообщение об ошибке.

В результате успешной авторизации, пользователю на тонкий клиент возвращается список доступных ему приложений с сопутствующей служебной информацией: имя приложения (отображается пользователю), идентификатор приложения, уровень доступа (для работы с документами разного уровня секретности; в этом списке может быть

несколько MS Office Word – один для документов класса Top Secret, другой – для повседневной рутины и т.п.), информация для запуска приложения в облаке (полный путь к приложению; ОС, в которой оно будет запущено; список IP-адресов мастеров пулов, в которых приложение может быть запущено).

Минимальный набор информации, отображаемый клиентским приложением пользователю, состоит из имени приложения (часть которого составляет имя соответствующей данному программному обеспечению ОС) и уровня его доступа.

4.3.2 Работа с приложением

Работа пользователя с приложением может быть разделена на следующие этапы:

1. Выбор пула для запуска приложения
2. Запрос приложения
   Запуск приложения на одном из хостов пула
3. Работа с приложением по протоколу X11
4. Завершение работы с приложением

4.3.2.1 Выбор пула для запуска приложения

Если приложение может быть запущено на более чем одном пуле, то система должна определить, в каком из них будет лучше запустить приложение с точки зрения оптимального распределения нагрузки на хосты – своеобразный высокоуровневый Load Balancing. Осуществляется это посредством запроса у мастеров пулов некоторого коэффициента загруженности пула, который зависит от количества запущенных в нем виртуальных машин; загруженности процессоров, объема свободной оперативной памяти и прочих характеристик. В случае отсутствия ресурсов, достаточных для запуска новой виртуальной машины, пользователю отправляется соответствующее сообщение и предлагается либо приостановить процесс запуска требуемого приложения, либо дождаться освобождения ресурсов для его завершения.

Получив от мастеров пулов значения коэффициентов их загруженности, клиент выбирает пул с минимальным из них и продолжает дальнейшую работу только с выбранным пулом.

4.3.2.2 Запрос приложения и его запуск

Запрос приложения осуществляется посредством отправки мастеру пула сообщения, содержащего идентификаторы требуемого приложения и служебную информацию о

клиенте (минимальный объем этой информации – IP-адрес клиента). Параллельно на клиенте выполняется запуск X-Server.

Мастер пула, получив запрос на доставку приложения от клиента, осуществляет поиск свободной виртуальной машины с требуемой ОС. В случае её отсутствия инициируется клонирование соответствующей эталонной укомплектованной виртуальной машины, а клиенту отправляется сообщение об ожидании. Определив свободную виртуальную машину, мастер, используя разработанный нами механизм, модифицирует её файловую систему, добавляя автозапуск требуемого приложения, X-Client и при необходимости подключение сетевого диска пользователя [30]. Параллельно выполняется очередное копирование эталонной ВМ для последующего быстрого доступа к ней.

4.3.2.3 Работа с приложением по протоколу X11

В результате запуска модифицированной виртуальной машины по протоколу X11 осуществляется соединение между X-Server на стороне клиента и X-Client на стороне ВМ с запущенным приложением. Пользователю отправляется сообщение об успешном запуске приложения и готовности работы. В случае, если виртуальная машина по какой-либо причине была не запущена, пользователю отправляется сообщение о неудачном запуске ВМ.

Работа пользователя с приложением реализована на основе протокола X11: X-Server захватывает события мыши и клавиатуры, отправляет их на X-Client, где они буферизуются и применяются к приложению. X-Client в свою очередь отправляет на X-Server фреймы приложения, которые формируются на клиентской машине в видео-поток [35].

По завершению работы с приложением – его закрытию – корректно завершается сессия X11, и мастеру пула отправляется сообщение о завершении работы. Мастер пула выполняет завершение работы виртуальной машины и удаляет её образ.

5. Итоги и результаты

На первом этапе были детально изучены технология виртуализации и основные принципы построения отказоустойчивых систем. Также был проведен обзор существующих решений отказоустойчивых систем. Была собрана тестовая отказоустойчивая система с использованием гипервизора Xen. Были проведены различные тесты.

Второй этап был посвящен изучению применения технологии виртуализации в области безопасности. Сделан обзор на Qubes-OS – одного из развивающихся представителей сферы.

Основные идеи, заложенные в Qubes-OS, были переосмыслены, дополнены и использованы при проектировании архитектуры системы Multi-Cloud Desktop, которая решает задачу обеспечения безопасности рабочего пространства пользователя в условиях повышенного риска потери конфиденциальной информации.

На следующем этапе разработаны логика работы системы Multi-Cloud Desktop, несколько её модулей и протокол взаимодействия компонентов системы, учитывающий специфику рассматриваемой области, связанной с обеспечением повышенной безопасности рабочего пространства; собран тестовый стенд – база для создания Proof of concept.

Дальнейшими направлениями развития проекта являются задачи создания GUI, внедрения модуля разделения привилегий на основе мандатного разделения доступа и шифрования данных, передаваемых между клиентом и облачной инфраструктурой, а также интеграция всех модулей в единую систему.

Список литературы

[1] **AMD Virtualization** [В Интернете] / авт. AMD // AMD. - http://sites.amd.com/us/business/it-solutions/virtualization/Pages/virtualization.aspx#2.

[2] **Architectural principles for virtual computer systems** [Report] / auth. Goldberg Robert P.. - Harvard : Harvard University, 1973.

[3] **Attacks on Virtual Machine Emulators** [Отчет] / авт. Ferrie Peter. - [б.м.] : SYMANTEC.

[4] **Citrix XenApp Comparative Feature Matrix** [В Интернете] / авт. Citrix // Citrix. - http://www.citrix.com/content/dam/citrix/en_us/documents/products/Citrix_XenApp_6.5_Comparative_Feature_Matrix.pdf.

[5] **Citrix XenApp Overview** [В Интернете] / авт. Citrix // Citrix. - http://www.citrix.com/products/xenapp/overview.html.

[6] **Current OS Compatibility** [В Интернете] / авт. Xen // Xen Wiki. - http://wiki.xensource.com/xenwiki/OSCompatibility.

[7] **Detection of VM-Aware Malware** [Журнал] / авт. David Yu Zhu Erika Chin. - Berkley : University of California, 2007 г..

[8] **Fault-tolerance and fault-intolerance: complementary approaches to reliable computing** [Конференция] / авт. Avizienis Algirdas. - Los Angeles : University of California, CS department, 1975.

[9] **How Does Xen Work?** [В Интернете] / авт. Xen // Xen Project. - http://www.xen.org/files/Marketing/HowDoesXenWork.pdf.

[10] **IBM 700/7000 series** [В Интернете] / авт. Wikipedia // Wikipedia. - https://en.wikipedia.org/wiki/IBM_700/7000_series.

[11] **IBM CP-40** [В Интернете] / авт. Wikipedia // Wikipedia. - http://en.wikipedia.org/wiki/IBM_CP-40.

[12] **Intel 64 and IA-32 Architectures Software Developper's Manual** [Книга] / авт. Intel. - T. 1 Basic Architecture.

[13] **Intel Virtualization Technology: Hardware Support for Efficient Processor Virtualization** [Статья] / авт. Gil Neiger Amy Santoni, Felix Leung, Dion Rodgers, Rich Uhlig // Intel Technology Journal. - [б.м.] : Intel, 206 г.. - 03 : Т. 10.

[14] **Methods for Virtual Machine Detection** [В Интернете] / авт. Omella Alfredo Andres // S21sec. - 2006 г.. - http://charette.no-ip.com:81/programming/2009-12-30_Virtualization/www.s21sec.com_vmware-eng.pdf.

[15] **MIT Exokernel Operating System** [В Интернете] / авт. MIT CSAIL // Parallel & Distributed Operating Systems Group. - http://pdos.csail.mit.edu/exo/.

[16] **On the Cutting Edge: Thwarting Virtual Machine Detection** [В Интернете] / авт. Tom Liston Ed Skoudis. - http://handlers.sans.org/tliston/ThwartingVMDetection_Liston_Skoudis.pdf.

[17] **QEMU** [В Интернете] / авт. Wikibooks // Wikibooks. - http://en.wikibooks.org/wiki/QEMU.

[18] **QEMU/Images** [В Интернете] / авт. Wikibooks // Wikibooks. - http://en.wikibooks.org/wiki/QEMU/Images.

[19] **Qubes-OS wiki** [В Интернете] / авт. Qubes OS Team // Qubes-OS. - http://www.qubes-os.org/trac.

[20] **Reasons to Use Virtualization** [Online] / auth. Oracle // Oracle documentation. - Oracle. - http://docs.oracle.com/cd/E26996_01/E18549/html/BHCJAIHJ.html.

[21] **Reference Architecture–Based Design for Implementation of Citrix XenDesktop on Cisco Unified Computing System, Citrix XenServer, and NetApp Storage** [В Интернете] / авт. Cisco. - 2010 г.. - http://www.cisco.com/en/US/docs/solutions/Enterprise/Data_Center/Virtualization/ucs_xd_xenserver_ntap.pdf.

[22] **Remus: High Availability via Asynchronous Virtual Machine Replication** [Журнал] / авт. Brendan Cully Geoffrey Lefebvre, Dutch Meyer, Mike Feeley, Norm Hutchinson, Andrew Warfield. - Vancouver : The University of British Columbia, Department of Computer Science.

[23] **SIMMON** [В Интернете] / авт. Wikipedia // Wikipedia. - http://en.wikipedia.org/wiki/SIMMON.

[24] **System Documentation for Developers** [В Интернете] / авт. Joanna Rutkowska Rafal Wojtczuk // Qubes-OS. - Qubes-OS. - http://files.qubes-os.org/files/doc/arch-spec-0.3.pdf.

[25] **System z PR/SM** [В Интернете] / авт. IBM // IBM Systems Software Information Center. - http://publib.boulder.ibm.com/infocenter/eserver/v1r2/index.jsp?topic=%2Feicaz%2Feicazzlpar.htm.

[26] **The History of Virtual Memory** [В Интернете] / авт. P. Christy // eHow Tech. - http://www.ehow.com/facts_6905251_history-virtual-memory.html.

[27] **Xen Cloud Platform** [В Интернете] / авт. Xen // Xen Project. - 2009 г.. - http://www.xen.org/files/XenCloud/reference.pdf.

[28] **Xen Cloud Platform: Как поселить чужую VM в новый дом** [В Интернете] // Habrahabr. - http://habrahabr.ru/post/177679/.

[29] **XenCenter** [В Интернете] / авт. Citrix Community // Cirix Developer Network. - http://community.citrix.com/display/xs/XenCenter.

[30] **Безопасное рабочее пространство пользователя MCD. Модуль внедрения автозапуска приложений** [Отчет] / авт. Р.С. Одеров. - СПб : СПбГУ, 2013.

[31] **Виртуализация. Технология, изменяющая индустрию IT** [В Интернете] / авт. Intel. - world-it-planet.org/upload/VTx.pdf.

[32] **Отказоустойчивая виртуализация на основе гипервизора Xen** [Конференция] / авт. Одеров Р.С. Серко С.А., Чередник К.Е. // Материалы конференции "Актуальные проблемы организации и технологии защиты информации". - СПб : ИТМО, 2012. - Т. 2.

[33] **Отказоустойчивая виртуализация на основе гипервизора Xen** [Конференция] / авт. Одеров Р.С. Серко С.А., Чередник К.Е. // Материалы научной сессии МИФИ-2013. - Москва : НИЯУ МИФИ, 2013.

[34] **Построение отказоустойчивой (fault tolerant) системы** [В Интернете] // Habrahabr. - http://habrahabr.ru/post/118496/.

[35] **Разработка и реализация X-клиента для платформ ОС Windows** [Отчет] / авт. А.О. Малыгин. - СПб : СПбГУ, 2013.

[36] **Цикл статей "Xen Cloud Platform в условиях предприятия"** [В Интернете] // Habrahabr. - http://habrahabr.ru/post/104025/, http://habrahabr.ru/post/104881/, http://habrahabr.ru/post/105262/, http://habrahabr.ru/post/105568/.