NYKOLAYCHUK YA.N., IVASIEV S.V., YAKYMENKO I.Z., KASIANCHUK M.N. Method of factorization of multi-digital numbers based on the properties of quadratic remainders in the system of residual classes

This work is devoted to develop of advanced method of factorization of multi-bit numbers based on Fermat's theorem with using of the system of residual classes, This method is excluded the operation of squaring and, besides that, arithmetic operations are performed on numbers which are smaller than the selected module. Last one allows to shifted zone of bit computing resources on several orders to deeper side and replace the operation of finding the square root, which is caused of computational complexity of the Fermats' algorithm onto generating a binary key of factorization.

УДК 004.94

Коваленко В.Ю., Костюк Д.А.

ВИРТУАЛИЗОВАННАЯ ФЕРМА ДЛЯ ТЕСТИРОВАНИЯ И ДЕМОНСТРАЦИИ ПРИЛОЖЕНИЙ ПЛАТФОРМЫ ANDROID С ВЕБ-ДОСТУПОМ

Введение. На сегодняшний день клиент-серверные приложения с веб-интерфейсом приобрели широчайшее распространение и по ряду направлений стали заменять классические настольные приложения. К числу причин следует отнести наличие веб-браузера на всех платформах и архитектурах, а также достижение современными браузерами достаточной производительности исполнения кода JavaScript, на котором строится клиентская часть веб-приложений. Универсальная доступность (в т.ч. на платформах с сенсорным интерфейсом и других так называемых «слабых клиентах», т.е. на мобильных и портативных устройствах, нацеленных преимущественно на использование облачных сервисов) делает работу через браузер наиболее удобной точкой входа для конечного пользователя приложения/сервиса. Особенно актуально это, когда основная вычислительная нагрузка ложится на сервер и/или другие узлы в сети, а сам клиент служит лишь для управления и доступа к ресурсам (обратный случай, к сожалению, на текущий момент неэффективен, так как реализация на JavaScript подразумевает заметно бóльшие накладные расходы по сравнению с традиционными языками серверных платформ).

Одной из задач, способных получить существенный выигрыш при их клиент-серверной реализации с использованием частного облака, является тестирование и отладка мобильных приложений.

Проблему тестирования и отладки программного кода в условиях сильной фрагментации целевых аппаратных платформ нельзя назвать новой, однако на сегодняшний день мобильная платформа Android является одной из наиболее фрагментированных. Разработчику необходимо держать на своей машине набор образов для эмулятора с различными версиями операционной системы и запускать их по очереди. При этом, даже если штатные средства разработки Android

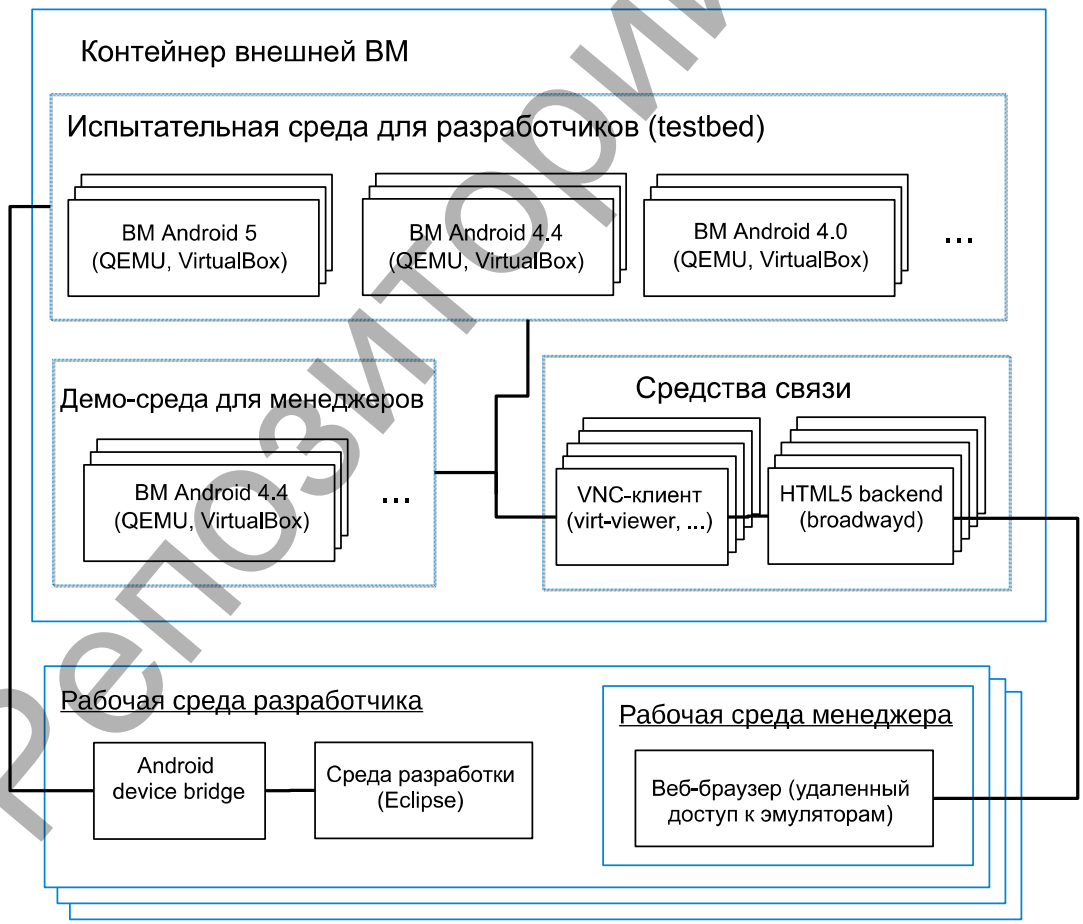

Рис. 1. Структура тестовой фермы Android-приложений

Коваленко Владимир Юрьевич, *старший преподаватель кафедры ЭВМиС Брестского государственного технического университета.*
Костюк Дмитрий Александрович, *к.т.н., доцент кафедры ЭВМиС Брестского государственного технического университета.*
Беларусь, БрГТУ, 224017, г. Брест, ул. Московская, 267.

предоставляли бы удобный интерфейс для одновременного запуска нескольких экземпляров эмулятора для тестирования программы, низкая производительность эмуляции, связанная с разницей процессорных архитектур, при множественном запуске эмулятора негативно сказалась бы на производительности и эффективности разработки.

При коллективной разработке иногда оказывается, что методы локальной эмуляции совершенно не подходят. Периодически требуется коллективный доступ к экземпляру тестового приложения, а также и демонстрационный доступ к промежуточным сборкам.

1. Концепция тестовой фермы Android-приложений. Очевидным выходом является развертывание тестовой фермы на сервере и доступ к ней по сети (в нашем случае – с использованием веб-доступа через интранет-ресурс предприятия). Благодаря использованию открытых и взаимозаменяемых средств разработки Android, построение альтернативной инфраструктуры дает такие преимущества, как экономия вычислительных ресурсов машин разработчиков, возможность централизованного автоматического управления эмуляторами, удобство демонстрации работы запущенных в них программ, а также, для ряда задач, возможность замены штатного эмулятора более производительной виртуальной машиной, которая осуществляет нативную эмуляцию аппаратной платформы, идентичной с хост-системой. В настоящей работе мы представляем вариант организации такого решения, разработанный для нужд конкретного предприятия, однако он достаточно универсален для того, чтобы представлять более широкий практический интерес.

2. Архитектура фермы. Общая структура разработанной фермы представлена на рис. 1. Как и в случае традиционного стека инструментов Android-разработки, приложение тестируется на различных версиях ОС Android, запущенных в эмуляторах. Однако для удобства развертывания фермы и ее миграции набор эмуляторов в нашем случае находится внутри ещё одной внешней виртуальной машины (ВМ), т.е. используется так называемая вложенная (nested) виртуализация.

В качестве внешней ВМ может выступать либо система виртуализации VirtualBox, либо коммерческая система VMWare, в зависимости от уже существующей инфраструктуры предприятия, в которую должна быть интегрирована ферма (совмещение двух систем виртуализации в сети может оказаться нецелесообразным по организационным причинам). Число вложенных ВМ (т.е. эмуляторов, содержащих различные версии ОС Android) варьируется в процессе эксплуатации [2]. В качестве вложенной ВМ в предлагаемом решении можно использовать QEMU или VirtualBox (выбор зависит от внешней ВМ, совместно с которой должна работать аппаратно-ускоренная вложенная виртуализация).

Как уже упоминалось, штатный эмулятор Android SDK имеет проблемы с производительностью, меньше подходит для самостоятельного развертывания и использования в демонстрационных целях. По этой причине некоторые разработчики используют сборки Android для x86-совместимых систем [1], запущенные в одной из нативных систем виртуализации (без эмулирования другой процессорной архитектуры). Такой подход существенно снижает вычислительную нагрузку на эмуляцию, а также сохраняет все преимущества и удобства, предоставляемые серверными и десктопными системами виртуализации по части автоматизации, клонирования и встраивания в инфраструктуру предприятия.

3. Образы вложенных ВМ. Кроме веб-доступа к эмулятору, универсального как для разработчиков, так и для менеджеров, разработчик взаимодействует с вложенной ВМ через стандартный интерфейс ADB (для установки приложений и выполнения сеансов отладки кода), а также имеет простой веб-интерфейс, позволяющий клонировать одну из вложенных ВМ.

В составе фермы имеются эталонные образы вложенных ВМ с установленной ОС Android различных версий (рис. 2). Эти образы сконфигурированы и настроены для максимально быстрого запуска новой машины. Для тестирования разрабатываемого ПО клонируется один из эталонных образов. Именно на эти клонированные образы ВМ загружается тестируемое и демонстрационное ПО. Развертывание и отладка программы в клонированной машине выполняется через ADB, а управление и пользовательское взаимодействие с ней – через веб-интерфейс. При необходимости демонстрации ПО нужные образы дополнительно клонируются для демо-среды, доступной через такой же веб-интерфейс, как и образы для разработки.

ОС Android, установленные в эталонных образах, преимущественно скомпилированы для архитектуры x86. Тесты показали, что в этом случае накладные расходы на вложенную виртуализацию (при совместимом сочетании внешней и вложенной ВМ) незначительны и составляют единицы процентов.

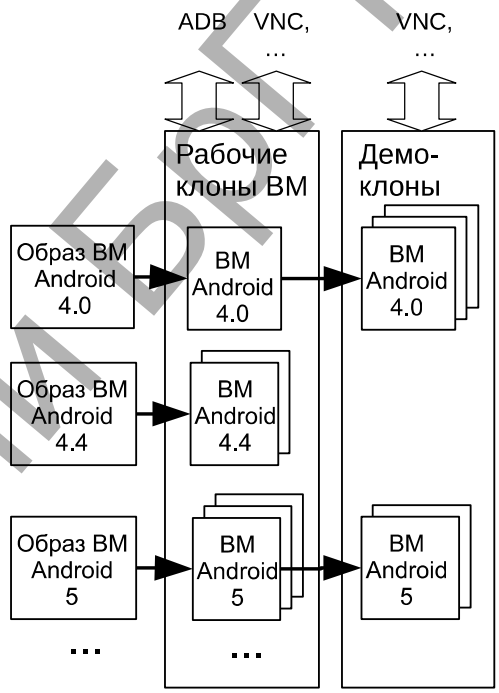

Рис. 2. Клонирование образов ВМ

Технически управление образами обеспечивается достаточно простыми служебными скриптами, которые по требованию выполняют клонирование эталонных образов, а также запуск и остановку рабочих.

4. Организация веб-доступа к ВМ. В зависимости от того, что именно используется в составе фермы качестве вложенных ВМ, доступ пользователей – как разработчиков, так и менеджеров – к эмулятору может осуществляться по протоколу VNC, SPICE или RDP (рис. 2).

Пользователи обращаются к нужной ВМ через интернет-страницу внутреннего веб-ресурса предприятия. Первоначально для соединения с эмулятором через веб-среду мы планировали использование удаленных клиентов на языке JavaScript, которые успешно применяются нами в других проектах. Однако в процессе тестирования был сделан выбор в пользу доступа через GTK-бэкенд broadway – подсистему отрисовки на языке HTML 5, встроенную в последние версии библиотеки GTK 3. Структура решения, на котором построен веб-доступ к ферме, показана на рис. 3.

На стороне клиента доступ к эмуляторам осуществляется через веб-браузер, который взаимодействует с GTK-бакэндом broadway, а broadway отрисовывает окна избранных GTK 3-приложений на объекте canvas HTML 5. Графическая программа, использующая библиотеку GTK, запускается в headless-режиме, т.е. только для сетевого доступа. Для этого должны быть запущены системные демоны broadwayd, каждый из которых занимает отдельный сетевой порт, а графической программе при запуске передается дополнительный набор переменных, которые считываются библиотекой GTK и таким

образом определяют режим отрисовки. В результате код запуска системного демона broadway и приложения, его использующего, выглядит следующим образом:

```
broadwayd :1 &
GDK_BACKEND=broadway          UBUNTU_MENUPROXY=
LIBOVERLAY_SCROLLBAR=0    BROADWAY_DISPLAY=:1
VNCViewer
```

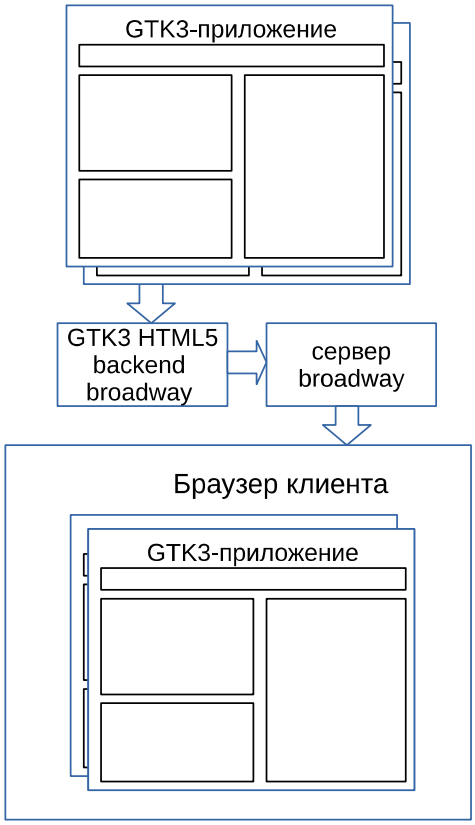

Рис. 3. Принцип работы broadway

Роль графической программы, запущенной внутри внешней ВМ и выполняющей подключение к содержимому вложенной ВМ, играет один из VNC- либо RDP-клиентов, написанных с использованием GTK 3 (в примере – клиент VNCViewer). Технология broadway в свою очередь транслирует пользовательский сеанс доступа в браузер разработчика или менеджера, находящийся за пределами внешней ВМ (рис. 4).

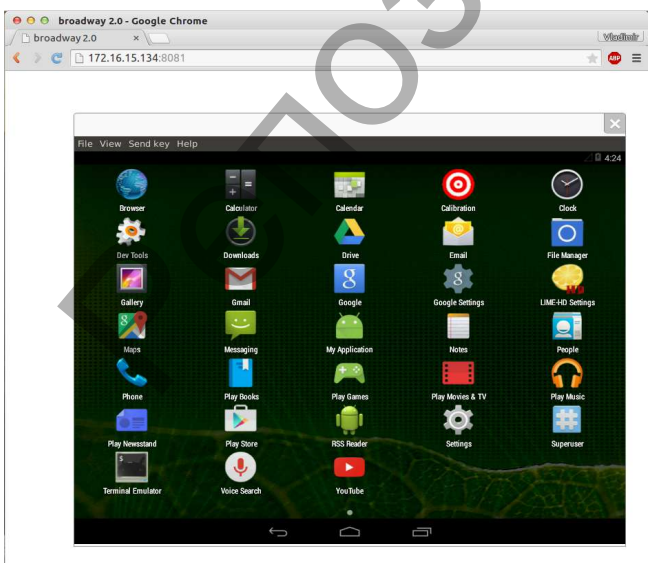

Рис. 4. Окно эмулятора Android в браузере

На выбор подобной архитектуры веб-доступа повлияли два соображения: лучшая производительность кода (которая будет рассмотрена ниже), а также отсутствие необходимости в дополнительных компонентах на компьютере пользователя, которые транслировали бы соединение TCP в веб-сокеты [3].

Что касается дополнительных компонентов на стороне клиента, которые были бы необходимы при использовании VNC-клиента на JavaScript, встроенного в HTML-страницу, их необходимость диктуется тем фактом, что код на JavaScript, выполняемый веб-браузером, не имеет доступа к протоколу TCP, используемому для удаленного доступа, и потому нуждается в приложении-прокси, запущенном на стороне клиента и транслирующем трафик TCP в веб-сокеты.

5. Тестирование производительности доступа к ферме. Очевидно, причиной лучшей производительности выбранного решения веб-доступа в сравнении с VNC-клиентами на языке JavaScript является то, что и узкоспециализированный веб-сервер, входящий в состав broadway, и VNC-клиент, запущенный внутри внешней ВМ – оба эти компонента написаны на языке C. Однако этот код выполняется на сервере, тогда как использование VNC-клиента на языке JavaScript увеличивает вычислительную нагрузку на клиентской машине. В свою очередь, при установке соединения broadway также передает на клиентскую машину код на JavaScript, выполняемый веб-браузером для приема, передачи данных и их отрисовки.

Для оценки реального влияния broadway на производительность разработанной фермы было отдельно выполнено тестирование производительности веб-доступа, для чего был сконфигурирован сервер на основе стандартной версии ОС GNU/Linux Ubuntu 15.04, процессора AMD FX-8320 и ОЗУ объемом 16 ГБ. Выбор ОС обоснован наличием в ее составе компонент, необходимых для работы broadway (библиотеки GTK версии 3.8 и выше, а также использующего ее ПО). Для теста на этом сервере были запущены одновременно и клиентская, и серверная части системы, в результате чего достаточно высокая ресурсоемкость графического интерфейса выбранной ОС воздействовала одновременно на обе части и т.о. не влияла на результат сравнения.

В качестве альтернативы используемому решению по организации веб-доступа для сравнения был выбран JavaScript-клиент noVNC. В качестве показателя нагруженности был использован параметр «load average», получаемый с помощью htop – как наиболее распространенный и общепринятый показатель оценки загруженности серверов.

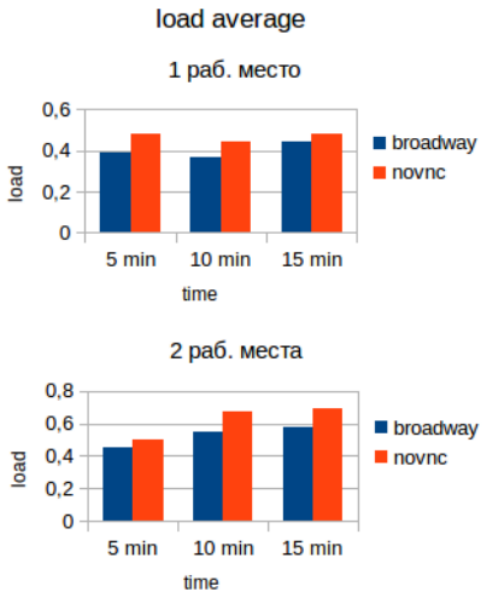

Рис. 5. Сравнение производительности broadway и noVNC

Результаты сравнения видны на рис. 5. Анализ показывает преимущества broadway по части нагрузки, причем с ростом количества мест эта разница увеличивается и оказывается существенной, если число виртуальных рабочих мест достигнет десятков сотен.

Меньшую загруженность сервера при использовании broadway, чем при использвоании схем с протоколом VNC, можно объяснить несколькими факторами: отсутствием необходимости в отдельном VNC-сервере, отсутствием операций, связанных со сжатием и кодированием потока для VNC, более оптимальным (за счет узкой специализации) кодом сервера, отдающим контент. Потребление ОЗУ в случае broadway так же ниже (экономия составляет порядка 30–40 МБ на одно виртуальное рабочее место).

Что касается нагрузки на клиентскую машину и объема генерируемого трафика, то в этом отношении оба подхода демонстрируют паритет (обе величины незначительны). Однако здесь преимуществом broadway оказывается полное отсутствие артефактов изображения, которые могут появляться при использовании VNC.

СПИСОК ЦИТИРОВАННЫХ ИСТОЧНИКОВ

1. Каваленка, У.Ю. Дэманстрацыйна-тэставая ферма праграм для платформы Android з вэб-інтэрфейсам / У.Ю. Каваленка, Д.А. Касцюк // П'ята міжнародна наукова-практычна канферэнція FOSS Lviv 2015: збірник наукових праць. – Львів, 23–26 квітня 2015. – С. 42–45.
2. Коваленко, В.Ю. Кроссплатформенные виртуальные рабочие места / В.Ю. Коваленко, Д.А. Костюк // Информационные технологии и системы 2015 (ИТС 2015): материалы международной научной конференции. – Минск: БГУИР, 29 октября 2015. – С. 104–105.
3. Касцюк, Д.А. Ужыванне віртуальных машын у складзе ілюстраваных аглядаў гісторыі праграмнага забеспячэння / Д.А. Касцюк, П.А. Луцюк, С.С. Уласенка, В.А. Жалудок // Трэтя міжнародна наукова-практычна канферэнція FOSS Lviv 2014: збірник наукових праць. – Львів, 24–27 квітня 2014. – С. 51–54.