Санкт-Петербургский государственный университет

Кафедра Системного Программирования

Комаров Константин Михайлович

Распределенная система автоматизированного тестирования алгоритмов обнаружения вредоносных программ

Бакалаврская работа

Научный руководитель:
ст. преп. М. В. Баклановский

Рецензент:
д. т. н., профессор И. А. Зикратов

Санкт-Петербург
2016

SAINT-PETERSBURG STATE UNIVERSITY

Chair of Software Engeneering

Konstantin Komarov

Distributed system for automated testing of malware detection algorithms

Graduation Thesis

Scientific supervisor:
Sr. Lecturer Maxim Baklanovsky

Reviewer:
Professor Igor Zikratov

Saint-Petersburg
2016

Оглавление

Введение

Борьба с вредоносными программами – одна из самых острых проблем компьютерной безопасности. Каждый день появляется около 332 тыс. уникальных вредоносных объектов [5]. Для борьбы с ними чаще используют так называемый сигнатурный подход. Он заключается в анализе существующих угроз и поиске шаблонов атак: ищутся известные файлы компьютерных вирусов, опасные сетевые пакеты, блокируются известные вредоносные воздействия на систему. Однако с появлением новейших видов вредоносного ПО сигнатурные методы противодействия все больше показывают свою неэффективность и технологическое отставание. Они позволяют противостоять лишь известным, старым угрозам. Существуют и другие методы борьбы с вредоносными программами. Например, методы, основанные на поиске аномалий в работе программ.

Для проверки эффективности алгоритмов обнаружения вредоносных программ используются системы массового запуска внутри виртуальных машин, изолированных от внешнего мира. Такие тесты проводятся, например, компаниями, занимающиеся аналитикой и тестированием антивирусных защит [10].

Целью данной бакалаврской работы является создание подобной системы, позволяющей эффективно тестировать алгоритмы обнаружения вредоносных программ. Планируется использовать ее, главным образом, для проекта КОДА [1] – это система противодействия вредоносным программам, разрабатываемая с 2009 г. на кафедре системного программирования СПбГУ. Проведенные эксперименты показали, что алгоритмы КОДА позволяют выявлять вредоносную активность. На данный момент, основной задачей проекта является разработка способа оценки степени угрозы потока процесса по характеристикам, полученным с помощью этих алгоритмов. Для этого решено использовать алгоритмы машинного обучения. Таким образом, необходимо провести большое количество тестов, чтобы получить выборку исходных данных для обучения.

Ручное тестирование методов борьбы с вредоносными воздействиями очень затратная по времени задача. Был создан прототип системы тестирования (разд. 1.1.2), однако, даже с использованием нескольких потоков, этот процесс занимает существенное время. Необходимо масштабировать систему тестирования для работы на нескольких компьютерах. Для решения этой задачи применяются распределенные вычисления [17].

Таким образом, можно выделить следующие задачи:

- Провести масштабирование системы
- Улучшить существующую систему
- Протестировать систему под нагрузкой

1. Обзор

1.1. Существующие решения

1.1.1. Сторонние компании

Тестированием средств обеспечения компьютерной безопасности сейчас занимается множество компаний. Например, [10, 8]. Цель тестирования - предоставить пользователю объективную информацию о функциональных возможностях популярных на рынке решений, на основании которой пользователь сможет самостоятельно принимать решение об их использовании. Компании проводят большое количество различных проверок, например:

- Тестирование на угрозах реального времени (Real-World Threats). Он состоит в том, что тестировщик собирает несколько сотен веб-ссылок на зараженные сайты из различных источников. Как правило, на такие ссылки каждый из нас натыкается в поисковиках, получает по e-mail, ICQ или другие средства интернет коммуникации, включая социальные сети. Затем тестировщик последовательно переходит по ссылкам на опасные веб-страницы и фиксирует все изменения тестовой системы. На основании полученных данных делается вывод, успешно ли тестируемый продукт обнаружил угрозу.

- Тестирование файлового сканера антивируса. Тестировщик собирает коллекцию вредоносных программ и с помощью файлового сканера, встроенного в проверяемый антивирус, проверяет, сколько из них обнаруживаются.

Можно заказать и тестирование любого алгоритма. Однако компании предоставляют только результаты проверок, сами тестирующие утилиты и другие специальные средства не выкладываются. А значит, мы не сможем обойтись своими силами, если нам понадобится провести ряд тестов заново.

1.1.2. Прототип

В рамках проекта КОДА уже был реализован рабочий прототип системы тестирования [6], который можно успешно применять для решения поставленной задачи. Однако он обладает следующими недостатками:

- Он *недостаточно быстр*, поскольку может использовать ресурсы только одной машины, несмотря на то, что позволяет производить процесс тестирования в несколько потоков.
- Система *жестко привязана к одному алгоритму*. Если сколько-нибудь существенно изменить тестируемый алгоритм, потребуется разобраться и переписать некоторые участки кода, чтобы заново использовать ее.
- *Отсутствие проверки корректности отчетов*. Для того чтобы протестировать алгоритм обнаружения вредоносных программ, как ни банально, но нужны эти самые вредоносные программы (семплы). В качестве базы была использована онлайн база вредоносных семплов [18]. В базе далеко не всегда попадаются действительно рабочие семплы. В результате, отчет, полученный при тестировании на нерабочем семпле, не может считаться достоверным.

1.2. Системы для организации распределенных вычислений

Понятие распределенной системы

В настоящее время существует большое разнообразие определений распределенной системы, причем среди них ни одно не является строгим или общепринятым. Пожалуй, наиболее полное определение принадлежит Эндрю С. Таненбауму (Andrew S. Tanenbaum) [3]: «Распределенная система - набор независимых компьютеров, представляющийся их пользователям единой объединенной системой». Другое определение представлено в работе [4]: «Распределенная система - это такая система,

в которой взаимодействие и синхронизация программных компонентов, выполняемых на независимых сетевых компьютерах, осуществляется посредством передачи сообщений». Мы в дальнейшем будем рассматривать распределенные вычисления в широком смысле, как теоретическую основу для построения распределенных систем обработки данных.

Для того чтобы поддержать представление различных независимых компьютеров и сетей в виде единой системы, организация распределенных систем часто включает в себя дополнительный уровень программного обеспечения, называемый программным обеспечением промежуточного уровня (middleware). Он располагается между верхним уровнем, на котором находятся пользователи и приложения, и нижним уровнем, состоящим из операционных систем.

Эффективная распределенная система должна удовлетворять следующим основным требованиям [2]: прозрачность, открытость системы, безопасность, масштабирование, надежность. Несмотря на кажущуюся очевидность и простоту перечисленных требований, на практике их реализация представляет из себя непростую задачу.

Проблемы построения распределенных систем

На практике, при построении распределенных систем, возникают множество проблем [12, Challenges]. Рассмотрим лишь наиболее существенные из них:

- *Конфликты при одновременном доступе к ресурсам.* Очень многим элементам системы может понадобиться один и тот же ресурс, например, файл. Ресурс в распределенной системе физически расположен на одном узле и может быть доступен другим узлам только через сеть. Ключевыми проблемами являются распределение нагрузки, обеспечение надежности, доступности, защищенности. В качестве простых способов решения можно отметить: очереди к ресурсам, создание копий ресурсов, использование распределенной файловой системы.

- *Проблема тестирования.* Сбои, возникающие в распределенных

системах, частичны, то есть некоторые узлы могут находиться в состоянии ошибки, в то время как другие продолжают исправно функционировать. Подобное поведение усложняет реализацию распределенной системы. Кроме того, при тестировании тяжело воспроизвести ситуацию реальной нагрузки. Таким образом, часть проблем выясняется только при пробном пуске. Поэтому система должна иметь очень развитую систему диагностики, чтобы за минимальное время выявить проблему.

Как было показано выше, создание распределенной системы задача нетривиальная, ставит перед собой необходимость решения многих проблем. К счастью, большинство из них можно избежать, поскольку проблема довольно хорошо изучена и уже существуют готовые решения.

Конкретизация задачи

Итак, для достижения поставленной задачи, необходима соответствующая программная платформа. Система должна уметь создавать множество подзадач тестирования, распределять эти подзадачи по вычислительным узлам, принимать результаты вычислений и объединять их в единое целое для дальнейшей обработки.

Наша задача представляет собой задачу с идеальным параллелизмом – каждое подзадание никак не связано с другими, поэтому легко поддается распределению на несколько узлов. Остается выбрать наиболее подходящую для этого систему.

1.2.1. На основе систем добровольных вычислений

С появлением и развитием интернета, идея использования компьютеров для добровольных вычислений стала получать все большую популярность. Существует несколько программных комплексов для упрощения организации вычислений.

Одной из таких систем является BOINC (разд. 1.3) – программное обеспечение промежуточного уровня для быстрой организации распределённых вычислений [9]. Она отличается простотой в установке, на-

стройке и администрировании, нетребовательна к памяти, а также обладает хорошими возможностями по масштабируемости. BOINC предназначен для выполнения большого числа подзаданий, которые являются частью одной большой затратной по времени задачи. Каждое подзадание при этом проходит установленный жизненный цикл, начиная от генерации и заканчивая обработкой результата.

1.2.2. На основе облачных вычислений

Многие производители предлагают услуги вычисления по подписке основанные на модели инфраструктура как сервис (IaaS) или платформа как сервис (PaaS) [19]. За плату пользователи могут приобрести виртуальную машину в облаке для разворачивания на ней инфраструктуры собственного приложения. Примером таких облачных платформ являются Amazon EC2, Microsoft Azure, Google App Engine.

Применимо к нашей задаче, мы можем использовать облачную платформу для запуска вредоносных семплов и генерации отчетов. Такой способ удобен, не требует написания программ для автоматического разворачивания виртуальных машин. Однако есть и недостатки, которые не позволяют нам выбрать этот метод в качестве основы. Прежде всего, это необходимость оплаты услуг, даже если есть возможность использовать небольшое количество ресурсов бесплатно, нет гарантии что так будет и в будущем.

1.2.3. На основе систем для работы с большими данными

Термин большие данные (Big Data [15]) появился относительно недавно и описывает набор методик необходимых для управления и обработки больших объемов данных, с которыми нелегко справиться традиционными средствами. Множество факторов привели к созданию решений для работы с большими данными, такие как рост объемов данных, быстрота и легкость передачи данных, повышение ценности данных, социальные медиа. Существует много систем для работы с большими данными, например, Apache Hadoop, Hortonworks Data Platform, HBase,

Cassandra.

Большая популярность и поддержка со стороны крупных компаний, привело к появлению сложных систем, состоящих из множества компонентов. Установка и настройка таких систем задача весьма нетривиальная, они сложны в освоении и сопровождении. В нашей работе, мы бы хотели сосредоточиться больше на вопросах виртуализации и мониторинге работы семплов, нежели на настройке и организации распределенной системы. Кроме того, наша задача не настолько сложна с точки зрения распределенных вычислений, чтобы решать ее такими мощными инструментами, поэтому вариант с использованием систем работы с большими данными использован не будет.

1.2.4. На основе непрерывной интеграции

Для решения поставленной задачи, также можно применить систему непрерывной интеграции. Непрерывная интеграция – это практика разработки программного обеспечения, когда участники команды часто объединяют результаты своей работы друг с другом: от одного до нескольких раз в день. Каждая интеграция проверяется автоматической сборкой и тестированием для определения ошибок настолько быстро, насколько это возможно. Систем для организации непрерывной интеграции гораздо больше чем систем для построения распределенных вычислений. Например, CruiseControl, TeamCity, Jenkins.

Существующие средства непрерывной интеграции предназначены в первую очередь для оптимизации процесса разработки программного обеспечения, нацелены на повышение качества программного обеспечения и представляют собой довольно громоздкие программные продукты. Особенность этих систем состоит в том что тестирующие узлы должны быть предконфигурированы, иначе нужно реализовывать систему автоматического разворачивания. Кроме того, такие системы не предполагают длительного процесса тестирования, длящегося несколько недель или месяцев.

Итог

Выбор конкретной системы для реализации распределенных вычислений не очевиден, поскольку нет достаточно веских аргументов в пользу той или иной системы. Было принято решение организовывать распределенные вычисления на платформе BOINC. Приложение для BOINC – это практически обычное приложение в виде исполняемого файла, которое может работать и без клиента в стационарном режиме. Таким образом, если потребуется перейти от BOINC к, например, серверу непрерывной интеграции, это небольшая проблема, которая потребует незначительных изменений в коде.

1.3. Berkeley Open Infrastructure for Network Computing

Для решения описанных выше проблем или, по крайней мере, минимизации их влияния, был использован программный комплекс для быстрой организации распределённых вычислений Berkeley Open Infrastructure for Network Computing (BOINC) [9]. Он представляет собой готовую обвязку для проектов, связанных с сетевыми вычислениями, которая значительно облегчает их запуск, хотя и не избавляет полностью от ручной работы, поскольку ряд серверных модулей необходимо готовить под конкретную задачу. Для того чтобы правильно спроектировать структуру приложения, нужно детально разобраться в структуре этой системы.

1.3.1. Архитектура

В основу архитектуры заложен принцип конечного автомата – сервер представляет собой набор сервисов, каждый из которых отвечает за отдельную задачу, например, за передачу файлов, создание заданий и т.п. Каждый сервис в бесконечном цикле проверяет состояние задания, производит необходимые действия, изменяя тем самым состояние задачи. Основные компоненты системы это управляющий сервер и мно-

жество клиентов. Клиент запрашивает задание, сервер формирует его и отправляет клиенту, клиент обрабатывает задание, и отправляет результат серверу для анализа и агрегации.

1.3.2. Устройство сервера

В качестве ОС для сервера BOINC технически может использоваться любая Unix система. В документации рекомендуются использовать последний релиз Linux [11, ServerIntro]. Сервер состоит из следующих компонент:

- Полноценного Web-сервера, который обрабатывает сообщения клиентов и предоставляет управление администратору проекта;
- Сервера баз данных, который является сердцем проекта. В ней хранится вся информация, относящаяся к проекту, включая сведения о приложениях и их версиях, о пользователях, о заданиях и текущем их состоянии;
- Минимум пяти служб, которые обслуживают систему и организуют распределение задач на клиентов, проверяя состояние базы данных. Каждая из этих служб может быть распределена на несколько машин с помощью Network Filesystem (NFS) для повышения производительности [3, Сетевая файловая система компании Sun].

Общая схема работы служб представлена на рис. 1.

Планировщик

Планировщик управляет запросами от клиентов. Это CGI программа. Она запускается, как только к серверу присоединяется клиент и запрашивает задачу. Запрос содержит описание ОС и данные о производительности клиента. Работая с планировщиком, клиент также передает сведения о завершенных задачах, которые еще не были зарегистрированы со времени последней сессии планирования. В конечном счете

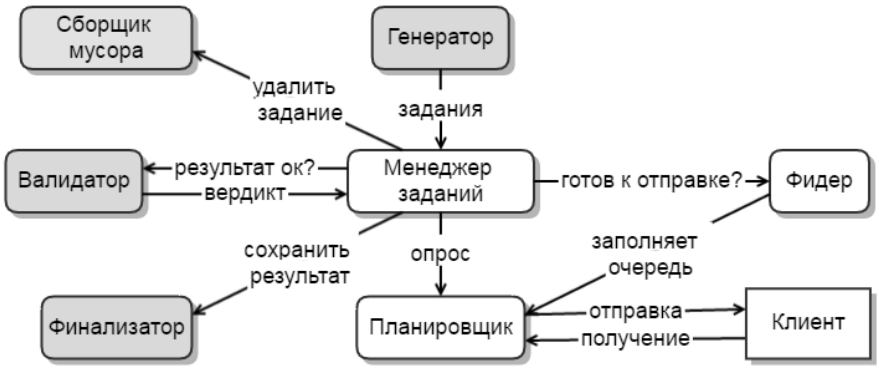

Рис. 1: Общая схема служб (серым выделены компоненты, оставленные для самостоятельной реализации)

клиент получает список заданий для обработки и список адресов, где можно получить файлы, необходимые для запуска заданий на клиентском компьютере.

Менеджер заданий

Проверяет в базе данных, в каком состоянии находится задача и обновляет соответствующие поля, когда задача готова перейти в другое состояние. Следит, чтобы каждая задача проходила правильный цикл обработки. Менеджер заданий одна из компонент наиболее требовательных к ресурсам сервера, поэтому, в первую очередь, рекомендуется распределять на несколько серверов именно эту службу.

Фидер

Служба фидер является вспомогательной – она загружает задачи, для которых еще не получен результат из базы данных в область разделяемой памяти. Это необходимо для того чтобы ограничить число запросов к базе данных и повысить производительность системы.

Валидатор

Этот компонент необходим для проверки полученных результатов. В целях обеспечения достоверности, каждая задача обрабатывается на

нескольких клиентах. Затем, полученные кворумом клиентов, результаты сверяются друг с другом, чтобы определить так называемое «каноническое» решение. Алгоритм для проверки результатов полностью зависит от конкретной задачи.

Финализатор

Проверяет наличие в базе данных завершенных заданий и определяет дальнейшую постобработку канонических результатов. Например, их можно архивировать, запускать последующий анализ и т.д. Задача будет помечена как завершенная только после прохода через эту службу.

Сборщик мусора

Эта служба нацелена на поддержание производительности файловой системы. Она проверяет наличие в базе данных финализированных задач, а при обнаружении таковых очищает все временные файлы этих задач. Конечно, можно удалять файлы еще на этапе финализации, однако такой подход приводит к излишней фрагментации и нагрузке на файловую систему, поскольку финализация происходит гораздо чаще сборки мусора.

1.3.3. Структура заданий

Когда BOINC проект создан и запущен, приложение реализовано и добавлено в работу, наконец, необходимо добавить задания которые будут выполнять клиенты. Создавать задания можно несколькими способами:

- Скриптом из коммандной строки [11, JobSubmission];
- Программой, которая будет использовать вызовы к BOINC API [11, WorkGeneration];
- С помощью удаленного вызова процедур BOINC [13];

- Через веб интерфейс [14].

Какой бы ни был выбран способ, для каждой задачи потребуется заранее сформировать шаблоны входных/выходных файлов и указать входные параметры задачи. Каждый набор входных параметров должен быть уникальным. Если потребуется провести вычисления на наборе одинаковых входных параметров, в одном и том же приложении, необходимо чтобы они имели разные имена. Кроме того, заданиям с разными именами, можно назначить разные варианты валидации и постобработки результатов. В любом случае, потребуется доступ к результатам заданий, поэтому, чтобы не нарушить процесс обработки, механизм доступа к ним производится также через BOINC API.

Каждый шаблон содержит список имен файлов, который необходим для запуска приложения или для отправки результатов, а также и способ их открытия. Имена играют важную роль во всех компьютерных системах. Они необходимы для совместного использования ресурсов, определения уникальных сущностей, ссылок на местоположения и т.д. Важная особенность именования состоит в том, что имя может быть разрешено, предоставляя доступ к сущности, на которую оно указывает. Разрешение имени, таким образом, представляет собой процесс доступа к именованной сущности. Чтобы облегчить управление файлами, приложение использует виртуальные имена файлов, вместо физических. За раскрытие имен файлов отвечает BOINC API. На рис. 2 приведен пример того, как выполняются приложения. Каждое приложение запускается в отдельной рабочей директории (слоту). Каждый файл может быть открыт в двух режимах: (1) как символьная ссылка, (2) напрямую, причем файл предварительно копируется в слот. вне зависимости от случая, BOINC предоставляет API для открытия `boinc_fopen()`. В нашем случае виртуальный файл действует как символьная ссылка. Кроме того, файлы шаблонов задают сколько необходимо операций с плавающей точкой для завершения задания, временные ограничения, ограничения на максимальный размер результатов, является ли файл обязательным для успешного завершения задания и необходимость немедленной отправки файла.

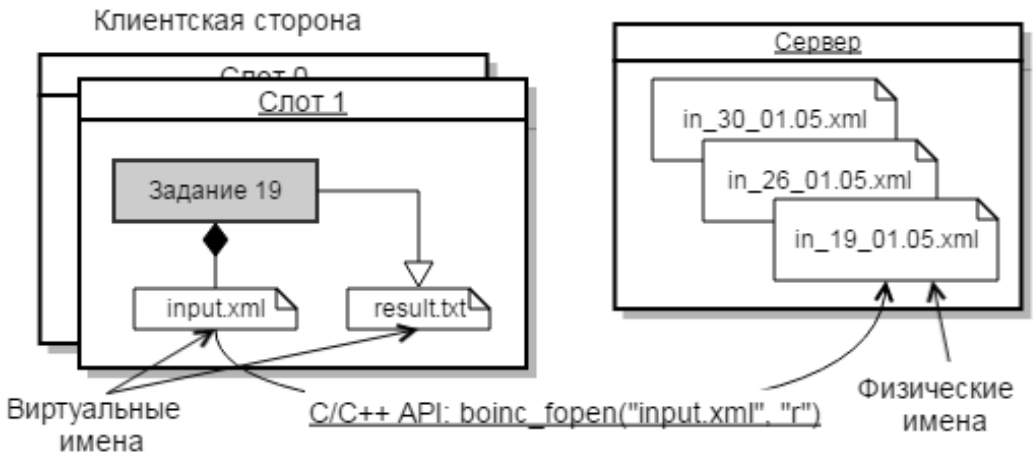

Рис. 2: Виртуальные имена и механизм слотов

1.3.4. Обработка заданий

Одно и то же задание рассчитывается на нескольких различных клиентах. Избыточные вычисления такого рода используются для обеспечения достоверности, а так же обнаружения фальсификации результатов клиентами. Каждый экземпляр задания, готовый для отправки клиенту, называется «результатом», несмотря на то что фактически результат этот экземпляр задания получит только в самом конце своего жизненного цикла. Полученные результаты сравниваются и проверяется не достигнут ли «кворум» по этому заданию. При этом алгоритм проверки результатов целиком зависит от решаемой задачи. Для каждого задания при создании необходимо указать следующие параметры:

- **delay_bound:** верхняя граница ожидания ответа от клиента в секундах. Обычно устанавливается в несколько раз больше среднего выполнения программы;

- **min_quorum:** минимальное количество успешных ответов, необходимое для успешного завершения задания. Валидатор не будет вызван пока не наберется это количество;

- **target_nresults:** количество заданий запускаемых в начале. Должно быть не меньше `min_quorum`, может быть больше, чтобы отра-

зить коэффициент потерь или набирать "квоту" более быстро;

- **max_error_results:** если количество клиентских ошибок превысит это значение, то всему заданию будет присвоено значение ошибки. Представляет собой защиту от неправильно составленного задания, вызывающего падение приложения;

- **max_total_results:** если количество клиентских ответов превысит это значение, то всему заданию будет присвоено значение ошибки. Представляет собой защиту от заданий, которые не возвращают ответ (напр. аварийное завершение самого клиента);

- **max_success_results:** если количество успешных клиентских ответов превысит это значение, то всему заданию будет присвоено значение ошибки. Представляет собой защиту от заданий, дающих неопределенный результат.

Рассмотрим несколько примеров обработки заданий. Пусть заданы следующие параметры: `delay_bound = 9`, `min_quorum = 2`, `target_nresults = 3`. Значение остальных параметров в примерах использовано не будет, но считается, что они не противоречит заданным. На рис. 3 показано создание одного задания (Время 0); в следующий момент времени в БД создается три записи для результатов заданий, которые отправляются клиентам для обработки. Процесс валидации запускается как только возвращается хотя бы два успешных результата (Время 7). После успешной валидации задание финализируется. Тем временем, возвращается третий результат (Время 9), а так как верхняя граница ожидания ответа не достигнута, тогда если этот результат успешно проходит процесс валидации, то степень доверия результатам, получаемым от этого клиента, увеличивается. Случай, когда какой-либо из результатов теряется, показан на рис. 4. Результат 2 теряется, но к концу ожидания ответа (Время 11) уже получены два успешных результата, поэтому задание проходит валидацию и успешно финализируется. Если же результат возвращается с ошибкой, но кворум еще не

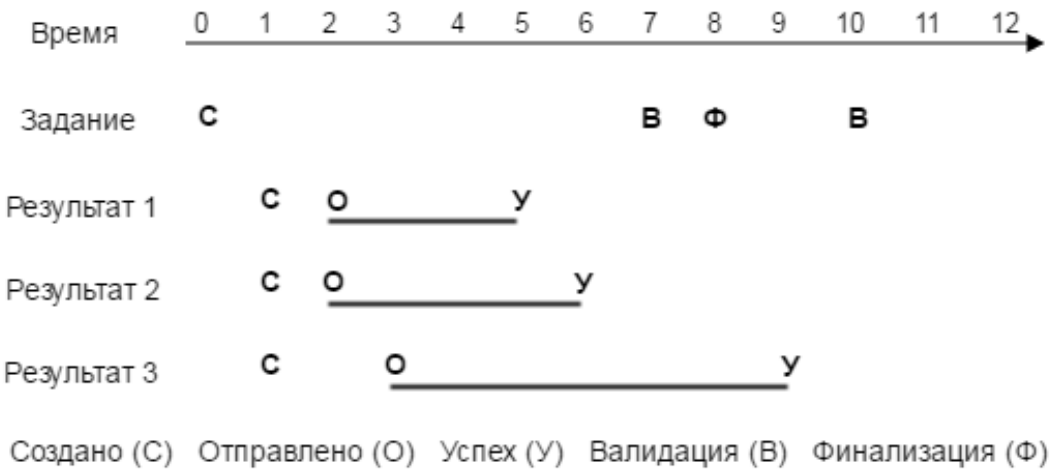

Рис. 3: Пример обработки задания

достигнут (рис. 5), получение результата передается другому клиенту (Результат 4).

В общем случае, во время получения результатов заданий могут произойти следующие ситуации [11, JobReplication]:

- Клиент успешно выполнил задание и вернул результат
- Клиент выполнил задание с ошибкой и вернул результат
- Клиенту не удалось скачать и отправить файлы
- Приложение на клиенте завершилось аварийно
- Клиент не вернул ничего
- Планировщик не может отправить задание, потому что ни у одного клиента недостаточно ресурсов для выполнения

1.3.5. Устройство клиента

Клиент состоит из:

- Менеджера работы, который производит запросы и запускает задачи;

Рис. 4: Пример обработки задания

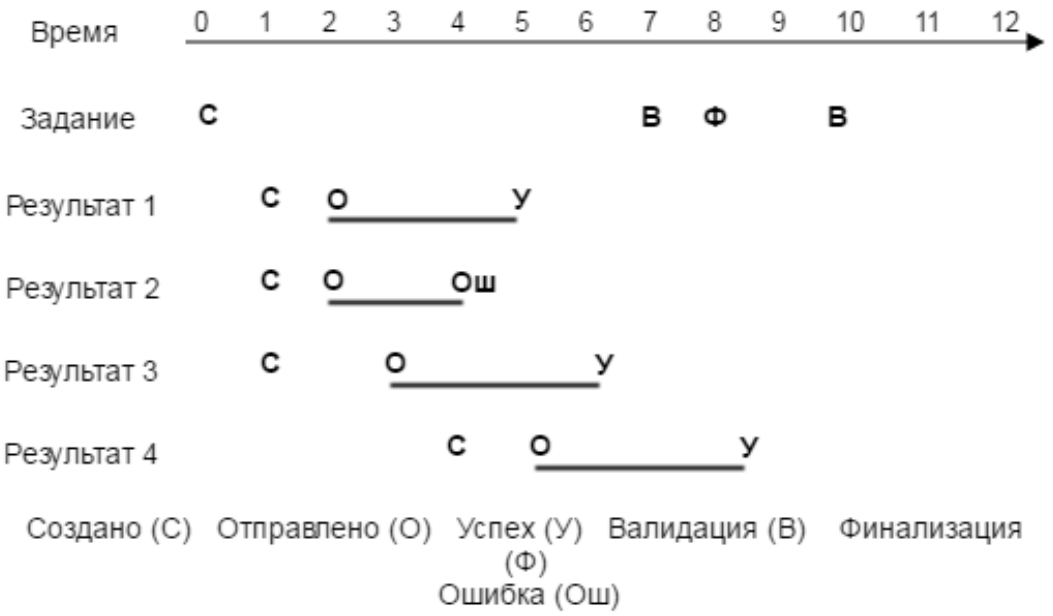

Рис. 5: Пример обработки задания

- Виртуальной машины, на которой запускается весь потенциально небезопасный процесс тестирования.

Для того чтобы подключиться к проекту необходимо установить BOINC клиент, указать URL сервера и идентификатор пользователя. Клиент скачивает, запускает задания и, как только готовы результаты, помещает их в очередь отправки обратно. Безопасность одна из самых важных проблем на клиентской стороне. Для того чтобы убедится что код получен из надежного источника и не является поддельным, проверяется цифровая подпись каждого исполняемого файла, они подписываются алгоритмом на основе RSA. Для каждого задания создается временная папка (слот), одновременно может запускаться несколько заданий.

Рис. 6: Общая архитектура системы (звездочкой отмечены компоненты для самостоятельной реализации)

2. Описание решения

2.1. Сервер

Для запуска системы необходимо самостоятельно реализовать минимум четыре службы сервера, клиентское приложение и приложение для виртуальной машины (рис. 6). Для реализации серверных компонент используется язык `C++`.

Процесс получения канонического результата

В теории работа виртуальной машины – детерминированный процесс, однако на практике добиться этого практически невозможно. Аппаратное ускорение вносит неопределённость течения времени с реального процессора внутрь виртуальности: одни и те же процессы внутри ВМ могут занимать немного разное время в разных прогонах, что проявляется и в различиях возвращаемых `rdtsc` значений. Поэтому, в общем случае, несколько отчетов одного и того же семпла не будут идентичными, а значит нет смысла прогонять один и тот же семпл несколько раз. На каждый тест отводится 1,5 мин, установим время ожидания ответа в 5 раз больше. Таким образом, для получения канонического результата каждого задания устанавливаются следующие параметры (листинг 1). Эти параметры используются генератором для

```
min_quorum = 1;
target_nresults = 1;
delay_bound = 90 * 5;
max_error_results = 4;
max_total_results = 8;
max_success_results = 4;
```

Листинг 1: Параметры заданий

создания заданий. Кроме того, в функционал генератора входит

Валидатор и финализатор тривиальны. Валидатор лишь проверяет, что в отчете присутствует маркер конца, а также при сравнение любых двух результатов отвечает, что они равны (хотя подобная ситуация и невозможна вследствие `target_nresults = 1`). Финализатор сохраняет все результаты в отдельной папке, которая периодически синхронизируется с облаком.

2.2. Клиентское приложение

Для запуска исполняемых файлов на виртуальной машине, раньше использовались возможности `VBoxManage` [16, VBoxManage controlvm]. С выходом новой пятой версии `VirtualBox` синтаксис именно этой команды сильно изменился, что повлекло за собой изменения в коде приложения. Кроме того, необходимо специальным образом настроить права на гостевой ОС; такой метод запуска не позволяет обращаться к файлам в сетевом окружении, таким образом нельзя запустить файл прямо из общей папки. На смену ему приходит следующее простое решение: на виртуальной машине уже должно быть запущено приложение с необходимыми правами, которое самостоятельно найдет и запустит необходимые программы.

Приложение также реализовано на языке `C++`.

В архитектуре BOINC не предусмотрено решения для раздачи очень больших файлов, таких как, например, файл-образа виртуальной машины, размер которого достигает 25Гб. Поэтому было решено переиспользовать протокол BitTorrent для передачи больших файлов.

Итак, при запуске приложение проверяет наличие необходимых файлов для развертывания виртуальной машины, файлов настроек и семплов. Затем создается и настраивается виртуальная машина (ВМ), если этого еще не было сделано раньше. При отсутствии снимка ВМ в выключенном состоянии, этот снимок создается. Это необходимо для того, чтобы исходный образ жесткого диска виртуальной машины оставался нетронутым, а все изменения проводились с разностным жестким диском. Такой подход дает возможность восстанавливать исходное состояние ВМ, в случае аварийного завершения работы без необходимости заново загружать файл-образ ВМ. Затем восстанавливается рабочее состояние. Проверяется наличие снимков ВМ в запущенном состоянии, если их нет, ВМ запускается и создается снимок из рабочего состояния. Это необходимо для того, чтобы не ждать загрузки ОС при каждом запуске. Можно было бы передать все снимки ВМ по сети, но (1) вес каждого снимка велик (1.5Гб) и передача занимает большое количество времени, (2) создавать снимки непосредственно на клиенте повышает переносимость приложения. Затем запускается таймер ожидания результата и производится мониторинг триггер-файлов в общей папке, создание которых указывает на завершение задачи с результатом, возможно, не всегда успешным.

2.3. Приложение на виртуальной машине

Виртуальная машина должна быть изолирована от сети, во избежания сетевой атаки доступных из виртуальной машины хостов.

Можно использовать любой язык для написания приложения на виртуальной машине, потому как оно взаимодействует с клиентским приложением косвенно через общую папку. За основу был взят собственный прототип (разд. 1.1.2), который написан на языке `Perl`, поэтому разработка продолжена на этом же языке.

На виртуальной машине в бесконечном цикле работает специальный скрипт, который проверяет содержимое общей папки и ждет появления файлов заданий для запуска и передачи им управления. Для организа-

ции сообщения применяются общие папки, их две:

- **shared_r:** доступна только для чтения и используется для того, чтобы передавать задания и сопутствующие файлы на ВМ.

- **shared_w:** к этой папке у виртуальной машины уже полный доступ, через нее передаются результаты заданий. В целях предотвращения запуска файлов вредоносных программ, которые могут быть созданы в этом каталоге, любые другие файлы, кроме файлов-результатов и триггер-файлов немедленно удаляются.

Как уже было описано выше (разд. 1.1.2), используется не самая надежная база вредоносных объектов. Для повышения точности результатов, необходимо более тщательно подходить к выбору семплов для тестов. Для этого было решено использовать три механизма:

- *Статический анализ PE-заголовка.* Семплы с некорректным заголовком не могут быть запущены, а значит не могут использоваться для тестов. В отличие от следующих двух пунктов, этот процесс производится на сервере в процессе генерации еще до попадания заданий клиентам. Конечно, мы можем вручную один раз просеять всю базу семлов и затем работать уже с ней, но тогда существует вероятность, что при обновлении базы в нее будут ошибочно добавлены семплы с некорректным заголовком.

- *Проверка кода возврата* запущенного процесса. Такой метод, в действительности, не гарантирует отсеивания только нерабочих семплов, однако ловит большое количество ошибок загрузчика.

- *Мониторинг появление диалоговых окон ошибок.* Следит не только за возможными ошибками в коде вредоносных семплов, а также за неисправностями, аварийным завершением компонентов окружения, в том числе и тестирующих утилит.

3. Эксперименты

Система тестировала алгоритм обнаружения вредоносных программ системы КОДА [1]. Задание для клиента состояло в следующем: запустить систему КОДА, запустить очередной семпл вредоносной программы, записать в файл лог работы системы КОДА (отчет) и отослать на сервер.

Прежде всего, было произведено тестирование на той же самой выборке (1,8 тыс.), что и тестирование прототипа [6]. Результатом работы прототипа было то, что из 1,8 тыс. вредоносных программ запустилось только 38%, таким образом, было получено 690 отчетов. В следствие улучшения алгоритмов фильтрации базы вредоносных программ от прошлого набора в 690 валидных отчетов осталось только 430. Анализ набора в 690 отчетов показывал, что уровень обнаружения вредоносных объектов не менее 82,31%. Тот же анализ, но уже отфильтрованного набора в 430 отчетов показывает, что уровень обнаружения не менее 94%.

В качестве базы для тестирования была использована онлайн база вредоносных семплов [18]. Для запуска отбирались семплы с пометкой `Win32`. Около 32,85% не прошли проверку корректности PE заголовка и были отсеяны еще до запуска процесса тестирования на клиентах, таким образом отобрано ровно 60 тыс. вредоносных семплов. Общая статистика результатов их запуска представлена в табл. 1. Дополнительно стоит сказать, что технически система КОДА – это пользовательское приложение, которое никак себя не защищает, поэтому не удивительно, что случаются аварийные завершения. В будущем, если у системы КОДА появится защита, то эти 21,17% потенциально могут стать валидными отчетами. «Заблокирован брандмауэром» означает, что во время тестирования брандмауэр Windows помешал нормальной работе семпла и заблокировал некоторые его функции. В последующих тестах брандмауэр будет отключен. Вердикт «Подделка под антивирус» выносился, когда вредоносная программа пыталась выдать себя за антивирусное ПО, с расчетом на то, что пользователь сам установит ее в системе.

Вердикт	Количество семплов	%
Найдены окна ошибок	22242	37,07
Валидный отчет	19875	33,13
Аварийное завершение КОДА	12705	21,17
Плохой код возврата семпла	4664	7,77
Заблокирован брандмауэром	381	0,64
Подделка под антивирус	113	0,19
Обнаружена ВМ	20	0,03

Таблица 1: Статистика тестирования на наборе из 60 тыс. семплов

В очень малом количестве случаев, вредоносная программа обнаруживала работу под виртуальной машиной и честно выводила сообщение об этом. Конечно, это не означает, что другие вредоносные программы присутствие виртуальной машины не обнаруживали.

Было замечено, что на каждые 5 тыс. заданий, примерно 5-6 задний генерируют пустой результат. Такой результат считается ошибкой, а задание отправляется на обработку заново, после чего всегда получает уже непустой результат.

4. Заключение

В рамках данной работы были получены следующие результаты.

- Произведено масштабирование системы. Скорость тестирования возросла пропорционально числу узлов в распределенной сети.
- Система доработана.
- Проведено тестирование системы.

Все цели были достигнуты, а задачи выполнены.

В качестве дальнейшего развития, можно различными способами улучшать реализованную тестирующую систему, масштабировать ее на более серьезные производительные мощности, тестировать с помощью нее новые алгоритмы обнаружения и получать более состоятельные результаты.

Данная работа была представлена на Всероссийской научной конференции по проблемам информатики СПИСОК 2016 [7].

Список литературы

[1] Баклановский Максим Викторович, Ханов Артур Рафаэльевич. Поведенческая идентификация программ // Моделирование и анализ информационных систем. — 2014. — Vol. 21, no. 6. — P. 120–130.

[2] Алпатов А. Н. Цветков В. Я. Проблемы распределенных систем // Перспективы науки и образования. — 2014. — no. 6 (12).

[3] Таненбаум Э. ван Стеен М. Распределенные системы. Принципы и парадигмы. — СПб: Питер, 2003. — 880 с.

[4] М.С. Косяков. Введение в распределенные вычисления. — Санкт-Петербург: НИУ ИТМО, 2014. — 155 с.

[5] Статистика Лаборатории Касперского. — 2015. — URL: `https://securelist.ru/analysis/ksb/27543/` (дата обращения: 2016-05-01).

[6] К.М. Комаров. Система автоматизированного массового тестирования проекта CODA // - Математико-механический факультет СПБГУ. — 2015. — URL: `http://se.math.spbu.ru/SE/YearlyProjects/2015/YearlyProjects/2015/344/344-Komarov-report.pdf` (дата обращения: 2016-05-01).

[7] Комаров К.М. Распределенная система автоматизированного тестирования алгоритмов обнаружения вредоносных программ // СПИСОК-2016: Материалы всероссийской научной конференции по проблемам информатики. — 2016.

[8] Тесты антивирусов 2016 - Anti-Malware.ru. — URL: `http://www.anti-malware.ru/antivirus_tests` (дата обращения: 2016-05-01).

[9] Официальный сайт BOINC // `https://boinc.berkeley.edu/`. — URL: `https://boinc.berkeley.edu/` (дата обращения: 2016-05-01).

[10] AV-Comparatives Real-World Protection Test. — URL: http://www.av-comparatives.org/dynamic-tests/ (дата обращения: 2016-05-01).

[11] Computing with BOINC. — URL: http://boinc.berkeley.edu/trac/wiki/ProjectMain (дата обращения: 2016-05-01).

[12] Coulouris George F, Dollimore Jean, Kindberg Tim. Distributed systems: concepts and design. — pearson education, 2005.

[13] Giorgino Toni, Harvey Matt J, De Fabritiis Gianni. Distributed computing as a virtual supercomputer: Tools to run and manage large-scale BOINC simulations // Computer Physics Communications. — 2010. — Vol. 181, no. 8. — P. 1402–1409.

[14] Legion: An extensible lightweight web framework for easy BOINC task submission, monitoring and result retrieval using web services / Genghis Ríos, Oscar Díaz, Pablo Fonseca et al. — 2014.

[15] Marz Nathan, Warren James. Big Data: Principles and best practices of scalable realtime data systems. — Manning Publications Co., 2015.

[16] Oracle VM VirtualBox – User Manual. — URL: http://www.virtualbox.org/manual/ (дата обращения: 2016-05-01).

[17] Tanenbaum A.S. van Steen M. Distributed Systems: Principles and Paradigms (2nd Edition). — Pearson Prentice Hall, 2007. — 704 p. — ISBN: 978-0132392273.

[18] VX Heaven - Massive, continuously updated collection of virus samples, virus generators etc. — 2016. — URL: http://vxheaven.org/ (дата обращения: 2016-05-01).

[19] Zhang Qi, Cheng Lu, Boutaba Raouf. Cloud computing: state-of-the-art and research challenges // Journal of internet services and applications. — 2010. — Vol. 1, no. 1. — P. 7–18.