Распределенная система автоматизированного тестирования алгоритмов обнаружения вредоносных программ

К.М. Комаров
студент кафедры системного программирования СПбГУ,
E-mail: erwo@mail.ru

Аннотация

В статье приведено краткое описание архитектуры распределенной системы для тестирования алгоритмов обнаружения вредоносных программ в защищенной среде. Система реализована с помощью библиотеки BOINC.

Введение

Для проверки эффективности алгоритмов обнаружения вредоносных программ используются системы массового запуска внутри виртуальных машин, которые изолированы от внешнего мира. Такие тесты проводятся, например, компаниями, которые занимаются аналитикой и тестированием антивирусных защит [1]. Ручное тестирование методов борьбы с вредоносными воздействиями очень затратная по времени задача. Автоматизированное тестирование на одном ПК даже с использованием несколько потоков занимает существенное время [2]. Необходимо масштабировать систему тестирования для работы на нескольких компьютерах. Для решения этой задачи применяются распределенные вычисления [3].

Типичные проблемы, возникающие при построении больших распределенных систем

- Конфликты при одновременном доступе к ресурсам. Очень многим элементам системы может понадобиться один и тот же ресурс, например, файл. Способы решения: очереди к ресурсам, создание копий ресурсов [4]
- Проблема тестирования. При тестировании тяжело воспроизвести ситуацию реальной нагрузки. Таким образом, часть проблем выясняется только при пробном пуске. Поэтому система должна иметь очень развитую диагностику, чтобы за минимальное время выявить проблему

Архитектура

Для решения описанных проблем или, по крайней мере, минимизации их влияния, был использован программный комплекс для быстрой организации распределённых вычислений Berkeley Open Infrastructure for Network Computing (BOINC) [5]. BOINC 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.

В основу архитектуры заложена идея конечного автомата – сервер состоит из набора отдельных подсистем, каждая из которых отвечает за свою вполне определенную задачу, например, выполнение вычислений, передачу файлов и т.д. Каждая из подсистем проверяет состояние подзадачи, производит какие-то действия и изменяет состояние подзадачи – так они работают в бесконечном цикле.

Система состоит из сервера, который формирует для клиента задания и множества клиентов, выполняющих задания сервера. Клиент запрашивает задание, сервер формирует задание и отправляет клиенту, клиент обрабатывает задание, и отправляет результат серверу.

Система тестировала алгоритм обнаружения вредоносных программ системы КОДА [6]. Задание для клиента состояло в следующем: запустить систему КОДА, запустить очередной семпл вредоносной программы, записать в файл лог работы системы КОДА и отослать на сервер.

Устройство сервера

Сервер состоит из:

- Как минимум одного Web-сервера, который обрабатывает входящие и исходящие сообщения
- Сервера баз данных, который отслеживает состояние подзадач и соответствующие им результаты, а также хранит информацию о клиентах
- Минимум пяти различных служб (демонов), которые периодически проверяют состояние базы данных и выполняют необходимые работы по обслуживанию системы и распределению подзадач (Рис 1.). Каждая из этих служб может быть распределена на несколько машин с помощью Network

Filesystem (NFS) для повышения производительности [7]

Сердце проекта - это сервер баз данных. В ней хранится вся информация, относящаяся к проекту, включая сведения о пользователях, о приложениях и их версиях, о заданиях и текущем их состоянии.

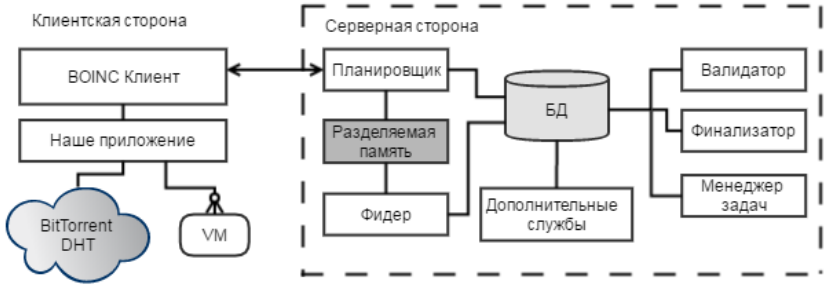

Рисунок 1. Архитектура

Планировщик

Планировщик управляет запросами от клиентов. Это CGI программа, которая запускается, когда к серверу присоединяется клиент и запрашивает задачу. Каждый запрос включает описание клиента и сведения о производительности хоста. Во время сессии работы с планировщиком клиент также сообщает о завершенных работах, которые были уже загружены на сервер со времени последней сессии планирования. В итоге клиент остается со списком заданий на обработку и списком адресов, с которых можно получить необходимые файлы, т.е. входящие файлы и файлы приложения, если их нет на клиентском компьютере.

Менеджер заданий

Эта служба проверяет в базе данных, в каком состоянии находится задача и обновляет соответствующие поля, когда задача готова перейти в новое состояние. Следит, чтобы каждая задача проходила правильный цикл обработки. Менеджер заданий одна из наиболее ресурсоемких с точки зрения нагрузки на сервер, 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 сегмент разделяемой памяти. Эта предварительная работа выполняется для повышения

производительности системы BOINC в целом с помощью ограничения числа запросов к базе данных.

Валидатор

Назначение службы – организация проверки полученных результатов. В целях обеспечения достоверности каждая из подзадач рассчитывается на нескольких различных клиентах. Поэтому полученные в итоге результаты необходимо проверить, сверив между собой и определив «каноническое» решение – результат, полученный кворумом клиентов. При этом алгоритм проверки результатов целиком зависит от решаемой задачи.

Финализатор

Проверяет наличие завершенных заданий и определяет дальнейшую постобработку канонических результатов. Например, их можно архивировать, запускать последующий анализ и т.д. Задача помечается как завершенная только после прохождения через эту службу.

Устройство клиента

Клиент состоит из:

- Менеджера работы, который делает запросы и запускает задачи
- Виртуальной машины, на которой запускается весь потенциально небезопасный процесс тестирования
- Торрент-клиента, через который происходит обмен компонентами большого размера, например файла-образа виртуальной машины, размер которого достигает 25Гб.

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.

При запуске приложение проверяет наличие необходимых файлов для развертывания виртуальной машины, файлов настроек и семплов. Затем создается и настраивается виртуальная машина, если этого еще не было сделано раньше. Проверяется и 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 снимок ВМ по сети, но (1) вес снимка велик (1.5Гб) и это занимает большое количество времени, (2) создавать снимок 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, не всегда успешным.

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 организации сообщения применяются общие папки, причем их две. Первая доступна только для чтения и используется для того, чтобы передавать задания. Ко второй папке у виртуальной машины уже полный доступ, через нее передаются результаты заданий. Причем любые другие файлы, кроме файлов-результатов и триггер-файлов немедленно удаляются.

Семплы для запуска берутся из общедоступной базы вирусов, в которой иногда попадается мусор. Поэтому для повышения точности результатов запуска таких файлов следует избегать. Для этого применяется два механизма. Во-первых, для каждого семпла статически анализируется и проверяется на корректность PE-заголовок. Во-вторых, во время тестирования на виртуальной машине, мониторится появление диалоговых окон ошибок и проверяется код возврата запущенного процесса.

Заключение

В документе была рассмотрена распределенная архитектура для тестирования алгоритмов обнаружения вредоносных программ, а также рассмотрены основные проблемы при построении подобных систем.

Литература

[1] AV-Comparatives Real-World Protection Test. — URL: http://www.

av-comparatives.org/dynamic-tests/ (дата обращения: 25.04.2016).

[2] 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: 12.11.2015).

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

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

[5] BOINC.—https://boinc.berkeley.edu/.—[Online; accessed 12-November-2015].

[6] М. В. Баклановский, А. Р. Ханов, “Поведенческая идентификация программ”, Модел. и анализ информ. систем, 21:6 (2014), 120–130

[7] C.B.Ries. BOINC - Hochleistungsrechnen mit Berkeley Open Infrastructure for Network Computing. —Berlin Heidelberg : Springer, 2012. — ISBN: 978-3-642-23382-1.