Санкт-Петербургский Государственный Университет
Математико-механический факультет

Кафедра Системного Программирования

Соболев Артём Александрович

Анализ взаимодействия приложений с файловой системой

Курсовая работа

Научный руководитель:
Козловский В.

Санкт-Петербург
2013

Оглавление

Введение **3**

1. Постановка задачи **4**

2. Реализация **5**

2.1. Профилируемое приложение 5

2.2. Имитация нагрузки 5

2.3. Воспроизведение профиля 7

3. Результаты **8**

Заключение **11**

Введение

В последние годы объемы хранимой и обрабатываемой информации многократно возрасли и продолжают увеличиваться. В этих условиях становится крайне важной задача предоставления надёжных и быстрых средств для работы с ней. Эту роль на себя берут системы хранения данных.

Разработка и оптимизация систем хранения данных осложняется тем, зачастую нет возможности протестировать решение в реальных условиях — на нагрузке, создаваемой покупателями. Клиенты не выдают свои данные, поскольку те обычно представляют собой коммерческую тайну или содержат личные данные их клиентов и, как следствие, не подлежат огласке. Так же в общем случае невозможно инструментировать клиентские приложения так, чтобы была возможность регистрировать все операции. Основанием этого может быть как недоверие, так и возможные трудности в поддержке.

Вместе с тем, было бы весьма полезно иметь возможность протестировать систему на таких данных. Ключевым замечанием тут является тот факт, что сами данные для нас не представляют интереса, нам важен лишь характер взаимодействия. Таким образом, достаточно регистрировать сами события, не заботясь о данных, с которыми идёт работа.

1. Постановка задачи

В рамках текущей задачи предлагается использовать файловую систему в качестве системы хранения данных, поскольку общие принципы одни и те же, но файловия система проще.

1. Развернуть приложение, работающее с файловой системой, реализовав возможность логирования операций ввода-вывода.
2. Нагрузить приложение так, чтобы профиль взаимодействия с файловой системой был аналогичен реальному.
3. Собрать данные о взаимодействии приложения с файловой системой, механизмы из первого пункта.
4. Проанализировать логи: исследовать характер запросов чтения-записи, выделить какие-то характерные черты.
5. Реализовать приложение, воспроизводящее эту нагрузку.

2. Реализация

Реализация состоит из нескольких независимых компонент: профилируемого приложения, системы имитации нагрузки, различных скриптов для анализа данных и построения графиков, а также приложения для воспроизведения нагрузки.

2.1. Профилируемое приложение

В качестве исследуемого приложения было решено выбрать базу данных веб-сайта. За основу была взята классическая связка LAMP: Linux, Apache, MySQL, PHP.

Веб-сайтом являлся блог на платформе Wordpress, выбранный за его популярность и простоту в использовании.

Для логирования IO операций было инструментировано ядро GNU Linux (см. работу Новожилова Евгения).

Для локализации действий профилируемого приложения его данные были вынесены на отдельный раздел, который подключался с опциями логирования. Это является обычной практикой при использовании систем хранения данных — обычно такие системы покупаются под одно приложение.

2.2. Имитация нагрузки

Для имитации настоящей нагрузки на веб-сайт было решено написать ботов, которые бы вели себя как настоящие пользователи.

Было рассмотрено несколько подходов к написанию ботов:

Прямые запросы к целевым страницам

Идея отправки запросов средствами инструментов вроде curl и wget отвёргнута в силу относительной сложности реализации: необходимо разбирать ответ, заботиться о сессии, перенаправлениях и пр. Если часть важной логики реализована на стороне клиента, то она не будет учтена. Так же загружается только сама страница, в то время как браузеры загружают всё содержимое, включая присоединённые документы вроде картинок, стилей и скриптов. Несмотря на то, что данное обстоятельство несущественно при текущей постановке задачи, кажется целесообразным разработать общий инструмент, имитирующий пользователя максимально точно.

Браузерное расширение, контролирующее поведение содержимого

Плохо масштабируется: сессионные данные внутри одного браузера едины, пришлось бы запускать множество различных экземпляров.

Специализированные инструменты UI-тестирования

Примером таких инструментов является PhantomJS, являющийся, по-сути, headless

версией браузера Chromium. Он имеет специальный API [2] для работы с окном и возможность отловить практически любое событие. Из недостатков следует отметить некоторую "сырость" приложения: иногда PhantomJS неожиданно завершается с сообщением об ошибке. Тем не менее, происходит это достаточно редко и только в экзотических ситуациях, поэтому мы остановимся на этом инструменте.

Было реализовано приложение, управляющее ботами, и панель управления к нему. Имеются 2 типа ботов: гости и авторы. Первые ходят по сайту, "читая" различные материалы, не выполняя никаких других действий. Вторые добавляют статьи и комментируют их, предварительно зарегистрировавшись и авторизовавшись.

Архитектура ботов довольно гибкая и легко расширяема: боты имеют роли и выполняют действия. Каждый бот имеет собственный список доступных действий, который пополняется в процессе выполнения других действий. Роль определяет возможности пользователя — какие действия он может совершать, а какие — нет. Действия отвечают за само взаимодействие с сайтом и содержат всю необходимую логику.

Реализованы следующие действия:

Open

Открытие страницы. Все ссылки, имеющиеся на странице и неизвестные ранее, добавляются в список доступных действий, становясь, тем самым, доступными для посещения. Для страниц, где возможно комментирование, создаётся соответствующее действие.

WriteArticle

Написание статьи. Это действие доступно по-умолчанию для ботов категории "Автор".

WriteComment

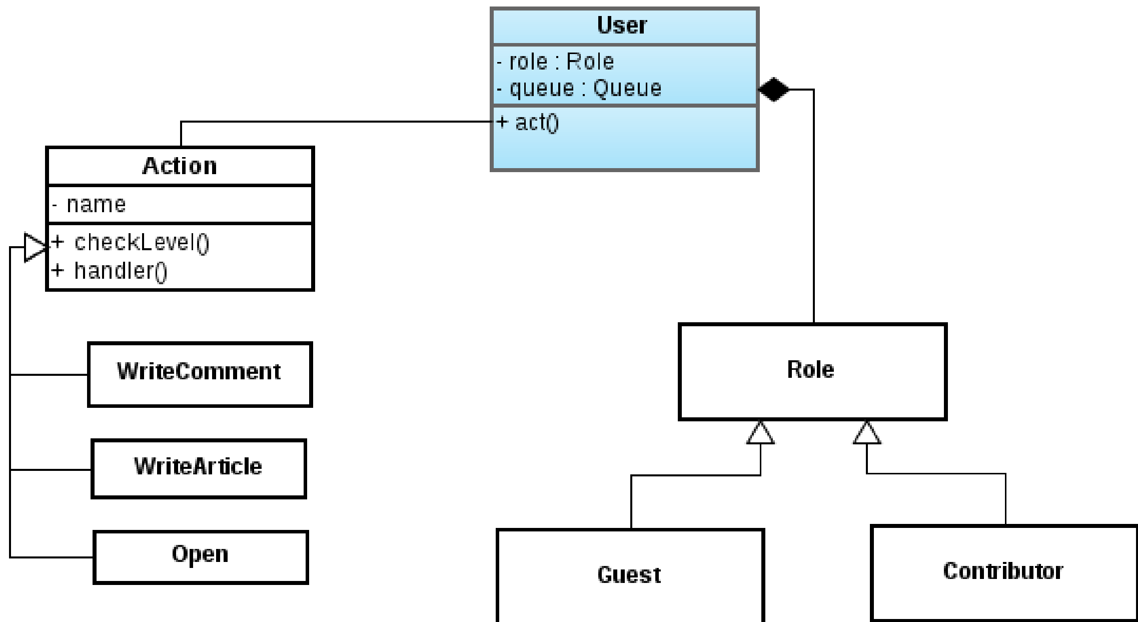
Написание комментария. Это действие становится доступно при открытии страницы с формой комментирования.

Register

Регистрация. Это действие выполняется при попытке авторизоваться, не являясь зарегистрированным пользователем.

Login

Авторизация. Это действие выполняется при попытке выполнить действие, требующее авторизации вроде написания комментария или статьи.



Для того, чтобы боты не опрашивали сервер слишком часто, введена случайная задержка между действиями в 3-25 секунд.

Несмотря на отсутствие в JavaScript концепции многопоточности, боты, тем не менее, работают параллельно за счёт использования асинхронного API.

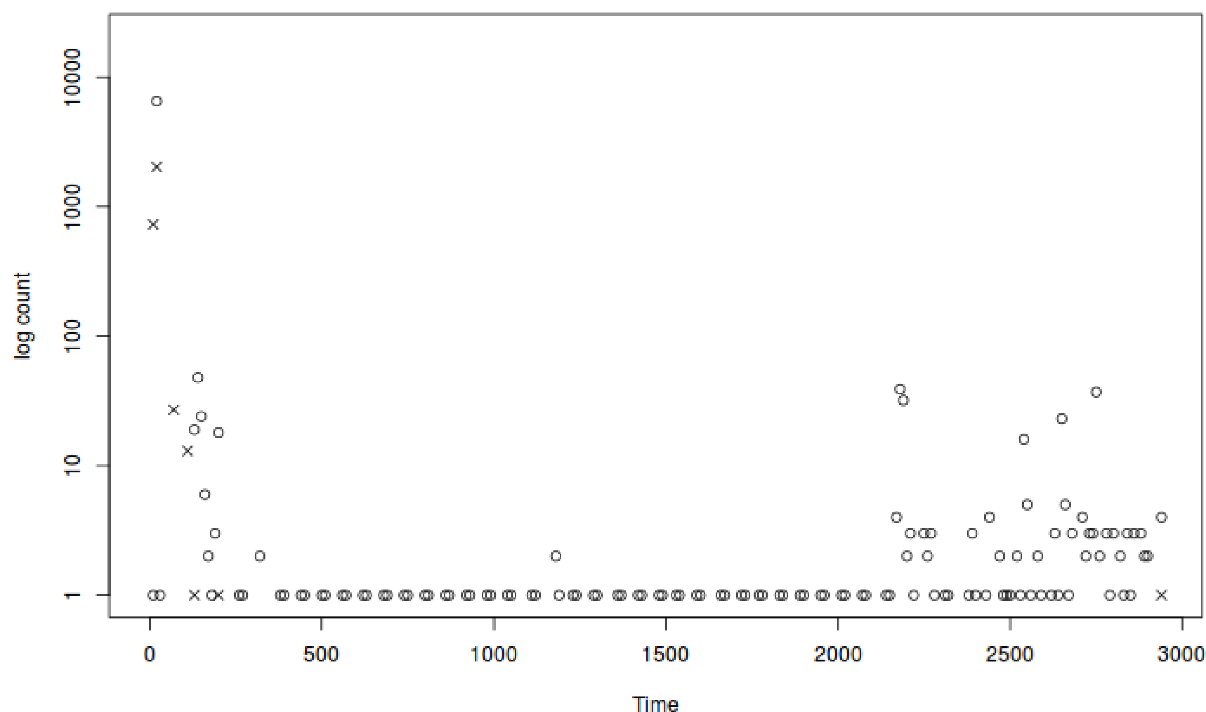
2.3. Воспроизведение профиля

Для воспроизведения нагрузки написано приложение на C++, принимающее лог взаимодействия профилируемого приложения с файловой системой и воспроизводящее его.

В ходе реализации выяснилось, что стандартные функции работы с файловой системой библиотеки libc вносят слишком много ”шума” (например, буферизуют данные), поэтому при таком подходе воспроизвести нагрузку не представлялось возможным. Было решено использовать системные вызовы, так как именно их и регистрирует логгер.

3. Результаты

Ниже приведён график в логарифмической шкале зависимости количества запросов чтения-записи от времени. Операции записи обозначены окружностями, операции чтения — крестиками.



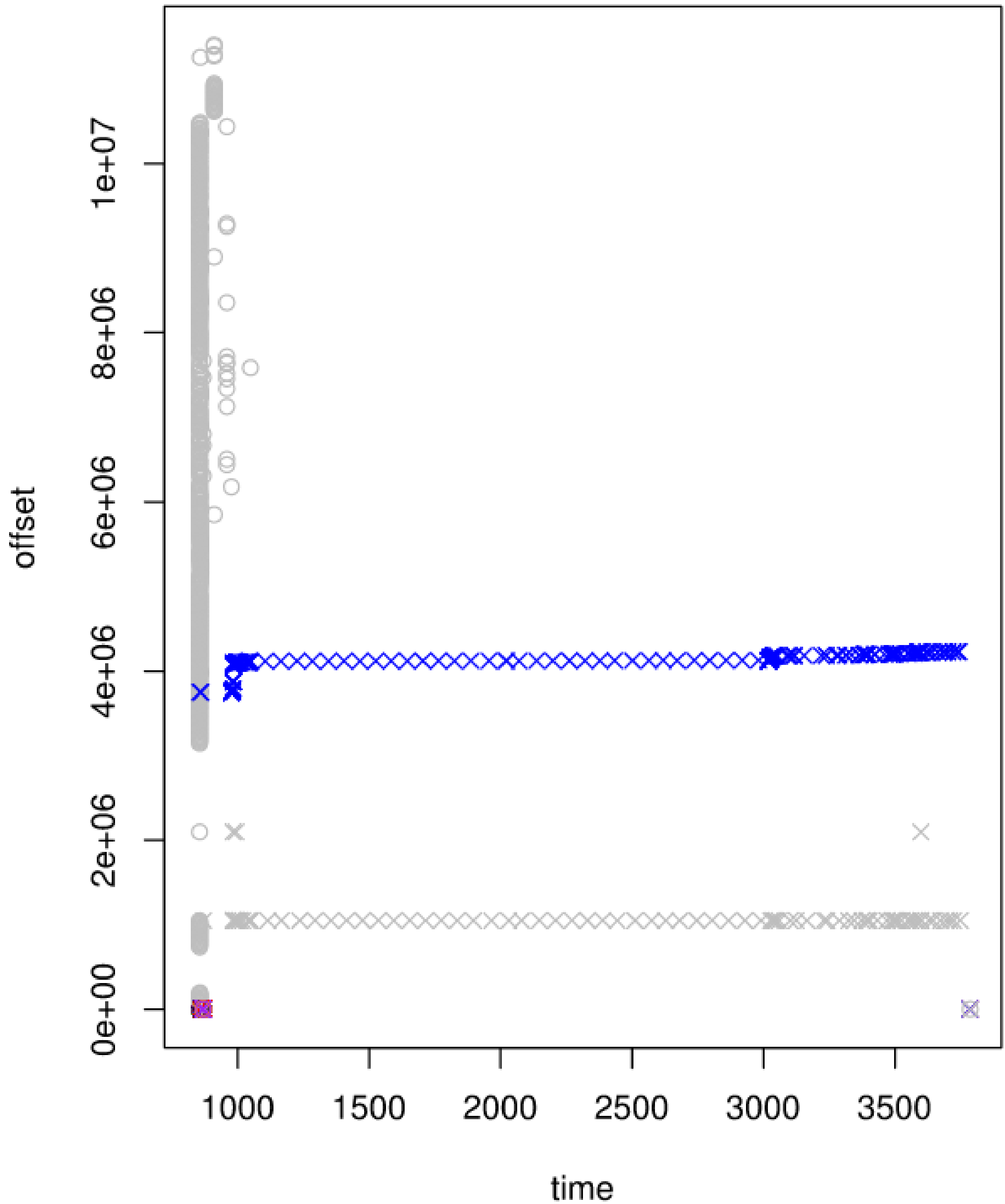
По графику видно, что большая часть операций чтения приходится на начало временного отрезка — момет запуска базы данных. В это время происходит ”инициализация” — подгружаются служебные таблицы, данные загружаются в оперативную память.

Далее в течение работы приложения операции чтения-записи затрагивают только основной файл данных и файл журнала.

При завершении работы базы данных все данные записываются на диск, что объясняет большое количество операций записи.

Согласно официальной документации [1], при использовании движка InnoDB все данные и индексы таблиц хранятся в одном файле, то время как файлы, соответствующие базам и таблицам, хранят лишь метаинформацию. Отсюда понятно, почему всё взаимодействие идёт с несколькими файлами: файл данных обновляется по мере их изменения, все действия дополнительно протоколируются в журнале на случай сбоя. По окончаю работы сервиса обновляется метаинформация на диске.

Подтверждением вышесказанного является график зависимости смещения операции от времени



Здесь окружности — чтения, а крестики — записи. Различным цветом выделены различные файлы. В самом начале читается один большой файл, а потом, в процессе работы приложения, происходят последовательные записи в файлы журнала и обновление данных, соответствующих изменяемым таблицам.

Таким образом, можно характеризовать работу MySQL приложения при нашем характере нагрузки следующим образом:

1. В момент запуска сервиса активно используются сервисные файлы вроде журнала. Такие файлы обычно имеют небольшой inode-номер, поскольку создавались вместе с установкой системы.

2. Во время работы приложения используются преимущественно сервисные файлы вроде журнала и основного файла.

3. При завершении работы обновляется метаинформация в файлах, соответствующих изменённым таблицам.

Данные характеристики могут быть использованы для определения приложения, использующего систему хранения данных с целью оптимизации работы. В случае, если мы знаем тип приложения и характер его взаимодействия с системой, можно предсказывать его будущее поведение и, например, размещать горячие файлы на более быстрых дисках.

Заключение

В ходе данной работы была реализована расширяемая тестовая платформа, проимитирована настоящая нагрузка на эту платформу, собраны данные этой нагрузки, проанализированы. Выявлены ожидаемые закономерности, отражающие процесс работы приложения. Реализовано приложение, позволяющее воспроизвести нагрузку.

Список литературы

[1] InnoDB Configuration // MySQL 5.0 Reference Manual. — http://dev.mysql.com/doc/refman/5.0/en/innodb-configuration.html.

[2] PhantomJS Documentation. — https://github.com/ariya/phantomjs/wiki.