ПРОФИЛИРОВАНИЕ ПРОГРАММНЫХ ПРИЛОЖЕНИЙ И РАЗРАБОТКА АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ АНАЛИЗА ПРОИЗВОДИТЕЛЬНОСТИ ПРОГРАММНЫХ СРЕДСТВ

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Шелкова В. В.

Сторожев Д. А. – магистр экон. наук, преподаватель

Сегодня, когда мы говорим о проблемах информационных систем, несомненно, одним из главных вопросов является оптимизация существующих программных приложений с целью повышения их эффективности. Усовершенствовать процесс производительности ПО можно путем поиска модулей, снижающих его эффективность, тем самым уменьшая время ожидания пользователя.

Профилирование ПО подразумевает измерение производительности данного программного средства или его частей с целью нахождения участков программы, на выполнение которых расходуется наибольшее количество времени и ресурсов.

Причина появления в коде программы участков, резко снижающих производительность, заключается в том, что подавляющее большинство вычислительных алгоритмов, так или иначе, сводятся к циклам, т. е. многократным повторениям одного фрагмента кода. Зачастую циклы обрабатываются не последовательно, а образуют иерархии, порождённые вложенностью циклов. В результате, большую часть всего времени выполнения программа проводит в циклах с наибольшим уровнем вложения. Именно на их оптимизацию направлен анализ производительности программ. Следует отметить, что оптимизация неэффективных, но редко использующихся участков кода не приведёт к заметному повышению производительности программного средства в целом. Следовательно, при анализе производительности важную роль играет статистическая информация, собранная во время выполнения программы.

В архитектуре разрабатываемой системы, разумно выделить две основные логические части: одна отвечает за получение информации о времени выполнения от исследуемого приложения и связана с механизмами низкоуровневого взаимодействия с исполняемым кодом (будем называть ее инструментирующая часть). Вторая часть отвечает за взаимодействие пользователя с программным средством, а также за обработку и представление данных, полученных от инструментирующего модуля (модуль визуализации). Вся работа с проектируемой системой будет связана с использованием JVM (Java virtual machine (англ.) - виртуальная машина платформы Java). Преимущество использования данной платформы заключается в том,

что исходный код программы (а также исходный откомпилированный байткод) не может подвергатся никаким модификациям. Также, Java-байткод является платформонезависимым языком, что достигается за счёт того, что он выполняется на виртуальной машине, следовательно, система является кроссплатформенной.

Обобщённый принцип работы программного средства выглядит следующим образом. Исследуемое приложение запускается с опцией «-javaagent», позволяющей зарегистрировать инструментирующий модуль программного средства в виртуальной машине. Исследуемое приложение выполняет порученную ему работу так, как если бы оно работало без каких-либо изменений, но так как каждый его метод в реальном времени дополнен логикой профилирования, во время его работы создаются .dump-файлы с журналом событий, наступающих в приложении. Журналирование завершается одновременно с завершением работы исследуемого приложения. После отработки приложения в режиме анализа, вся необходимая информация о процессе его работы доступна через созданные файлы. Пользователь запускает модуль, реализующий упомянутую выше логическую часть визуализации, и выполняет анализ данных на предмет падения производительности. Модуль визуализации, в свою очередь, обрабатывает .dump-файлы, строит на их основе древовидную структуру, отражающую цепочку вызовов в приложении, собирает статистику и предоставляет эти данные пользователю. Принцип работы разрабатываемого программного средства представлен на рисунке 1.

Разработанная система обладает следующими функциями и соответствует следующим требованиям:

1. функцией подключения к исследуемому приложению без вмешательства в его исходный код;
2. функцией сбора информации о работе приложения в реальном времени его выполнения;
3. работа исследуемого приложения в режиме анализа производительности не искажается по сравнению с его автономной работой;
4. результаты, предоставляемые разрабатываемым программным средством пользователю, скорректированы с учётом наведённых искажений работы исследуемого приложения;
5. исследуемое программное средство не претерпевает каких-либо изменений после отключения разрабатываемого профилировщика;
6. разрабатываемое программное средство преобразовывает собранную в результате профилирования информацию в удобный для восприятия вид. Пользователь имеет возможность просматривать полное

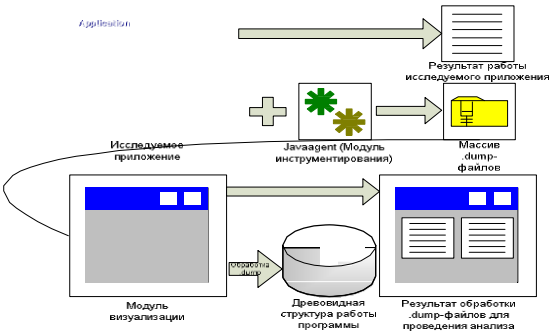

Рис. 1 – Принцип работы программного средства

дерево вызовов подпрограмм исследуемого приложения. Информация представлена в виде графа, который позволяет оценить, за счёт чего тот или иной вызов является продолжительным по времени. Каждый узел имеет цветовой индикатор, меняющийся от зелёного к красному в зависимости от того, сколько времени занимает данный вызов по отношению ко времени родительского вызова, что, в свою очередь, упрощает поиск уязвимых модулей (рисунок 2).

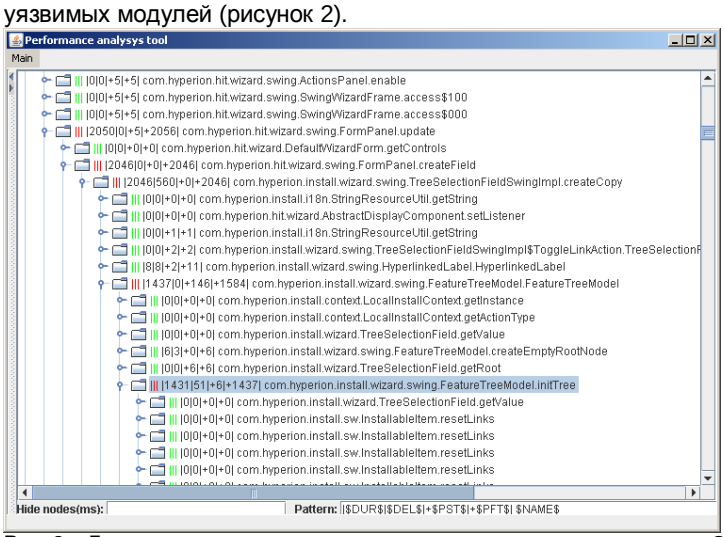

Рис. 2 – Дерево вызовов потока на примере тестирования приложения Oracle/Brio “Hyperion Installation Technology”

7. программное средство предоставляет статистику по отработанным подпрограммам, включающую в

себя количество вызовов конкретного метода, суммарное время выполнения всех вызовов данного метода и отдельно время выполнения каждого вызова, а также суммарное и отдельное собственное время выполнения (собственным временем выполнения называется время, которое было затрачено на отработку метода без учёта времени выполнения методов, вызванных в теле данного метода);

8. обеспечена возможность поиска конкретного метода по его имени, а также фильтрация методов по имени и временным показателям, что является безусловным плюсом.

Основной сложностью при создании системы анализа производительности стал выбор наиболее приоритетного подхода к профилированию программ. Нами был выбран подход «легковесного инструментирования». Данный подход подразумевает, что исходный код программы остаётся неизменным, однако исполняемый код дополняется инструкциями, которые выполняют необходимую логику журналирования. Реализация такого подхода сложна, так как необходимо прибегать к приёмам низкоуровневого программирования и вносить изменения в машинный, а не программный код. Для каждой платформы легковесное инструментирование представляет собой нетривиальную задачу. Для платформ, использующих промежуточный язык (.NET, Java), можно выделить общий принцип решения этой задачи. Так как исходный код программ на таких платформах сначала компилируется в промежуточный код, а затем этот промежуточный код загружается в виртуальную машину и исполняется на ней, то есть возможность перехватить промежуточный код после загрузки его в виртуальную машину, но до начала его исполнения, и произвести необходимые изменения, что было успешно реализовано и подтверждено на практике.

Таким образом, была разработана система анализа производительности программных приложений, которая обладает возможностью нахождения в исследуемом приложении модулей, снижающих эффективность исследуемого ПО, что выгодно позиционирует ее на фоне аналоговых систем профилирования.

Список использованных источников:

1. Касперски, К. Техника оптимизации программ / К. Касперски. – М. : БХВ-Петербург, 2003. – 464 с.
2. Адамс, М. Тестирование производительности web-приложений / М. Адамс. – СПб. : Питер, 2010. — 368 с.