РОЧЕВ К. В., БАЗАРОВА А. М. РАЗРАБОТКА АТРИБУТНО-ОРИЕНТИРОВАННОГО ПРОФИЛИРОВЩИКА ДЛЯ АНАЛИЗА БЫСТРОДЕЙСТВИЯ ФУНКЦИЙ ПРОГРАММНОГО КОДА НА ЯЗЫКЕ C#

УДК 004.4`2, ВАК 05.13.18, ГРНТИ 50.41.01

Разработка атрибутно-ориентированного профилировщика для анализа быстродействия функций программного кода на языке C#

Development of an attribute-oriented profiler for analyzing the program code functions performance in C#

К. В. Рочев, А. М. Базарова

Ухтинский государственный технический университет, г. Ухта

K. V. Rochev, A. M. Bazarova

Ukhta state technical university, Ukhta

В статье описана разработка комплекса программных средств для профилирования быстродействия выполнения участков кода. Система состоит из библиотеки, сочетающей возможность удобного подключения профилировщика с помощью атрибутов с возможностью отправки результатов замеров в глобальное хранилище данных и веб-системы аналитики, показывающей полученные результаты измерений. В совокупности данные решения позволяют обеспечить мониторинг затрат процессорного времени на отдельные участки кода при работе онлайн-серверов в реальном времени.

The article describes the development of a set of software tools for profiling the performance of code sections. The system consists of two parts: a library that combines easily connect a profiler using attributes with the ability to send measurement results to a global data warehouse and a web-based analytics system that shows the obtained measurement results. Together, these solutions allow you to monitor the cost of processor time for code sections and functions when running online servers in real time.

Ключевые слова: *профилировщик, быстродействие, оптимизация кода, производительность, C#, DOT.NET Framework.*

Keywords: *profiler, code optimization, performance, C#, DOT.NET Framework.*

Введение

Быстрое развитие программного обеспечения, компьютерной техники, создание новых и совершенствование существующих технологий построения программных продуктов, расширение спектра использования автоматизированных систем в современном мире обусловливают параллельное развитие всех составляющих процесса построения и внедрения ПО. Повышение

сложности и многокомпонентности современных программных проектов требуют специализированного подхода в процессе их создания и применения.

Разработка и совершенствование алгоритмов, реализующих, в частности, методы оптимального управления для построения высокоскоростных цифровых систем является актуальной задачей сегодняшнего дня, особенно в контексте того, что за последние десятилетия стремительными темпами растёт объём внедрения информационно-управляющих систем с интенсивным использованием программного обеспечения.

Из числа практических задач программной инженерии особое место занимает задача оптимизации работы программного обеспечения. При этом под оптимизацией понимается модификация программного обеспечения для улучшения его эффективности. Следует отметить, что термин «оптимизация программного обеспечения» (или «оптимизация программного кода») не подразумевает строгого решения задачи оптимизации (т.е. нахождение экстремума) в строгой математической постановке. Оптимизированное программное обеспечение обычно только соответствует заданным ограничениям, при этом, как правило, невозможно доказать, что полученные характеристики соответствуют экстремуму [1].

Основные инструменты и методы оптимизации включают в себя:

1. Межпроцедурный анализ;
2. Прямой анализ глобальных и статических переменных;
3. Оптимизация по профилю;
4. Встраивание;
5. Комплексная оптимизация ветвей.

Поскольку пропускная способность процессора зависит от постоянного поступления исполнительной инструкции, производительность может сильно ухудшаться, если ему приходится часто ждать, пока инструкции или данные будут получены из внешней памяти или записаны во внешнюю память. Роль компилятора в минимизации таких нежелательных событий имеет решающее значение для оптимальной производительности [2].

Оптимизация может происходить на разных уровнях, в зависимости от того, насколько она близка к машинному коду. Код может быть оптимизирован и на архитектурном уровне с помощью интеллектуальных шаблонов проектирования. На уровне исходного кода оптимизация может быть достигнута благодаря применению передовых методов кодирования и соответствующих инструментов. Кроме того, можно улучшить производительность приложения, внедрив руководства по стилю кодирования в рабочий процесс [3].

Ранее нами был проведен анализ производительности типовых функций среды .NET CLR, в окружениях Mono (Unity), .NET Framework (WPF/Windows Forms) и .NET Core [4, 5]. Результаты проделанной работы могут помочь принимать более эффективные решения при написании критических с точки зрения производительности участков кода. Однако оптимальное функционирование при одних входных данных не означает оптимального функционирования при всех возможных их вариантах. К тому же, обычно, нет

смысла оптимизировать весь код и лучше сосредоточиться на наиболее узких местах. Поиск таких мест, обычно, решается с помощью профилирования.

Существуют различные механизмы профилирования программного кода, некоторые из которых интегрированы непосредственно в среду разработки. Но несмотря на то, что в среде .NET существуют эффективные и удобные профилировщики, к сожалению, они могут функционировать не во всех окружениях, зачастую требуют много ресурсов или существенных усилий для интеграции.

Принимая во внимание вышеизложенное, **цель данной работы** заключается в разработке компактного механизма измерения времени выполнения участков кода и анализа его быстродействия для дальнейшего повышения производительности критических участков высоконагруженных информационных систем.

Для достижения поставленной цели необходимо решить следующие задачи:

1) разработать механизм измерения быстродействия программного кода;
2) обеспечить возможность достаточно удобного подключения реализуемого механизма к отдельным участкам анализируемой системы;
3) обеспечить возможность функционирования в любых .NET окружениях;
4) обеспечить относительно низкие накладные расходы при применении;
5) реализовать возможность сбора данных профилирования на протяжении больших временных интервалов и демонстрации их в реальном времени.

Для достижения поставленной цели было принято решение реализовать библиотеку, обеспечивающую возможность сбора данных о результатах быстродействия кода и панель мониторинга полученных результатов. Разработанный механизм состоит из следующих основных частей:

1) Измеритель быстродействия кода на основе атрибутов;
2) Подсистема предварительного агрегирования и отправки данных;
3) Подсистема аналитики.

Измеритель быстродействия кода на основе атрибутов

Основная часть проекта – библиотека для измерения быстродействия. Для обеспечения удобства ее использования был использован подход на основе атрибутов. Атрибуты предоставляют средство для связывания метаданных или декларативной информации с кодом (сборки, типы, методы, свойства и т. д.) [6]. Мы используем их для обеспечения возможности подключения методов, выполняющих измерение времени выполнения кода к событиям начала и окончания выполнения измеряемых функций программы. Для этого использовано решение MethodBoundaryAspect.Fody [7], позволяющее отслеживать подобные события с помощью атрибутов.

Благодаря этому, в разработанном решении есть возможность отметить атрибутом для измерения любой метод проекта, либо целый класс. В случае применения атрибута к классу будет измеряться быстродействие всех его методов. Атрибут [`ProfilerAttribute`] подключают отслеживание времени входа в метод и выхода из него и запись разницы с указанием названия и

аргументов метода. В случае возникновения исключения в процессе выполнения метода, осуществляется запись времени и подсчет количества ошибок.

Далее представлены основные части кода атрибута профилировщика.

```
public sealed class ProfilerAttribute : OnMethodBoundaryAspect
{
    private string GetName(MethodBase m) =>
        $"{m.DeclaringType.Name}.{m.Name}" +
        $"({m.GetParameters().Select(x => x.Name).ToStringJoin()})";

    public override void OnEntry(MethodExecutionArgs args)
    {
        TimeProfiler.Begin(GetName(args.Method));
    }

    public override void OnExit(MethodExecutionArgs args)
    {
        var name = GetName(args.Method);
        TimeProfiler.End(name);
    }

    public override void OnException(MethodExecutionArgs args)
    {
        var name = GetName(args.Method);
        TimeProfiler.End(name);
        TimeProfiler.OnException(name);
    }
}
```

Методы OnEntry, OnExit, OnException обеспечивают отслеживание входа и выхода в измеряемые функции.

Функция GetName формирует название измеряемого метода с помощью отражения (Reflection) [8] для дальнейшего агрегирования и показа.

Подсистема предварительного агрегирования и отправки данных

Класс TimeProfiler обеспечивает непосредственное выполнение измерений с помощью инструмента Stopwatch и группировку данных по названию и аргументам измеряемых методов на основе словаря.

Далее приведен интерфейс, демонстрирующий основные методы класса `TimeProfiler`:

```
ITimeProfiler
{
    TimeProf Begin(string n);
    void End(string name);
    void Sample(string name, Action action);
    TimeProf GetTime(string name);
    TimeProfile CalcTime();
}
```

Методы Begin-End используются для указания начала и окончания измерения. Sample позволяет измерить время работы метода, переданного по ссылке. GetTime получает результаты замеров по указанному методу.

Метод CalcTime позволяет объединять данные профилирования блоками за указанный интервал, например за 1 минуту (для минимизации объёмов хранения и передачи) и отправлять их в базу данных.

Группировку данных за указанный интервал по сигнатуре профилируемых методов можно представить формулами 1 и 2.

$$T_f = \sum_{i=0}^{n} t_i \quad (1)$$

где T_f – суммарное время выполнения отдельной функции за интервал измерения

f – сигнатура функции, состоящая из её названия и списка аргументов

t – время выполнения функции, тиков

n – количество вызовов функции

$$T = \sum_{i=0}^{n} T_{fi} \quad (2)$$

где T – суммарное время выполнения всего кода за интервал измерения

T_f – суммарное время выполнения отдельной функции за интервал измерения

n – количество различных профилируемых функций

На текущий момент в первой версии системы в качестве базы данных используется облачный кластер mongodb, однако реализация хранения может быть легко заменена без изменений остальной части системы.

Подсистема аналитики

Полученные и собранные в БД сведения о профилировании могут быть изучены с помощью веб-интерфейса. Подсистема для изучения данных профилирования реализована на ASP.NET Core. Она дополнительно агрегирует результаты замеров (по формуле 3) по указанным параметрам и обеспечивает возможность их мониторинга в реальном времени.

$$dT = \frac{\sum_{i=0}^{n} T_{fi}}{\sum_{i=0}^{n} T_i} \quad (3)$$

где dT – доля времени выполнения функции

T – суммарное время выполнения всего кода за интервал измерения

T_f – время выполнения отдельной функции за интервал измерения

n – количество интервалов измерения

На рисунке 1 представлен интерфейс панели аналитики, показывающий нагрузку по серверам и детализацию затрат времени измеряемых функций по выбранному серверу.

Затраты процессорного времени на работы профилировщика также измеряются и выводятся в отчёте. При разном уровне детализации измерений это время меняется. Например, при мониторинге работы наших серверов, где записывается время всех поступающих запросов и работа части внутренних функций (в совокупности около 500 часто используемых методов), оно составляет на данный момент 1-5% общей вычислительной нагрузки.

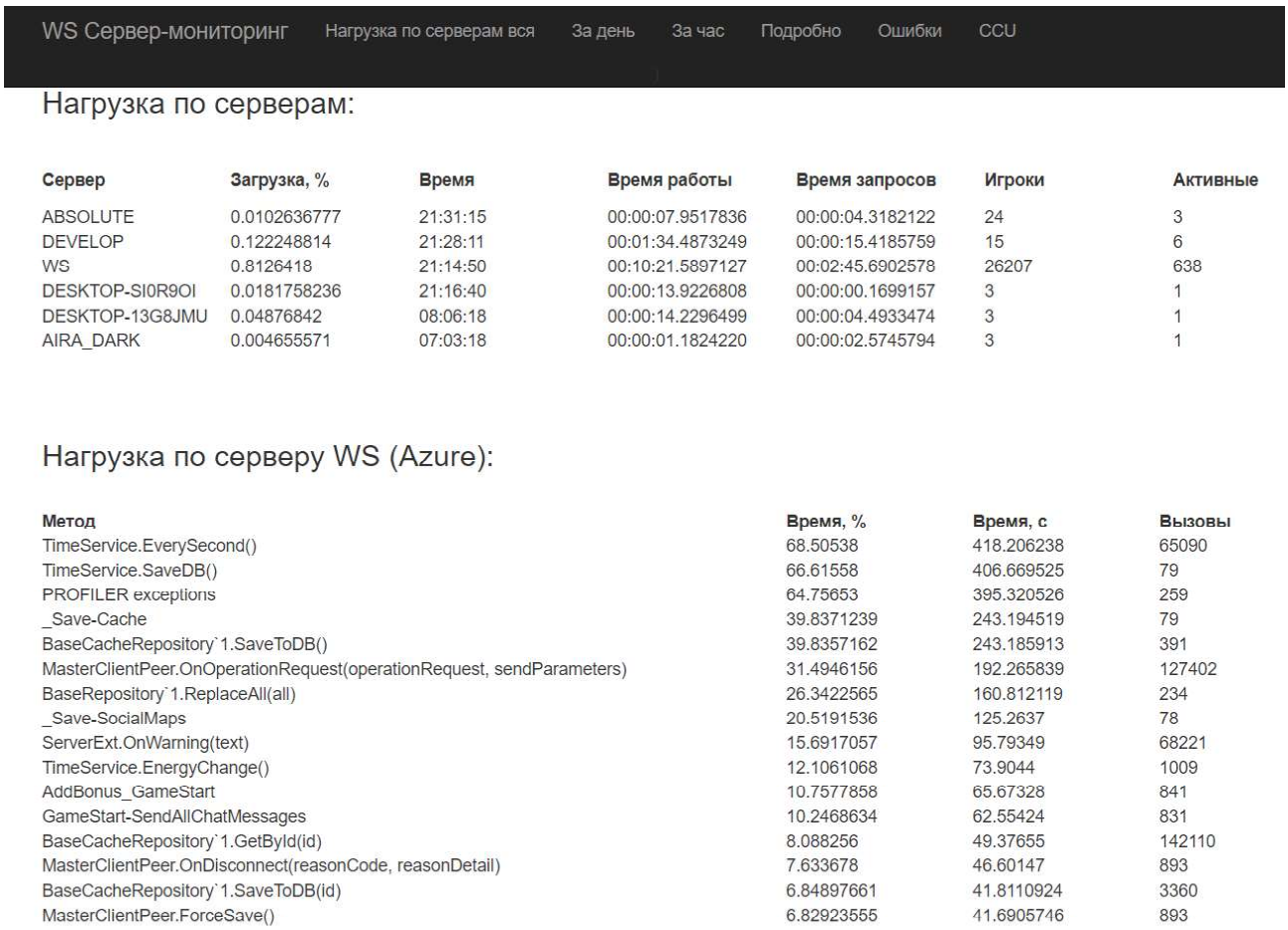

Рисунок 1. Интерфейс панели аналитики

Заключение

Результатом работы стал комплекс программных средств, состоящий из библиотеки, обеспечивающей профилирование быстродействия кода, сочетающей возможность удобного подключения с помощью атрибутов с возможностью отправки результатов замеров в глобальное хранилище и веб-системы аналитики, показывающей полученные результаты измерений. Подключение профилировщика требует внесения изменений в исходный код, но эти изменения минимальные – они заключаются в указании атрибутов у нужного метода или класса. Каждую часть проекта, естественно, можно использовать отдельно, однако в комплексе они обеспечивают возможность детального мониторинга работы и производительности критических функций нагруженных онлайн-систем в реальном времени.

В дальнейшем планируется оптимизировать механизм измерения за счет кэширования данных, получаемых с помощью отражения, а панель мониторинга и аналитики расширить дополнительными графическими отчетами, такими как гистограмма средней и максимальной нагрузки за период по интервалам. Немаловажным видится также добавление группировки функций по иерархии вызовов.

Список использованных источников и литературы

1. Зеленая ИТ-инженерия. В 2-х томах. Том 1. Системы, индустрия, социум / Под ред. Харченко В.С. 2014. – 594 с.
2. Perrin, G. Adaptive calibration of a computer code with time-series output // Reliability engineering & system safety. 2020. Volume 196; pp 176-182.
3. Li, Genghui Multifactorial optimization via explicit multipopulation evolutionary framework // Information sciences. 2020. Volume 512; pp 1555-1570.
4. Рочев К. В. Анализ быстродействия типовых операций языка C# на платформах DOT.NET и MONO // Информационные технологии в управлении и экономике. 2019. № 1. С 7-20.
5. Рочев К. В., Семяшкина А. В. Анализ быстродействия строковых операций языка C# на разных платформах // Информационные технологии в управлении и экономике. 2020. № 2. С 13-20.
6. Атрибуты (C#) https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/concepts/attributes/
7. Репозиторий проекта MethodBoundaryAspect.Fody https://github.com/becdetat/Aspects.Fody
8. Reflection (C#) https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/reflection.

List of references

1. Green IT engineering. In 2 volumes. Volume 1. Systems, industry, social-mind. Ed. Kharchenko V. S. 2014. 594 p.
2. Perrin, G. Adaptive calibration of a computer code with time-series output. Reliability engineering & system safety. 2020. Volume 196; pp 176-182.
3. Li, Genghui Multifactorial optimization via explicit multipopulation evolutionary framework. Information sciences. 2020. Volume 512; pp 1555-1570.
4. Rochev K. V. Standard operations performance analysis of the C# language on platforms DOT.NET and Mono. Information Technologies in management and Economics. 2019. No. 1. pp 7-20.
5. Rochev K. V., Semyashkina A.V. Analysis of the performance of C# string operations on different platforms. 2020. No. 2. pp 13-20.
6. Attributes (C#) https://docs.microsoft.com/ru-ru/dotnet/csharp/programming-guide/concepts/attributes/
7. Repository of the MethodBoundaryAspect project.Fody https://github.com/becdetat/Aspects.Fody
8. Reflection (C#) https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/reflection.

СВЕДЕНИЯ ОБ АВТОРАХ

Артамонова Ирина Александровна

Курганский государственный университет, г. Курган;
кандидат экономических наук,
доцент кафедры «Учет и внешнеэкономическая деятельность»

Artamonova Irina Alexandrovna

Kurgan State University, Kurgan;
Candidate of Economic Sciences,
Associate Professor, Associate Professor of the Department of Finance and Economic Security

E-mail: baturina76@mail.ru

Базарова Ирина Александровна

Ухтинский государственный технический университет, г. Ухта;
доцент кафедры вычислительной техники, информационных систем и технологий

Bazarova Irina Aleksandrovna

Ukhta State Technical University, Ukhta; Associate Professor, Department of Computer Engineering, Information Systems and Technologies

E-mail: ibazarova@ugtu.net

Базарова Анна Максимовна

Ухтинский государственный технический университет, г. Ухта;
старший преподаватель кафедры электроэнергетики и метрологии

Bazarova Anna Maksimovna

Ukhta State Technical University, Ukhta;
Senior Lecturer of the Department of Electric Power Engineering and Metrology

E-mail: abazarova@ugtu.net

Батурина Ирина Николаевна

Курганский государственный университет, г. Курган;
кандидат экономических наук,
доцент, доцент кафедры Финансы и экономическая безопасность

Baturina Irina Nikolayevna

Kurgan State University, Kurgan;
Candidate of Economic Sciences,
Associate Professor, Associate Professor of the Department of Finance and Economic Security

E-mail: baturina76@mail.ru

Бухтиярова Татьяна Ивановна

Российская академия народного хозяйства и государственной службы при Президенте Российской Федерации, Челябинский филиал, г. Челябинск; доктор экономических наук,

Bukhtiyarova Tatiana Ivanovna

Russian Academy of National Economy and Public Administration under the President of the Russian Federation, Chelyabinsk branch, Chelyabinsk; Doctor of Economics, Professor, Professor of the Department of Economics, Finance and Accounting