УДК 004.9

Моделирование нагрузки на базу данных с помощью асинхронного программирования

Николаев Сергей Валерьевич
Приамурский государственный университет им. Шолом-Алейхема
магистрант

Аннотация
Любой программный продукт перед выходом в эксплуатацию проходит несколько этапов тестирования. Одним из таких тестирований является нагрузочное тестирование. С помощью данного теста имитируется количественная нагрузка на приложение. В данной статье будет рассмотрен один из способов реализации подобного теста с помощью асинхронного программирования. Целью теста будет база данных MS SQL Server, а реализован алгоритм будет с помощью Microsoft .Net Framework-а, а именно языка программирования C#.
Ключевые слова: Информационные технологии, C#, .Net. асинхронное программирование, моделирования промышленной среды, нагрузочное тестирование.

Modeling the load on the database using asynchronous programming

Nikolaev Sergey Valerievich
Sholom-Aleichem Priamursky State University
Master student

Abstract
Any software product before going into service passes several stages of testing. One of these tests is load testing. This test simulates the quantitative load on the application. In this article, one of the ways of implementing such a test using asynchronous programming will be considered. The purpose of the test will be the Ms SQL Server database, and the algorithm will be implemented using the Microsoft .Net Framework, namely the C # programming language.
Keywords: Information technology, C #, .Net. asynchronous programming, industrial environment simulation, load testing.

Любой программный продукт перед выходом в эксплуатацию проходит несколько этапов тестирования. Одним из таких тестирований является нагрузочное тестирование. Суть данного теста, смоделировать работу системы в таком количестве запросов или операций, с которым информационной системе придется столкнуться при работе в промышленной среде. Часто нагружают больше, чем может понадобиться, чтоб система имела запас в ресурсах, в реальных условиях работы.

Данный тип тестирования используется в многопользовательских приложениях, когда число пользователь может варьироваться в зависимости от времени. Таким образом у приложения могут быть пиковые часы нагрузки, когда наиболее вероятно, что оно начнет тормозить или вообще выдавать отказ в обслуживании. Существуют даже виды хакерских атак, целью которых является перегрузить систему запросами что приводит к тому, что система не может быть использована пользователями, клиентами, а владелец приложения или системы теряет деньги и репутацию. Но подробное рассмотрение подобных атак не являются целью данной статьи.

Нагрузочные тесты, нужны для того, чтобы понять, может ли информационная система справиться с планируемым объемом работы. Часто целью таких тестов являются веб-сервера, на которых развернуто веб-приложение, сервера баз данных.

В данной статье будем продемонстрирован способ, с помощью которого можно провести нагрузочное тестирование, используя один компьютер. Безусловно, на тот максимум нагрузки, который возможно сгенерировать влияют технические характеристики компьютера, но используя асинхронное программирования, ресурсы будут задействованы наиболее оптимальным способом.

Нам необходимо имитировать работу большого числа пользователей, что говорит о том, что они не будут посылать запросы к базе данных строго, друг за другом. Отсюда следует, что будт использоваться параллельные вычисления. Кратко пройдем по терминам:

- конкурентное исполнение (concurrency);
- параллельное исполнение (parallel execution);
- многопоточное исполнение (multithreading);
- асинхронное исполнение (asynchrony).

Конкурентность (concurrency) - это наиболее общий термин, который говорит, что одновременно выполняется более одной задачи. Конкурентное исполнение - это самый общий термин, который не говорит о том, каким образом эта конкурентность будет получена: путем приостановки некоторых вычислительных элементов и их переключение на другую задачу, путем действительно одновременного исполнения, путем делегации работы другим устройствам или еще как-то.

Параллельное исполнение (parallel execution) - подразумевает наличие более одного вычислительного устройства (например, процессора), которые будут одновременно выполнять несколько задач.

Многопоточность (multithreading) - это один из способов реализации конкурентного исполнения путем выделения абстракции "рабочего потока" (worker thread). Потоки "абстрагируют" от пользователя низкоуровневые детали и позволяют выполнять более одну работу "параллельно". Операционная система, среда исполнения или библиотека прячет подробности того, будет многопоточное исполнение конкурентным (когда потоков более чем физических процессоров), или параллельным (когда число

потоков меньше или равно числу процессоров и несколько задач физически выполняются одновременно).

Асинхронность (asynchrony) Асинхронность (asynchrony) подразумевает, что операция может быть выполнена кем-то на стороне: удаленным веб-узлом, сервером или другим устройством за пределами текущего вычислительного устройства.

Основное свойство таких операций в том, что начало такой операции требует значительно меньшего времени, чем основная работа. Что позволяет выполнять множество асинхронных операций одновременно даже на устройстве с небольшим числом вычислительных устройств.

Цель тестирования сгенерировать запрос и отреагировать на него, только когда придет ответ, обработанный базой данных. Пока ответ не пришел, программе нет смысла тратить на этот поток ресурсы. Таким образом асинхронное исполнение подходит больше всего.

Для тестирования нам понадобятся следующие входные данные:

- количество запросов, которое надо сгенерировать
- время, за которое надо их сгенерировать
- sql-запрос, который будет исполняться

Рассмотрим приложение. Выполнено оно в виде консольного приложения, интерфейс которого можно увидеть на Рисунке 1.

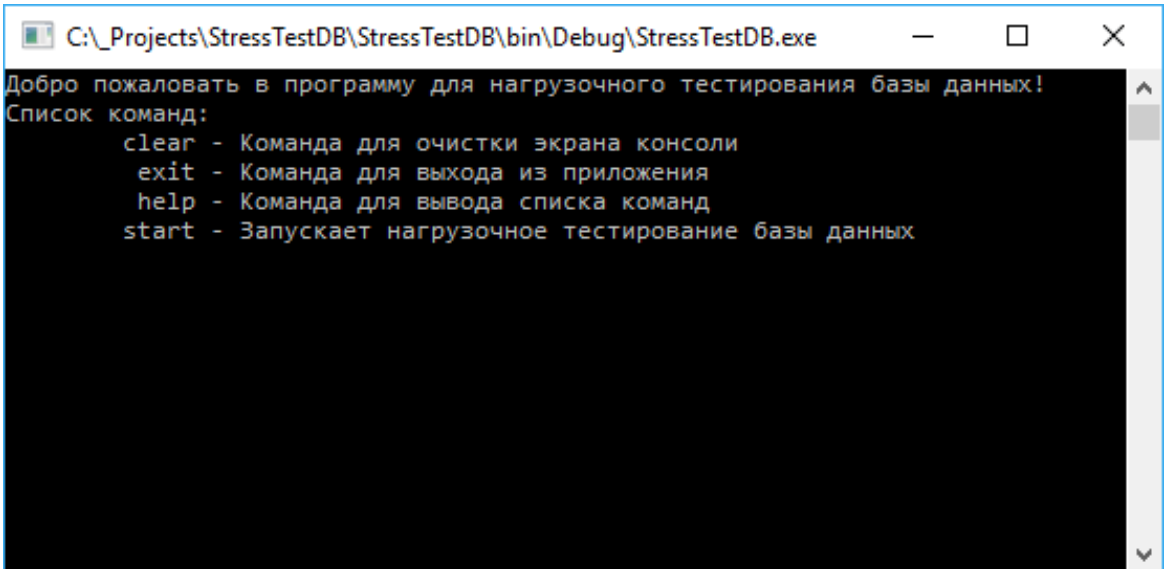

Рисунок 1 – Интерфейс приложения

Нас интересует команда **start**, так как именно она запускает нагрузочное тестирование. Рассмотрим ее исходный код на рисунке 2.

```
1   using StressTestDbLibrary;
2
3   namespace StressTestDB.Commands
4   {
5     public class StartTestCommand: Command
6     {
7       public override string Name { get; set; } = "start";
8       public override string Description { get; set; } =
9                   "Запускает нагрузочное тестирование базы данных";
10
11      public override void Execute()
12      {
13        var task = StressTest.StartAsync();
14        task.Wait();
15      }
16    }
17  }
```

Рисунок 2 – Исходный код команды start

Самое важное на строках 13-14. В них запускается экземпляр класса **Task** (представляет асинхронную операцию), а затем исполняется метод **task.Wait**, который не позволяет приложению завершиться раньше, чем закончиться исполнение **StressTest.StartAsync**. Рассмотрим **StartAsync** (Рисунок 3).

```
1   using System.Collections.Generic;
2   using System.Threading.Tasks;
3
4   namespace StressTestDbLibrary
5   {
6     public static class StressTest
7     {
8       public static async Task StartAsync()
9       {
10        var clients = new Client[TestSettings.CountQuerys];
11        var tasks = new List<Task>();
12        for (var i = 0; i < clients.Length; i++)
13        {
14          clients[i] = new Client(i);
15          tasks.Add(clients[i].ExecuteQueryAsync());
16        }
17
18        await Task.WhenAll(tasks);
19      }
20    }
21  }
```

Рисунок 3 – Исходный код классы StressTest

В данном методе (строки 10-18) мы подготавливаем массив класса **Client** – это специально разработанный класс, который и имитирует клиента, который будет осуществлять запрос. При создании массива в качестве размерности используется параметр **TestSettings.CountQuerys** – данный параметр хранит в себе количество запросов и берется из специального класса **TestSettings**, который содержит в себе настройки тестирования (количество запросов, время, сам запрос).

Далее в цикле инициализируем значение **Client** через конструктор, и, после окончания цикла, запускаем все клиенты на исполнение запроса, одновременно. Заглянем внутрь класса **Client**, чтоб разобрать работу метода **ExecuteQueryAsync** (Рисунок 4).

```
1  public class Client
2  {
3      private readonly string queryText;
4      private readonly Random random;
5      private readonly int indexNumber;
6
7      public Client(int indexNumber)
8      {
9          this.indexNumber = indexNumber;
10         random = new Random(indexNumber);
11         queryText = string.Format(TestSettings.QueryTextTemplate, random.Next(0,
           TestSettings.CountDocuments));
12     }
13
14     public async Task ExecuteQueryAsync()
15     {
16         await Task.Delay(random.Next(1, TestSettings.Time));
17         using (var connection = new SqlConnection(TestSettings.ConnectionString))
18         {
19             await connection.OpenAsync();
20             var command = new SqlCommand
21             {
22                 CommandText = queryText,
23                 Connection = connection,
24                 Transaction = connection.BeginTransaction($"{indexNumber}")
25             };
26             await command.ExecuteNonQueryAsync();
27             command.Transaction.Commit();
28         }
29     }
30 }
```

Рисунок 4 – исходный код класса Client

В конструкторе класса **Client** (строки 8-12) задаются параметры при инициализации, формируется текст sql запроса, где в тексте шаблона запроса меняются значения для того, чтоб разнообразить генерируемые запросы. Таким образом сервер базы данных, должен каждому клиенту отдать уникальный набор данных.

На строке 16 вводиться задержка с помощью генератора случайных чисел. В данном виде тестирования он имеет равномерное распределение, но при желании, можно переопределить функцию генерации задержки, создав повышенную нагрузка в определенный момент времени, имитирую «час пик».

Далее на строках 20-25 создаются параметры sql-команды, формируемая в виде транзакции. Далее команда исполняется. Стоит еще раз обратить внимание, данный код, исполняется одновременно, имитируя работу клиентов базы данных. Так же отметим, что пока происходит задержка в строке 16, программа не тратит ресурсы, а занимается генерацией других запросов.

Для трассировки запросов к базе данных можно использовать инструмент встроенный в Ms Sql Server под названием Sql Server Profiler. Визуально данный инструмент представлен на рисунке 5.

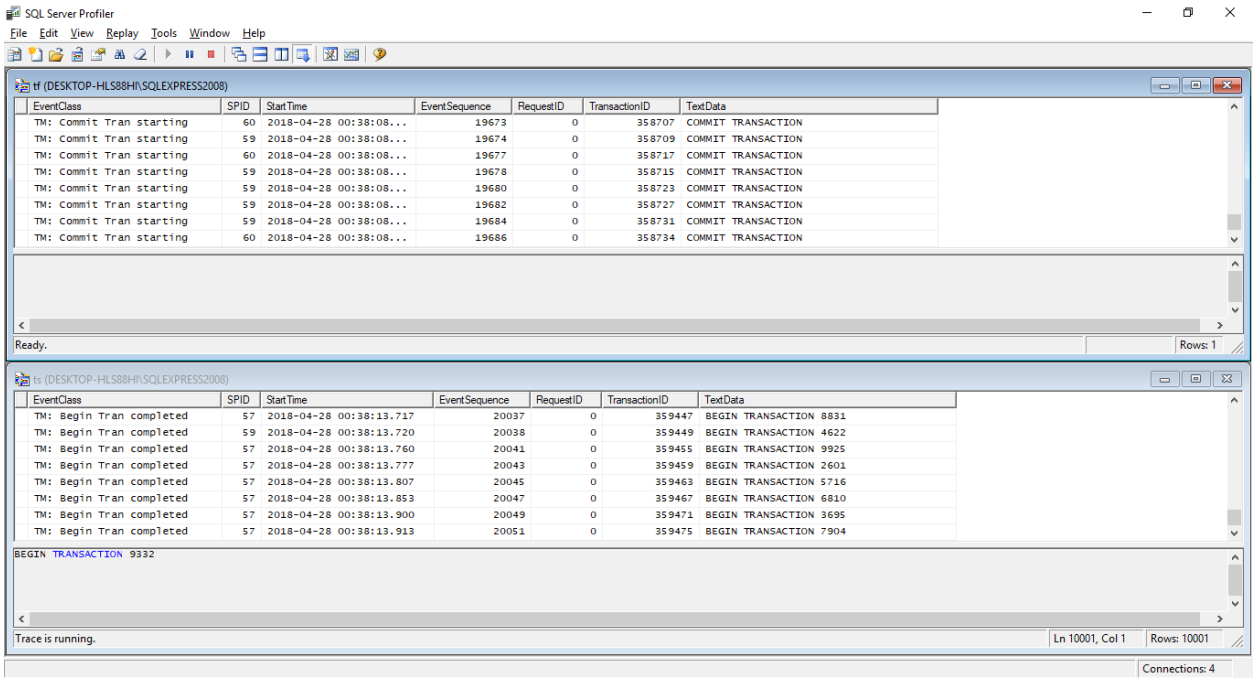

Рисунок 5 – Sql Server Profiler

С его помощью можно собрать информацию и при желании записать ее в файл или в таблицу базы данных.

Таким образом можно создать нагрузочное тестирование на сервер базы данных MS Sql Server. Стоит отметь, что стоит подменить класс клиента и данный алгоритм можно использовать и для тестирования другого рода систем, например веб-сервер приложения.

Библиографический список

1. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C#. М.: Питер, 2013. 928 с.
2. Троелсен Э. Язык программирования C# 5.0 и платформа .NET 4.5 . М.:

Вильямс, 2015. 486 с.
3. Шилдт Г. C# 4.0: полное руководство. М.: Вильямс, 2011. 1056 с.