CHAPTER 18

Performance and Profiling

Performance analysis and tuning are often done after release—but we really do not have to wait until then. Instead we should integrate the profiling into our daily routines to make sure we always keep an eye on the performance of our applications.

In software engineering, profiling is a form of dynamic application analysis that measures, for example, the memory usage, the usage of particular instructions, or frequency and duration of function calls. The most common use of profiling information is to help developers optimize their applications.

Profiling is achieved by instrumenting either the application source code or the binary executable. TFS and Visual Studio include profiling tools that will help us with this task.

Profiling Overview

There are many ways a profiling tool can gather and output data. The profiling tools can collect different kinds of information like performance counters, hardware interrupts, code instrumentation, and operating system information just to mention a few. Let’s take a brief look at some ways the profiling tools can collect data:

- **Event-based profilers:** Some programming languages offer an event-based profiler. In Java we have the JVMTI (JVM Tools Interface) API, formerly JVMPI (JVM Profiling Interface), which provides hooks to profilers, for trapping events like calls, class-load, unload, thread enter leave. In .NET we can attach a profiling agent as a COM server to the CLR using Profiling API. Like Java, the runtime then provides various callbacks into the agent, for trapping events like method JIT / enter / leave, object creation, etc.

- **Statistical profilers:** Some profilers operate by sampling where the profiler probes the target application’s program counter at regular intervals using operating system interrupts. Sampling profiles are typically less numerically accurate and specific, but allow the target program to run at near full speed, which helps the profiler detect issues that would be hard to catch otherwise. Often a drawback with some profiling methods is that the application takes a performance hit during profiling. The resulting data from statistical profilers are not exact, but a statistical approximation.

- **Instrumenting profilers:** Some profilers instrument the target program with additional instructions to collect the required information. This profiling method can affect the performance of the program, which could cause inaccurate results. Instrumenting will always have some impact on the program execution, typically always slowing it. However, instrumentation can be very specific and be carefully controlled to have a minimal impact. The impact on a particular program depends on the placement of instrumentation points and the mechanism used to capture the trace.
There are two major outputs a profiler might produce:

- A statistical summary, or profile as it is also called, of the events the profiler has observed. Summary profile information is often shown annotated against the source code statements where the events occur, so the size of measurement data is linear to the code size of the program.

- A trace, which is a stream of recorded events. For sequential programs, a summary profile is usually sufficient, but performance problems in parallel programs (waiting for messages or synchronization issues) often depend on the time relationship of events, thus requiring a full trace to get an understanding of what is happening.

Let's now take a look at what Visual Studio has to offer when it comes to profiling.

### Profiling In Visual Studio 2012

Visual Studio 2012 offers some great tools when it comes to profiling. These tools let us measure, evaluate, and find performance-related issues in our code. The tools are fully integrated into the Visual Studio IDE to provide a seamless and approachable user experience; however the profiling tools are also available from the command line if we would like that approach.

The profiling tools in Visual Studio offer five ways to collect and analyze data:

- **Sampling**: This is the recommended way to start exploring application performance. The sampling method is nonintrusive and has little impact on the execution of the application being monitored. Sampling profiling interrupts the processor at given intervals and collects the executing call stack. Based on these statistics, call counts are calculated so we can analyze execution patterns.

- **Instrumentation**: When using the instrumentation the profiled code gets injected so exact call times can be recorded. In the analysis report we use the following values to understand the application performance:
  - Elapsed Inclusive: Total time spent executing the method (including calls to other methods).
  - Application Inclusive: Same as Elapsed Inclusive but with calls to the operating system excluded.
  - Elapsed Exclusive: Time spent executing code in the method.
  - Application Exclusive: Same as Elapsed Exclusive but with calls to the operating system excluded.

- **Concurrency**: This profiling mode collects data about multithreaded application. The analysis reports resource contention and there is also a visualization of how the application executed which we can use to trace bottlenecks in the system.

- **.NET Memory**: The .NET memory profiling collects information when .NET objects are allocated and returned. Memory profiling can be used together with sampling or concurrency proofing.

- **Tier Interaction**: This mode collects information about ADO.NET calls made to a SQL Server database. The analysis data can help us understand database performance from an application perspective. Tier Interaction can be used with any of the other profiling modes.