Testing the Performance of Web Applications

Before you start making changes to your web application, you need to know if your application is performing well or not—is it up to the requirements specified in the SLA (service level agreement)? Does it behave differently under load? Are there any general issues which can be improved? To know all this and more we need to use testing and monitoring tools that can help us identify pitfalls and bottlenecks in our web application.

In Chapter 2 we discussed some general-purpose analysis tools to detect performance issues in code, such as the Visual Studio and ANTS profilers, but there are additional tools that can assist you in testing, measuring, and investigating the "web" part of your application.

This is just a brief introduction to the world of web application performance testing. For a more thorough description and guidance on how to plan, execute, and analyze web application performance testing, you can refer to the "Performance Testing Guidance for Web Applications" MSDN article (http://msdn.microsoft.com/library/bb924375).

Visual Studio Web Performance Test and Load Test

Among the testing features that are available in Visual Studio Ultimate is the Web Performance test, which enables you to evaluate the response time and throughput of a web application. With the Web Performance test, you can record the HTTP requests and responses which are generated when browsing a web application, as shown in Figure 11-1. (This is supported directly in Internet Explorer only.)

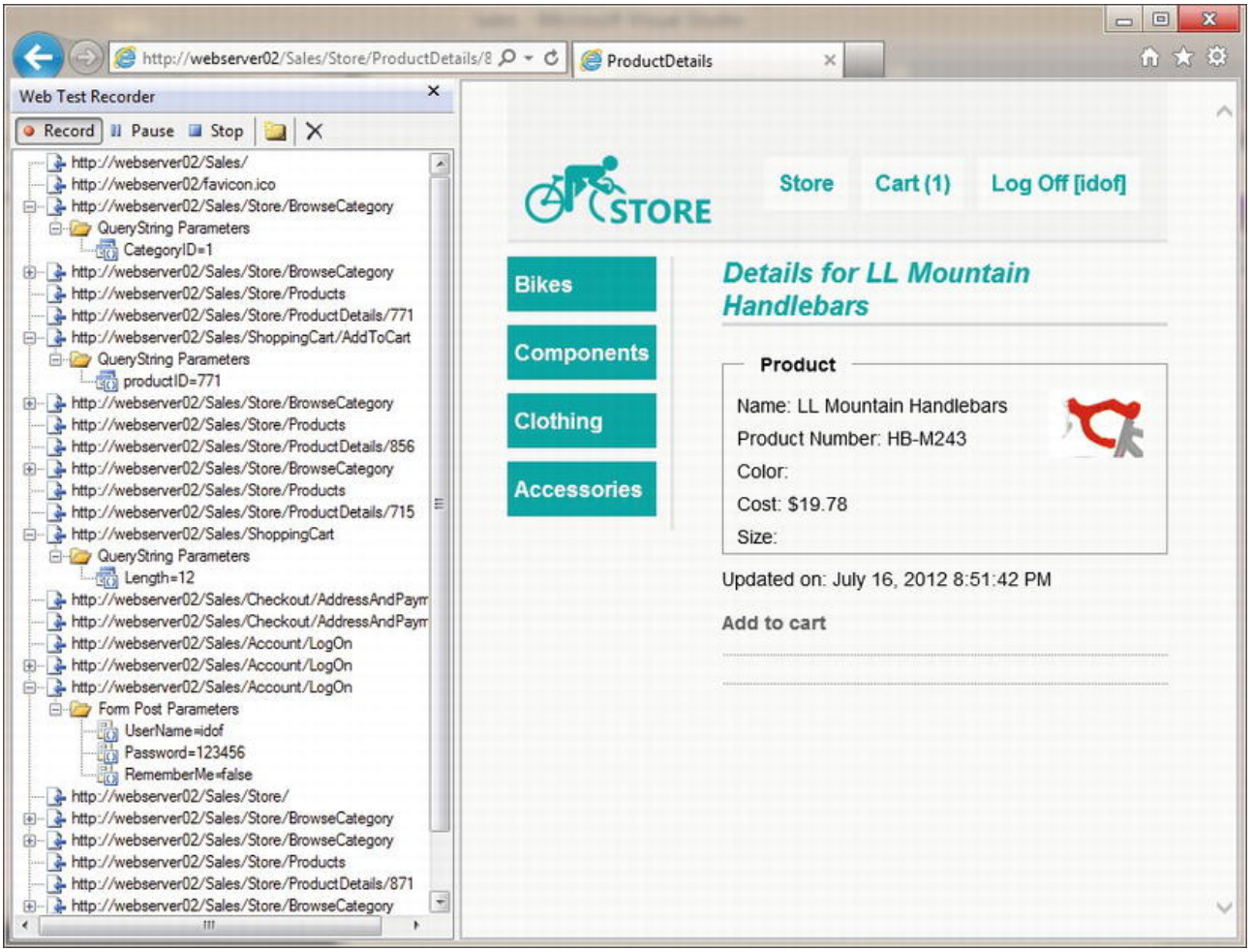

Figure 11-1. Recording a web application with Web Test Recorder

Once a recording has taken place, you can use that recording to test the performance of the web application, as well as test its correctness, by matching the new responses with the previously recorded responses.

Web Performance tests allow you to customize the flow of the test. You can change the order of requests and add new requests of your own, insert loops and conditions to the flow, change the headers and content of requests, add validation rules for responses, and even customize the entire test flow by converting it to code and editing the generated code.

Using the Web Performance test on its own has its benefits, but in order to test the performance of your web application under stress, use the Web Performance test in conjunction with Visual Studio's Load Test feature. This Visual Studio feature enables you to simulate loads on the system where multiple users are calling it concurrently, performing different operations on it, and to test how the system behaves during that time, by collecting various performance information such as performance counters and event logs.

Caution It is advisable that you do not load test a public web site, only your own web sites and web applications. Load testing a public

web site may be interpreted as a denial-of-service (DOS) attack, causing your machine or even your local network to be banned from accessing the web site.

Combining Load Test with the recordings of the Web Performance test, we can simulate dozens and even hundreds of users, concurrently accessing our web application, simulating calls to various pages with different parameters in each request.

To properly simulate hundreds of users, it is advisable that you use *test agents*. Test agents are machines that receive testing instructions from a controller machine, perform the required tests, and send the results back to the controller. The use of test agents helps in reducing the stress on the testing machine (not the machine being tested), because a single machine simulating hundreds of users may suffer from performance degradation, causing the test to produce faulty results.

During load test we can monitor various performance counters that can point out how our application behaves under stress, for example by checking if requests are queued in ASP.NET, if the duration of a request is increasing over time, and if requests are getting timed out because of faulty configuration.

By running load tests with various scenarios, such as different number of concurrent users or different types of networks (slow/fast), we can learn a lot on how our web application works under stress, and from the recorded data derive conclusions on ways we can improve its overall performance.

HTTP Monitoring Tools

Network monitoring tools that can sniff HTTP communication, such as Wireshark, NetMon, HTTP Analyzer, and Fiddler, can assist us in identifying issues with the HTTP requests and responses send to and from our web application. With the help of monitoring tools, we can verify various issues which can affect the performance of our web application. For example:

- **Properly using the browser's cache**. By looking at the HTTP traffic, we can identify which responses are returned without caching headers, and whether requests are sent with the proper "match" headers when the requested content has already been cached.
- **Number and size of messages**. Monitoring tools show each request and response, the time it took for each message to be received, and the size of each message, allowing you to track down requests that are sent too often, large requests and responses, and requests that are taking too long to process.
- **Applying compression**. By viewing the requests and responses, you can verify that requests are being sent with the *Accept-Encoding* header to enable GZip compression, and that your web server returns a compressed response accordingly.
- **Synchronized communication**. Some HTTP monitoring tools can show a timeline of the requests and which process generated which request, so we can verify whether our client application is able to send multiple requests at once, or are requests being synchronized due to lack of outgoing connections. For example, you can use this feature to detect how many concurrent connections a browser can open to a specific server, or check if your .NET client application is using the default two connections restriction enforced by `System.Net.ServicePointManager`.

Some tools, such as Fiddler, can also export the recorded traffic as a Visual Studio Web Performance test, so you can use Web Test and Load Test to test web applications that are called from client applications and browsers other than Internet Explorer. For example, you can monitor HTTP-based WCF calls from a .NET client application, export them as a Web Test, and use the Load Test to stress test your WCF service.

Web Analyzing Tools

Another set of tools that can be used to identify issues with web applications are web analysis tools, such as Yahoo!'s YSlow and Google's Page Speed. Web analyzing tools do more than just analyze the traffic itself, looking for missing caching headers and non-compressed responses. They analyze the HTML page itself to detect problems that can affect the performance of loading and rendering pages, such as:

- Large HTML structures that can affect the rendering time.
- HTML, CSS, and JavaScript content that can be shrunk in size by using minifications techniques.
- Large images that can be scaled down to reduce their size and match their dimensions in the HTML page.
- JavaScript code that can be executed after the page is loaded instead of during, to allow pages to load faster.