WHITE PAPER

Performance Analysis for Web Services

Understanding application performance in the era of Service-Oriented Architectures

The growth of applications using the .NET platform has generated an increased emphasis on performance measurement and analysis. Distributed applications, while much more flexible and potentially more scalable than monolithic ones, have characteristics that make it more difficult to achieve the goals of high performance and scalability. In such applications, performance issues may not manifest themselves in the individual components, but rather in the integrated application.

The problem arises both in the individual components within the Service-Oriented Architecture (SOA) and in their interactions with one another. Individual components, such as the application client or the Web service, may include computationally expensive code and bottlenecks that don't manifest themselves during unit testing, because the functionality is correct. And if they're not tested together, these issues may live on until the application is in production. Once separately developed components are integrated into the full application, performance bottlenecks may result from interactions between them.

These problems are especially common in distributed applications employing an SOA. In the case of traditional distributed components utilizing COM or CORBA, processing tended to be more synchronous, or at least more tightly coupled, which in turn can result in performance more in line with that of the individual components. In the case of asynchronous and loosely coupled Web services, there could well be significant differences in the ability of the SOA to provide the performance and scalability required by the application. This is true whether the front-end is a web page or a WinForms-rich client.

Developers who have worked primarily on stand-alone applications or tightly coupled clients may lack direct experience in the performance considerations needed by web-based distributed applications using services as a part of an SOA. In fact, many application developers may be surprised at the need to analyze performance and tune individual components, including Web services in a .NET application.

This doesn't represent a deficiency or limitation of .NET, but rather the reality of the trade-offs required to obtain the flexibility inherent in an SOA. Application developers simply assume that the Web services comprising the SOA will meet the performance needs of the client application, and typically measure performance only from the standpoint of that client.

In reality, individual Web services may themselves have performance problems or other limitations that prevent them from providing a rapid response to client applications. Even if the Web service has been fully tuned, interaction with other Web services within the context of the SOA, and with multiple client applications, may reveal problems that were hidden when testing the individual service.

Web services adapt the traditional web programming model for use from all sorts of applications, not just browser-based ones. These applications are loosely coupled and remarkably interoperable because they can be called from any location that is reachable with a URL or URI, and are not limited by the calling conventions of a specific language. So performance problems within collections of Web services may extend beyond web applications to include any application that makes use of the SOA.

Microsoft positions ASP.NET as a natural technology for implementing Web services based on the Microsoft .NET Framework. The ASP.NET Web services infrastructure provides an API for Web services based on mapping SOAP messages to method invocations. ASP.NET Web services support requests from clients using SOAP over HTTP, as well as with HTTP GET or POST operations. ASP.NET Web services automatically generate WSDL

files for Web services development efforts. Developers can also use ASP.NET Web services to implement a Web service SOAP listener that waits for service requests and accesses a business facade implemented as a COM component or as a managed code class.

Even though Web services using the .NET platform are straightforward to implement, they haven't been proven to stand up to the performance requirements of a wide variety of production uses. That's not to say that they won't or can't, but rather that this type of application model represents a significant unknown in practice. And clearly, this is critical from the standpoint of designing and implementing an enterprise SOA. Understanding the performance strengths and limitations of the Web services model in general, and of individual Web services implementations in particular, is necessary in order to rely upon them as the backbone of an enterprise architecture.

Where to measure performance

There are two aspects to measuring and evaluating the performance of an application composed of one or more Web services. The first is the throughput of the Web service itself—its ability to accept a request and provide a response in accordance with required performance parameters. On a larger scale, it's also the ability to actually have a transaction throughput that the component was designed to meet. We know that as "scalability," although it's difficult to test true scalability prior to integrating all application components and functionality.

The second aspect is the ability to evaluate the performance of the application in the aggregate, including the Web service. This includes not only the ability of the Web service to respond, but also how that response is coordinated with the responsiveness of the application as a whole. While the focus may be specifically on the performance of the Web service component, it must be analyzed within the context of the entire application.

Why? First, because that's the type of performance the end-user will experience. While the profiling data collected from developer tests doesn't easily correlate to the end-user experience, it can be representative of how the application will be used in practice. Second, bottlenecks may be exposed that may not be apparent from looking at the Web service separate from the client application. While they are loosely coupled, there can still be dependencies that affect overall performance, especially if the application is working with multiple Web services within the SOA.

To accomplish these goals, it's necessary to measure the performance of all of the application components simultaneously, during the same testing run, and correlate those disparate measurements into an integrated view. While the Web service may appear to perform acceptably within its own context, resource or processing issues, synchronization problems, networking or data throughput, and bottlenecks may prevent it from reaching its potential.

Unit testing during development

Unit testing a Web service presents the first significant challenge. As a practical matter, it requires an external stimulus to initiate execution, so it isn't as simple to call the service, as you would a DLL, or link it in, as you would a library.

Fortunately, using the Web services wizard in Visual Studio .NET has the side effect of creating a web page front-end for functional testing purposes, and it works well enough to initiate unit testing also. Otherwise, you would have to write your own call method into the service.

Unit testing should cover both functional and performance testing. Functional testing involves exercising the operations which compose the service, to ensure that they behave as specified. Whether you employ an automated test management system or conduct these unit tests manually, keeping track of code coverage is important to determine what code you've tested and how much you've tested it. Many developers already do this, and while Web services open certain challenges, the process is largely familiar.

The goal of performance testing is to identify slow code and potential bottlenecks. For developers with experience in monolithic or tightly coupled applications, this is often simply a matter of noting that the application doesn't perform as expected, isolating the component responsible and fine-tuning the code. It's not quite like that when building a loosely coupled application with Web services.

Profiling a .NET Web service, however, is a necessary part of the development process, even if you're just a consumer of an existing service. Since you're not looking at performance or scalability testing of the entire application at this point, you can initiate profiling early in the development cycle.

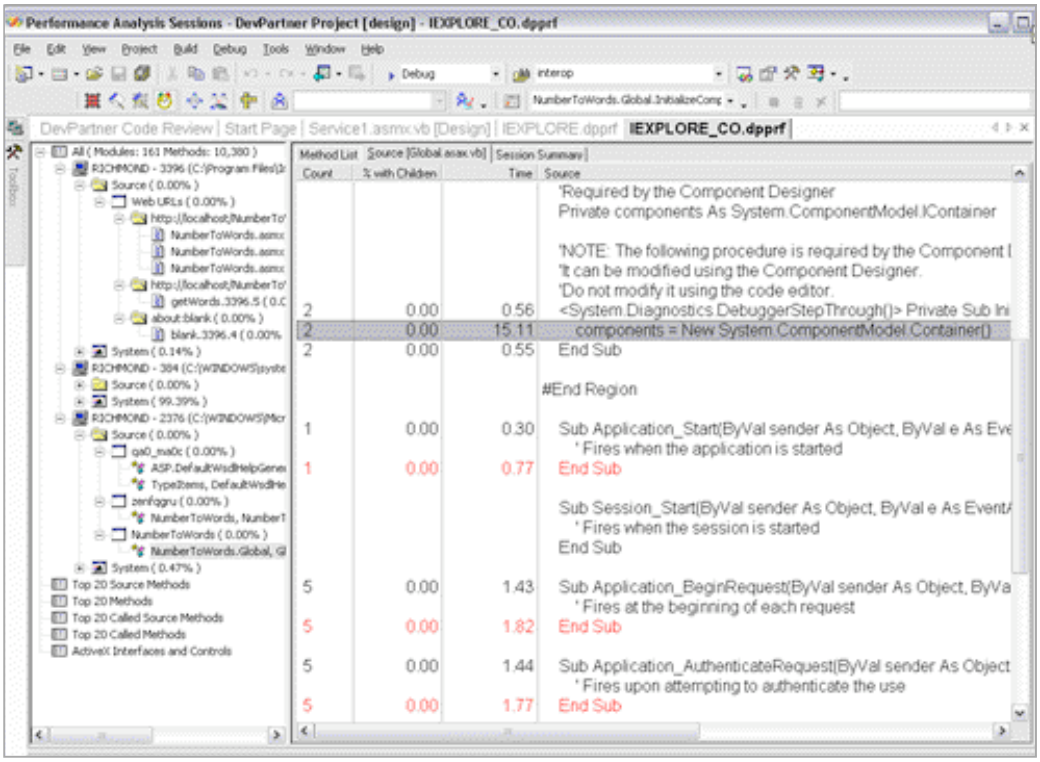

Figure 1. Compuware DevPartner Studio performance analysis lets you see the execution times associated with each line of code, as well as the number of times that line was executed.

At a minimum, profiling should collect two types of information: execution time and the number of times code executes (see Figure 1).

The reason for the first is readily apparent—so that you can quickly identify parts of the code that seemingly execute more slowly than others. It's not necessarily indicative of poorly performing code because it may just be performing a computationally intensive operation that can't be improved upon. However, information on the performance of your code could justify a closer look at specific operations.

The number of times code is executed can also be indicative of poor performance, but in a different sense. A single line or method may execute in an acceptable amount of time, but may be inefficient in the sense that a single call to that operation does too little work for too much overhead. The trick is to find the optimum amount of data that can be processed most efficiently.

Analyzing performance and diagnosing issues

Profiling a .NET Web service frequently requires looking at both managed and unmanaged code simultaneously because all but the most trivial calls into native code must undergo a mode transition. The mode transition, which is the physical process of moving data between the managed and unmanaged modes of operation, typically requires about two-dozen instructions. The second cost is marshalling the data to move across the boundary. Marshalling is computationally expensive, and the more data you move back and forth, the more expensive it becomes. Tests have indicated that marshalling complex data can involve up to several thousand instructions.

Alternatively, within the Microsoft model, Web services can also be unmanaged. Existing or new COM components can be used to implement the business facade, business logic and data access layers. Using Windows Server 2003, developers and system administrators can allow existing COM+ applications to be called using XML/SOAP by simply checking a configuration box. COM components also can be used as a part of a managed .NET Web service.

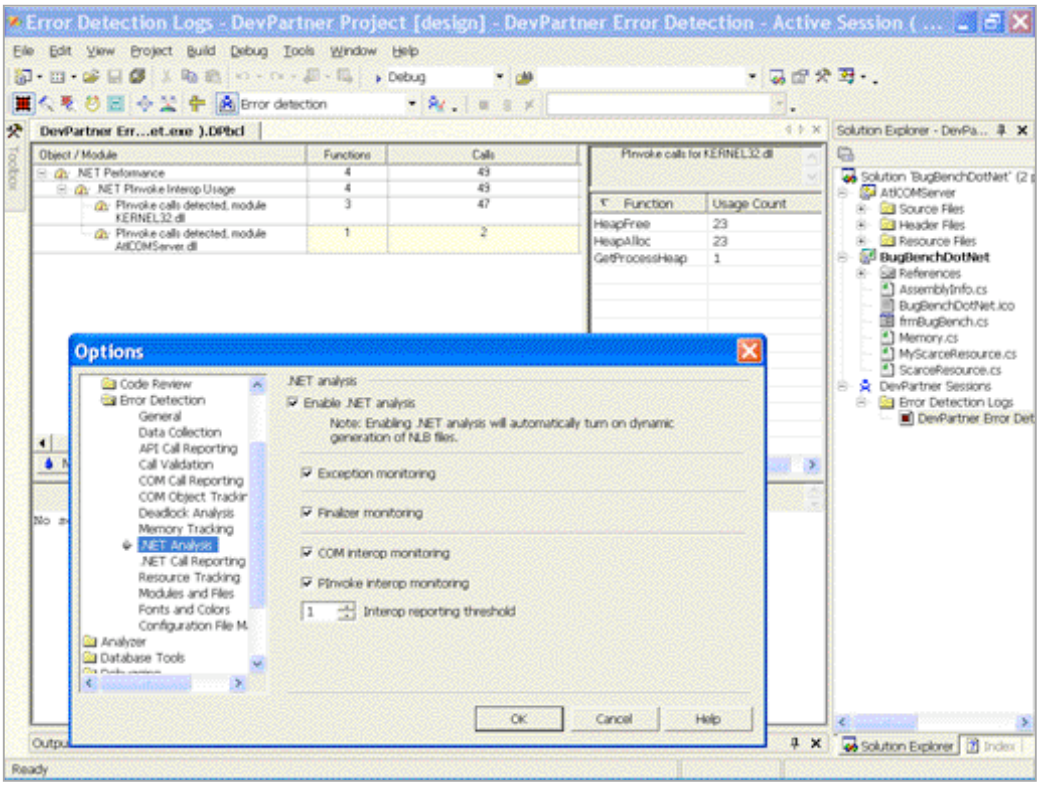

Figure 2. Compuware DevPartner Studio automatic error detection counts the number of transitions between managed and unmanaged code.

Therefore, you may be making unmanaged calls from your .NET Web service, whether you are using it as a gateway to call COM objects, making platform calls or using prepackaged native components. Profiling both managed and unmanaged calls together gives you a comprehensive view of the Web service, and most important, a view of the performance cost of mode transitions. You should also look at the number of times you're crossing the managed/unmanaged boundary. Because this process is so computationally expensive, you should seek to reduce the number of times it occurs (see Figure 2).

The granularity of your profiling should be at least to the method level, and preferably to the individual line of code. At the method level, you can quickly get a view of what operations your code spends most of its time performing.

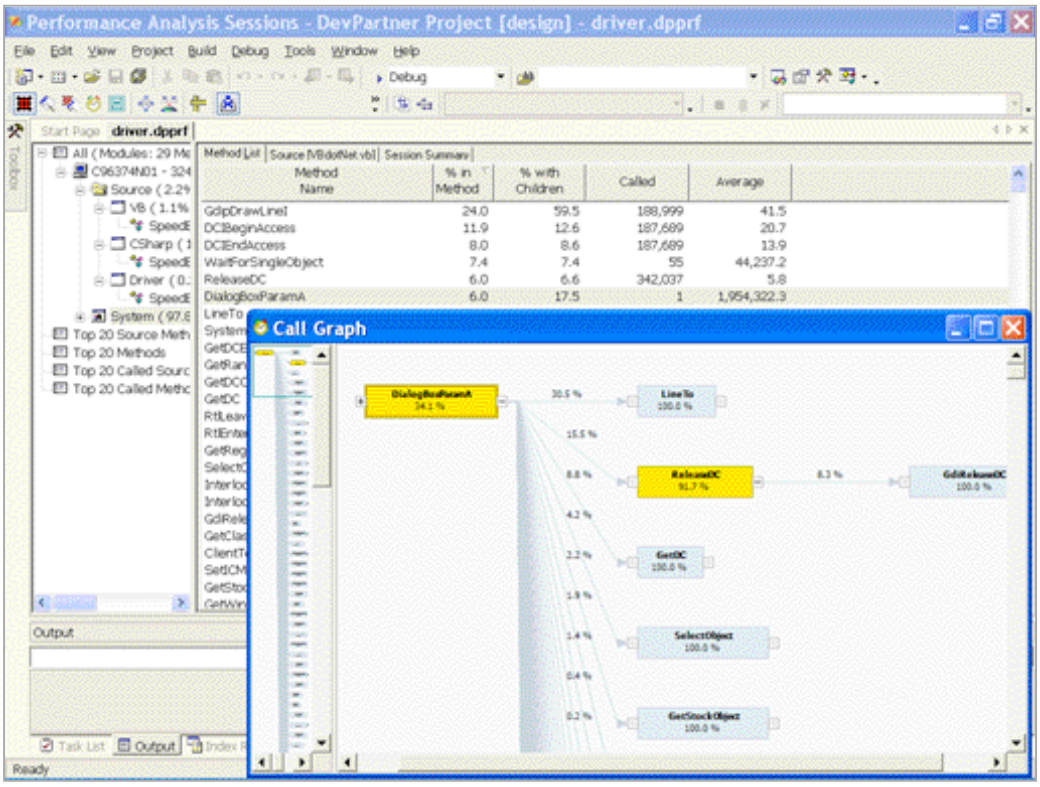

Figure 3. Compuware DevPartner Studio performance analysis provides a call graph that lets you visually track performance bottlenecks back to the offending method.

In addition to obtaining the execution time and number of calls, you should also be able to analyze your code in several different views of performance, including percent of time in both method and children. Walking the call stack is an excellent way of determining what resources, .NET services and OS services your Web service uses, and how expensive those services are (see Figure 3).

Modifying Web service performance

Once you have identified slow code, the next step is to address those issues. One popular technique for addressing how data is passed between the application and the Web service is to carefully monitor and optimize the amount of data transmitted in each SOAP call.

In this circumstance, one change you can consider is whether your code should be "chatty" or "chunky." As the names imply, chatty calls are those that occur often and pass little data, while chunky calls occur less frequently but do more work when they do occur. While at first glance it might appear that large calls to the service are more efficient, that's not necessarily true. A chatty interface passing less complex data more often may turn out to be less computationally expensive because its marshalling isn't as complex.

How do you determine whether or not you should use chatty or chunky calls to your service? There's no easy way that applies to all circumstances; it depends on the amount of data and frequency of calls. The best action is to prototype both the type of interface and the performance profile. By investing a little time early in the development phase, you can ensure you made the right performance choice and do not have to go back and make substantial changes after you deliver a working application.

That's not to say that you might not have to go back and make adjustments to your calls once the application is done. For example, you may find that in certain Web services features, chunky calls are more efficient because of the overall volume of calls. Prototyping your data and calls ahead of time, however, gives you better ability to determine the optimum amount of data to exchange with your Web services in individual transactions.

Another common problem is that certain .NET services can be computationally expensive. At first glance, it may appear you have little control over what the .NET Framework does, however, the framework is rich in features and there are often multiple ways of obtaining the services you need. Alternatively, the framework calls

you use may do more work than you actually need. You won't know any of this unless you have complete profiling information on child behavior from your own methods.

Taking the uncertainty out of SOA performance

Web services represent the potential to reuse code components as features across multiple applications simultaneously. The loosely coupled model and XML/SOAP communications standard is simple to understand and implement, but the performance implications are not yet well understood. In particular, it's unclear how a Web service used simultaneously by multiple applications will respond to such asynchronous requests.

This uncertainty is magnified when an SOA is considered. Multiple interacting Web services, with one or more servicing multiple client applications, may have performance issues that simply can't be detected while developing individual components in isolation. You have to look at the SOA and its applications as a complete system, rather than a set of parts.

You're going to be feeling your way in building and testing the performance of Web services. That makes it important to profile these services, both by themselves and within the context of the entire application or applications, to determine where slow code and bottlenecks may reside. In doing so, you can address those issues by changing the calls to or within the Web service, modifying expensive database calls, simplifying complex code paths or using other techniques.

To learn more about Compuware DevPartner, visit www.compuware.com/products/devpartner/default.htm

Compuware products and professional services—delivering quality applications

Compuware is a leading global provider of software products and professional services which IT organizations use to develop, integrate, test and manage the performance of the applications that drive their businesses.

Our software products help optimize every step in the application life cycle—from defining requirements to supporting production service levels—for web, distributed and mainframe platforms.

Our services professionals work at customer sites around the world, sharing their real-world perspective and experience to deliver an integrated, reliable solution.

Please contact us to learn more about how our comprehensive products and services can help your organization improve productivity, create higher quality applications and ensure performance in production.

Compuware Corporation Corporate Headquarters
One Campus Martius
Detroit, MI 48226

For regional and international office contacts, please visit our web site at www.compuware.com

COMPUWARE®
www.compuware.com