Realtime publishers

The Essentials Series: Improving Application Performance Troubleshooting

Why Diagnosing Application Problems is Too Hard

sponsored by

BLUESTRIPE SOFTWARE

by Don Jones

Copyright Statement

Why Diagnosing Application Problems Is Too Hard

Anyone working in any kind of business is familiar with this scenario: Users complaining that their applications have slowed down, managers getting tense that their employees aren't working, and the IT team running around trying to pinpoint the cause of the problem. In this *Essentials Series*, I'll share some of my own experiences in troubleshooting slow applications in a major enterprise, and try to unearth some of the reasons that troubleshooting seems to always be so slow and inefficient.

But let's be clear about something: I'm not discussing *application performance management.* APM is a huge IT discipline in and of itself, typically involving large, complex tool sets and very specialized skills. APM often involves measuring performance in somewhat unusual ways, such as running an agent on end-user systems as well as on back-end systems to measure the response times between the two. APM is certainly useful, but what's equally useful is being able to *do something* about applications that are experiencing some kind of performance setback. And perhaps more importantly, being able to do something *quickly.*

"Quickly" is incredibly important. In a 2008 study entitled *Service-Level Management and Application Performance Management,* Forrester Consulting found that only two-thirds of application performance issues were resolved in what the business considered an acceptable time period.

It Starts with a Help Desk Call

Let's start by setting the stage, and looking at the first ways in which the troubleshooting process begins to bend toward massive inefficiency. Normally, you'll hear about a slow application directly from its users, often through your company's Help desk. If you're a fan of IT Service Management (ITSM) practices, this notification becomes an *incident;* in more generic terms, you might just call it a *trouble ticket.*

But these incidents aren't as cut-and-dry as "my computer blue screened" or "the printer isn't working;" users are seldom able to be more specific than "my application is suddenly running really slowly," or "my application stopped responding."

Ideally, your first-tier support folks will try to verify the problem, and may be able to perform preliminary troubleshooting steps: Does the problem affect just a single user or are multiple users impacted? Does rebooting the user's client computer or closing and re-opening the application help resolve the problem? Are other applications on the same computer affected? Is the problem limited to a single remote office or to a single department within the company?

Sadly, that's usually where the first-tier team runs out of options. Unless an application or computer restart solves the problem, all the first tier can usually do is verify that the problem is occurring and perhaps try to define the scope of the problem. Because applications tend to be both complex and, in most cases, not specifically designed for performance troubleshooting, the first tier often has no choice but to toss the problem up the chain of command, and that's where the first major inefficiency kicks in: Who, exactly, should get notified of the problem?

And Quickly Involves the Entire Team

Your first-tier support might initially notify a software developer, but the problem almost always escalates from there. Think about all the technology disciplines involved in a modern application: client application developers, component developers, database administrators, database developers, and more. In addition, applications rely on many different shared infrastructure elements, such as the network, domain name services, routers and switches, and more. So before long, an entire army of high-end technical experts are examining the problem: developers, database administrators, network administrators, infrastructure administrators, and more.

The More, the Merrier?

I recall one troublesome application we had when I was working for a large telecommunications company in the US. One day, everyone seemed to think the application was running a bit slow, but we couldn't really pin down anything. By the end of the next week, the application was positively *crawling,* and managers throughout the company were screaming for support. Our Help desk was at a complete loss: They'd actually gone above and beyond their normal responsibilities and scheduled server reboots, router reboots, and more, but nothing they did seemed to be helping. So they finally kicked the problem upstairs.

We could have thrown a pretty good party with the number of people who became involved at that point: I was in charge of the network infrastructure, although I had specialists in charge of much of the infrastructure equipment, like our routers and switches. One of the software developers started looking at the application itself, and another started looking at its various sub-components. Our Unix admin started examining the Unix-based firewall and DNS services. A database administrator started poring over the SQL Server, while another developer started poking around the server-side code within the database. One of our desktop support specialists started looking at the client computers to see if maybe there was some kind of resource conflict there. Our managers, of course, hovered over everything and got in the way. Nine or ten people, taken away from extremely important projects, all to fight a single fire.

Look, Guess, Fix, Repeat

Everyone on the team starts looking at the slow application with their own particular tools:

- Network administrators might grab a packet analyzer and start looking at network traffic.
- Database administrators may turn to a profiling tool to look at database server performance.
- Software developers often have dedicated code performance tools (read more about those in *The Definitive Guide to Building Code Quality* from Realtime Publishers) that can profile application code performance.
- Desktop technicians may run client OS performance tools, such as Windows' System Monitor or Performance Monitor.
- Infrastructure engineers might look at statistics collected within routers and switches.
- Technology managers usually don't have any specific tools but try to make lots of "suggestions" to everyone else anyway.

Pay special attention to that list because you're looking at the exact reason why troubleshooting a slow application is so inefficient. I'll get to the specifics in a bit, but let's stick with the chronology and talk about what happens next: Someone takes a guess.

Maybe the software developer has a tool, like the one shown in Figure 1, that breaks down application performance within the application's code.

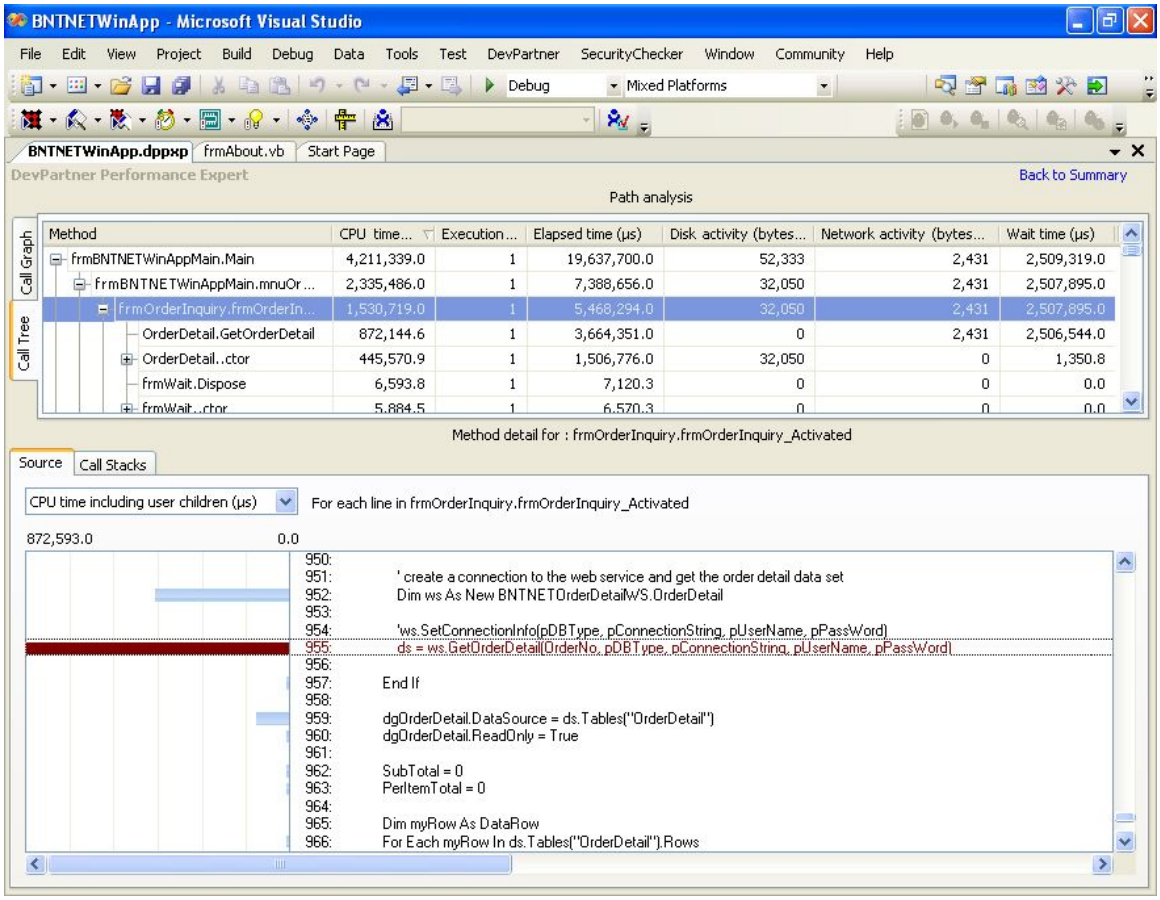

Figure 1: Analyzing application code performance.

The developer says, "aha, this frmOrderInquiry method is taking a long time and generating a lot of network activity—this must be it." The developer makes some tweaks to the code, runs it through his performance tool again, and sees an improvement. "I've found the problem!" he announces to the team. Everyone sighs in relief, the developer compiles a patch, and it's deployed to a few client computers (usually the ones with the users who've been complaining the loudest).

And then the bad news comes in: It didn't fix it. In my experience, the first guess at *why* the application is running slowly is wrong about *40% of the time.* So the team groans, drops what they're doing, and jumps back on the problem. They drag out their tools and start looking at their own little domains until someone comes up with a better guess.

Modern Applications Are Complex Systems…

The root cause of this troubleshooting inefficiency is that modern applications are really, really complicated. Even seemingly-simple applications can, in reality, be incredibly complex. Think about the average custom, in-house application:

- It's built on some kind of framework—perhaps .NET, perhaps Java, perhaps something else—and that framework will have its own internal performance characteristics.
- It's usually not a monolithic application but rather a collection of components. Some of those may be purchased and some may be developed in-house. Some may even be open source, downloaded from the Internet. Each component may have its own performance issues.
- Applications run on a computer, which may well have its own performance problems: lack of memory, slow disks, failing network card, and so forth.
- Applications rely on a network that consists of its own complex infrastructure: routers, switches, hubs, wiring (which is susceptible to electrical interference), and so on.
- The network provides services that applications rely on, including DNS, which applications may use as a precursor to communicating with network-based servers.
- Applications commonly communicate with one or more servers, which may have their own performance issues. Those servers run program code, too, which may feature any number of performance problems.
- Applications often utilize a database, which can have its own performance nightmares in addition to the performance problems of the server actually running the database software.

And aside from all these individual components, you've got the single complex entity called an *application,* which from a holistic viewpoint may exhibit what I call *synergistic performance problems.* In other words, sometimes a few well-oiled components can conspire to create poor overall performance. You can't point to any one component as the cause of slow performance, but taken together, the various components do, in fact, run slowly. Many of these components may only run slowly some of the time, which makes troubleshooting even *more* difficult—problems that can't be repeated consistently are much more difficult to solve.

And Application Downtime Costs Money

When applications are running slowly, users aren't productive. In one of my past jobs, we estimated that each hour of application downtime cost us $45,000 in lost productivity, diverted technical expertise, and more. On average, "my application is running slowly" would take 48 hours to finally resolve, costing the company $360,000. That's just an average, though—I remember one problem that took us over a month to finally sort out—a theoretical cost of millions of dollars. Ouch.

> **Note**
>
> In their study, Forrester found that 43% of businesses estimated downtime costing at least $10,000 per hour, and 10% cited more than $1 million. Research from Enterprise Management Associates suggests that an industry average of $45,000 per hour is accurate, although they say it is higher for transaction-intensive verticals such as financial services, where the cost can exceed millions per minute.

The point is that *time is money*, and you can't afford to keep doing the "look, guess, fix, repeat" cycle. You need to be able to look and to fix—definitively, without the guessing and repeating. So why is it so difficult?

The Problem Is in Your Tools

The problem is that first bullet list I showed you: All those IT experts using their *individual domain tools.* None of them was troubleshooting *the application:* They were troubleshooting individual bits of it. Sure, eventually one of them will stumble across the answer, but it's horribly inefficient.

When you hear a loud, annoying sound in your house, you don't immediately call in a plumber, HVAC technician, structural engineer, and an exterminator. You take a few minutes to try to narrow down the problem. You listen to the house as a whole, turn the taps on and off, shut the HVAC down, and so forth. When you've narrowed the problem a bit, you call the one specialist that can handle that problem. That's how you *should* be able to deal with a slow application: The Help desk—your first tier of support for the incident—should be able to direct the problem to the *one* technology expert who can actually fix it.