Realtime publishers

The Essentials Series: Improving Application Performance Troubleshooting

# Faster, More Efficient Application Troubleshooting

*sponsored by*

BlueStripe Software

by Don Jones

## Copyright Statement

# Faster, More Efficient Application Troubleshooting

So how do we resolve application performance problems more quickly? How can we ensure that "my application is running slowly" isn't met with a barrage of technical expertise that doesn't actually yield results very quickly?

The answer is *not* to buy more domain-specific tools. Most organizations already have plenty of tools within each technical "silo"—network analyzers, database profilers, and so forth. They're not helping because each tool only looks at one potential part of the problem. Also, those "silo tools" don't come into play until *after* your first tier of technical support has tossed the problem up the chain of command. What we really want is a way to make the first tier of technical support more effective because that will also make subsequent tiers more effective and efficient.

And I know what you're thinking: "Our Help desk folks are smart but not that experienced—that's why they're on the Help desk!" We often think that *making the Help desk more effective* means *giving the Help desk more tools and teaching them how to use them.* That's an education approach, and sometimes that's not practical. First, your first tier of support may well lack the background experience to use those tools. Second, you'll just be giving them the *same silo-specific tools you were already using.* Moving the tools to a lower tier of support isn't the answer, either. So what *is* the answer?

## Enable the Help Desk

The answer is to *enable* the Help desk, or whatever you call your first tier of support. Not necessarily *educate* them and not try to make them as smart as your second- or third-tier experts, but to *enable* the Help desk. Specifically, enable the Help desk to do a better job of verifying slow applications and routing the incident to the correct portion of your second or third tier.

What we need is a tool that can pinpoint exactly which bit of a complex application is causing or contributing to a slowdown. That lets them refer the problem to the experts responsible for that bit: No more looking, guessing, fixing, and repeating. The tool needs to look at the *entire application,* including all of its dependencies, back-end components, and other elements. That tool needs to understand the shape of the application and how the various components fit together, and it needs to be able to view their performance characteristics in real time so that it can spot those temporary slowdowns that bother users and frustrate technical experts.

## Enable a Focused, System-Wide Approach

We IT experts have to stop thinking of our applications as a collection of building blocks, and instead think of them as a single entity composed of many different contributing elements all working in tandem. A building engineer doesn't think of girders, concrete, and windows; he thinks about the *building.* Sure, girders are a part of it, but once welded and riveted together, those girders behave differently than they did as individual units. If the building's exterior starts to show cracks, the engineer has to consider not only the girders but also how they've been attached, how they're affected by the load of the building's walls and contents, and so forth, all as a single, interconnected set of elements. He—that *one* engineer—has to consider *all* the elements that make up the building, even if he isn't personally an expert in, say, windows.

That's where we need to take our application troubleshooting. *Everyone,* from the Help desk on up, needs to be able to see the *entire* application not just their individual silos. Everyone who troubleshoots a slow application needs to be aware of every element of that application, how those elements interact, how they fit together, and how they might—as a unit—be exhibiting performance problems that aren't immediately apparent in any individual component taken by itself.

Here's another non-computer example: Suppose you have a car that's getting really poor gas mileage. You disassemble the car and start testing each component—pretty much exactly how we troubleshoot slow applications today. The engine seems to be running well, the drive train is in good shape, and the tires look to be properly inflated. So where's the problem? Well, maybe you take a guess and rebuild the engine. You reassemble the car and the problem is still there.

Looking at individual elements of a system will almost never reveal the cause of a performance problem because those elements work differently on their own than they do as part of the system. To find the problem with the car, you need to examine those components *when assembled as a car,* not individually. You need to put little sensors and stuff on each component, and monitor them as the entire car runs. You might find out that the car's chassis is too heavy and that places stress on the drive train, which makes the engine work harder—something you'd never see if you examined individual components by themselves.

To start troubleshooting our applications as *systems*, rather than as individual components, we need to focus less on our silo-specific tools such as database profilers and network analyzers. Instead, we need to find tools that can look at the entire application—every element, every component—and all in real time.

## Find Tools for the Application, Not for Silos

The idea of looking at an entire application's performance is hardly novel. One of my first IT jobs was as an AS/400 system operator, and we had a tool that could, in real time, analyze each aspect of a running process' performance. It would tell us if a slow process was slow because of memory, processor time, database responsiveness, and so on. Of course, an AS/400 is a completely self-contained system, so our troubleshooting tool wasn't terribly complicated.

Modern tools exist that extend the same concept to today's complex, multi-tier, distributed applications. They understand—or can be configured to understand—that the application may start with a single process running on someone's client computer, but that the "application" also consists of an underlying network, a back-end database, numerous other processes and components, and so forth. These tools can check the performance of each element in real time and can do so continuously, actually alerting you in advance when an application element's performance strays outside normal parameters.

These tools enable a holistic, system-wide look at the application. They can make it easy to see which bit of the application isn't performing properly, and they do that while looking at the *entire* application.

## A Wish List for Application Management and Analysis

There's a problem with some whole-application monitoring and troubleshooting tools, though: installation and configuration. If the tool requires you to manually explain to it how the various application components fit together, you're less likely to use the tool. You're less likely to keep it updated as the application changes. You're *more* likely to make mistakes or miss components, giving the tool a lopsided, inaccurate view of the entire system. So with that in mind, I'll propose a sort of "wish list" for what a good, system-wide application troubleshooting tool might be able to do.

First on the list is *automatic.* Ideally, a tool should take a few minutes to install, and you should be able to point it at a running application. It should then take over, figuring out *on its own* what the application looks like and what dependencies that application has. The result is an *application map,* and it might offer views like the one in Figure 1.

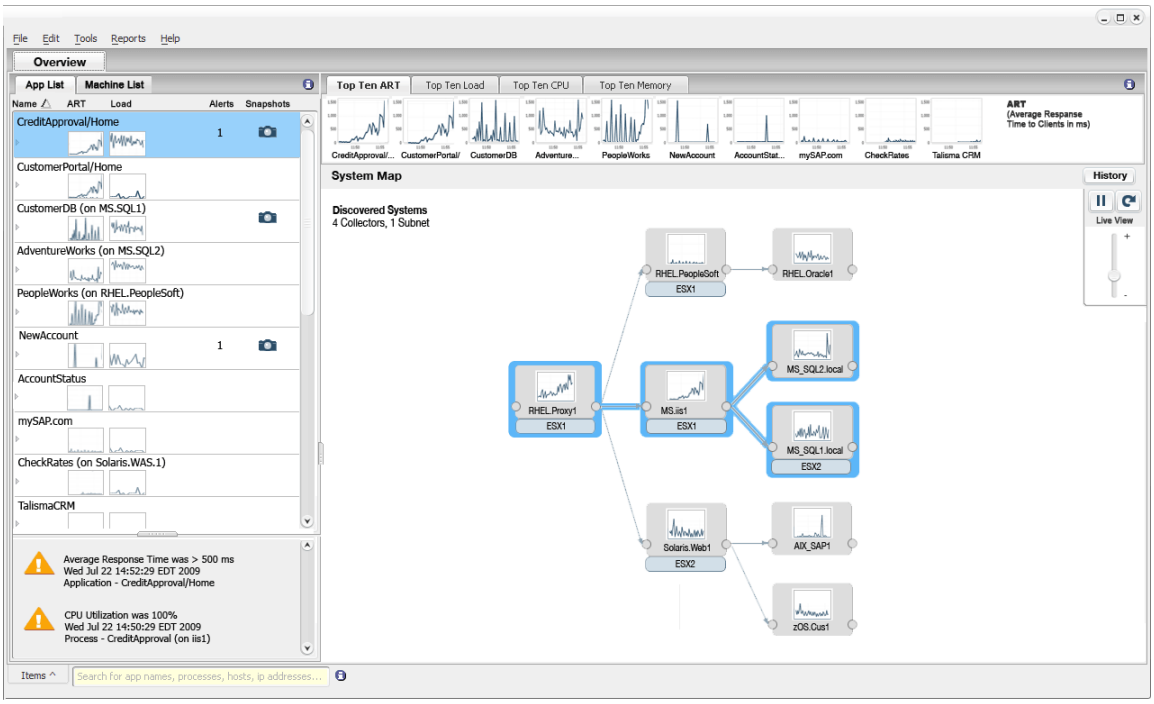


**Figure 1: Looking at the entire application.**

This view offers a list of applications on the left, listed by name. When reports of slow applications start coming in, the Help desk can select that application from the list, and look at the performance of each component that makes up that application.

That leads to the second item on the list: *intuitive*. This tool has to be usable by the Help desk, preferably with as little education as possible. It should call the Help desk's attention directly to slow application components so that the Help desk can simply notify the higher-tier expert responsible for those slow components.

Third on the list is *real time.* The tool is useless if it can't display the *current* state of the application's components. When the Help desk opens this tool, it needs to be up to date and ready to go. Those little performance charts should be moving and changing in real time so that the Help desk can easily spot system elements that are performing outside normal parameters.

The result? We always used to feel it took us 80% of our time to *find* the problem, and only 20% to fix it. A tool like this can really cut that 80% back a lot because it eliminates the "look, guess, fix, repeat" cycle. You can focus on what is *actually* causing the application to slow down, and spend your time *fixing* it, not guessing.

The fourth item for my wish list is *smarts.* If a tool can do the first three things, there's no reason it can't proactively alert someone when a system element's performance *begins* to show signs of a problem.

That takes us from reactive application troubleshooting to more proactive application management—a wonderful thing that any organization will benefit from.