InfoWorld

SPECIAL REPORT

Cloud Monitoring

Deep Dive

New monitoring architectures tackle more complex applications

New perspectives on application monitoring

A new generation of monitoring systems addresses the challenges of large-scale applications and more dynamic infrastructures

By Peter Wayner

NOT LONG AGO, I went to a website, browsed a catalog, and put something in my cart. The checkout went well until I typed in my credit card number and then it failed. At first I blamed myself, but after trying a second card I started to blame the site. Something was broken between the credit card processor and the checkout application. I left the products in the virtual cart and went to another site to buy what I needed. For all I know, the goods are still sitting in that virtual cart.

Problems like these can be difficult to detect -- close to impossible. Somewhere deep in the stack a connection failed, and I bet the operations department didn't notice until the money stopped rolling in. The front page worked. The search function worked. The cart worked. But one crucial connection didn't.

Some problems can be even more hidden. The failed purchase at least offered a symptom that would raise alarm bells. An accountant would probably log into a bank account to double check the sales figures from the day before and see a big zero. Many bugs aren't so kind. They don't raise alarms and the users often have better things to do than write detailed bug reports. But after a month goes by, one section of the site fails and the complexity makes the failure nearly impossible to discover.

These problems are compounded as Web applications migrate to the cloud, where the infrastructure encourages fluid deployment. When machines appear and disappear as part of the strategy, it becomes harder to structure the test program to determine when to flag a disappearing machine. So many machines appear and disappear that a human can't construct a test schedule.

The good news is that Web application monitoring solutions are evolving to root out trouble even when the problems are not obvious. The solution providers are extending the traditional architectures with adaptive mechanisms that react to the responses from the site and then probe it further with tests based on the answers. The tests are growing more sophisticated and doing a better job of detecting hidden failures.

The monitoring solutions are also casting a wider net, supplementing their site testing with additional data. Some reports integrate data gathered from the Web server, application server, database server, and the network while others search out data from third-party partners that can reveal how the website is performing. These third-party sources serve as a reality check, much like the bank account reveals that the money isn't flowing. If this external data reveals problems, the monitoring service can deploy more tests that probe the website to explain why the numbers aren't adding up.

Understanding these tests is now easier because the monitoring software vendors are producing more effective dashboards and reports. The eye-glazing columns of numbers are long gone, replaced by sophisticated, color-coded graphics that highlight problems quickly. If the errors are serious enough, the monitoring tools can raise alarms through multiple channels.

This new generation of monitoring solutions is making it possible for website managers to keep a handle on the burgeoning complexity of larger, more dynamic, and more distributed Web applications. The Web team now has a fighting chance to notice the problems that can hide in the layers of features and interconnected components.

NEW ARCHITECTURES FOR DETECTING PROBLEMS

In the most abstract sense, the job of detecting a problem with a website hasn't really changed. The monitoring tools still watch the way that humans interact with the site and look for unexpected responses. In many cases, the monitoring software pretends to be a human by approaching the website with a typical request. In others, the software interacts directly with the website to monitor the statistics gathered internally about load, response time, and throughput.

The basic theory hasn't changed, but the systems have grown much more sophisticated. They not only monitor many more parts, but they're also smarter about interpreting the results and raising alarms, reducing the number of times they cry wolf. They can detect when an occasional slow result is acceptable and know when the value might indicate a greater failure is just around the corner. They can amplify the intelligence of the monitoring team by finding the most significant issues and raising the appropriate level of alarm, narrowing in on the problem so the humans don't need to comb through endless tables.

The systems rely on a growing number of information sources. These include:

DIRECT TESTS. The first tool for identifying flaws was asking for a copy of the website, and simple requests for data remain a big part of the strategy. All Web monitoring tools begin by requesting specific URLs and tracking the time it takes for the data to arrive. Fast is almost always good -- except when it suggests that the website isn't doing everything it's supposed to do.

PARAMETER-DRIVEN TESTS. Simply asking for a Web page doesn't work when the data changes from user to user or from device to device. One way to detect problems is to include mock data in the database and send fake requests, watching the responses to ensure the results correspond to the request. This can avoid the problem of concluding that a website is fine because it returned an answer quickly.

COMPONENT TESTS. Many websites now use secondary calls to fill out parts of the page with data after the initial shell of a page appears. If these secondary calls aren't simulated and tested, the page may appear to work even if large parts are left blank when the secondary data doesn't return. Monitoring these usually involves sending additional URLs and testing the results, which may be in formats such as JavaScript, JSON, XML, or even HTML.

BACK-END SERVICES. Some monitoring tools can home in on database problems by sending test queries directly to the database server. Others can log into the database and collect statistics about the load and the average time it takes the database to answer queries.

DELIVERY INFRASTRUCTURE. Some failures occur because the routers, the DNS servers, or other network services fall down on the job. Some site monitoring tools include key network services in the mix, while others can at least flush their caches to ensure that they're executing a new test of the DNS services.

THIRD-PARTY SERVICES. Some solutions gather performance information from third-party services such as Google Analytics. Third parties might offer the same tests as the in-house monitoring service, but the services usually run the queries from an independent location relying on the general Internet. This can prevent blindness that occurs when a functioning website passes all of the internal tests but fails to deliver information to customers because a firewall or gateway is down.

NON-WEB SERVICES. Not all information travels through the browser. Monitoring tools can test email delivery to make sure that receipts and other notices are flowing. Some services use other protocols (such as FTP for file transfers), and the monitoring tools can test these as well.

OS-LEVEL TOOLS. Some monitoring services watch the load of the servers and check to see the transaction rates of virtual machines. JVMs, for instance, offer debugging tools that can also monitor problems with the service.

CLUSTER MONITORING TOOLS. Many of the big data tools choreograph the work of multiple machines, and the tools for organizing these jobs can provide information on the general health of the system. An overloaded Hadoop cluster may indicate that the larger system is overloaded and the customers may notice.

TEST STRUCTURE

When a test is completed, the monitoring tools begin determining whether the data indicates something good, something bad, or something that requires more checking. The tools rely on a variety of tests to arrive at these classifications:

BOOLEAN TESTS. The simplest question is whether there's an answer from the site. If there isn't, it often means a big problem.

TIME THRESHOLDS. Many sites set a goal of generating a response within a certain number of milliseconds. If the data takes too long, an alarm is generated.

CONSISTENCY. It is often possible to correlate the answers from two different tests and make sure they agree with each other. Correlated pairs aren't common, but they are valuable.

STATISTICAL THRESHOLDS. For many sites, it may not make sense to be too worried about occasional slow responses, but if the average response gets to be too long it means that everyone is having trouble. More sophisticated statistics can watch for occasions when the average or median response time remains steady, but a significant fraction of all responses takes too long.

CORRECT MOCK ANSWERS. It is common to load the database for personalized sites with fake names and profiles. The test packages can check to make sure that the responses have the correct name and address for John Doe, validating the connection between the database and the user.

INTERPRETING EVENTS

If the tests suggest something is amiss, the system will generate an event and start tracking how the event is handled. The earliest systems merely logged the events, forcing the supervisor to scroll through long lists. The modern systems have elaborate structures for classifying events and starting other tests in response:

TRIGGERS. Some events should trigger alerts immediately. If a test shows a serious problem, the event is pushed to the top of any dashboard and a combination of email alerts and text messages are sent out.

SCRIPTING. In some cases, it makes sense to step back and assess the depth of the problem. A script gives test engineers a way to program the system to work through a flowchart. It can schedule extra tests and react if these also raise questions. Scripting also allows you to build a more sophisticated alert schedule. A slow result may be reported only to the close engineering team, but a very slow result may generate alarm bells for everyone.

MACHINE LEARNING. Some of the most sophisticated systems deploy artificial intelligence algorithms to try to interpret multiple results. These are often experimental but they can detect confluences that escape the human eye. For instance, a very fast response time from one system might indicate a problem brewing in a completely different part of the site because the fast machines aren't being burdened by their usual workload due to the failure. A smart AI system can draw these kinds of correlations.

Machine learning algorithms can be very intelligent, but they usually require a human to interpret the results. They excel at noticing subtle patterns in big tables of numbers, but an admin must ultimately decide whether something should be done about a problem.

ADAPTIVE LOGGING. Many events are forgettable. Some are not. The best systems start tracking what happens until any trouble is resolved. The adaptive logging systems can also record statistics such as the load of the system while an event is active so the admins can make more intelligent decisions.

MULTIPLE PATH ALERTING. The teams taking care of a system don't need to know about all the events. The scripting systems can determine who learns of which events and who doesn't. The level of alarm will also change depending upon the severity. A problem with the database, for instance, may ring all of the alarms (beeper, SMS, email, phone) for the DBA but generate only an email message for a supervisor.

AUTOMATION

Automated discovery and configuration are two of the most important features for modern systems. As more and more websites move from a small collection of servers to a flexible cloud, architects are adding and subtracting servers so often that it's not possible for the testing team to reconfigure their testing strategy. Automated monitoring systems can identify new machines even before the test engineers know they exist.

The most sophisticated systems start recognizing new events when they arrive from the new servers. Some systems, for instance, keep track of all database queries and send statistics about them to the monitoring server. When new queries appear, the monitoring software flags them and begins tracking their behavior without being asked.

Noticing these new queries can help detect unexpected problems that lie outside the test plan. If an intruder fools a Web page into issuing a database query, for instance, automated detection software can flag this immediately. These sorts of anomalies were ignored by previous generations of monitoring tools because they weren't part of the preprogrammed script.

Here are some of the features that are becoming common in automated test environments:

EMBEDDED AGENTS. These tools are included in the software stack deployed to each server. They can be configured to report detailed statistics about response time and success rates for data.

SMART MONITORING TOOLS. When the agents report interactions with other servers, the monitoring software begins tracking these servers, too.

NETWORK MONITORING. Network-level tools can query routers, load balancers, and other infrastructure to reveal which machines are running.

DEPLOYMENT TOOLS. In some cases, the system can deploy new servers from the cloud automatically. This may happen if the load rises above a certain level or it may be triggered automatically at a certain time. This can save the admin the effort and also speed up the response.

THIRD-PARTY SERVICES. Some automated test environments supplement their analyses with data collected from third-party monitoring services.

NEW ARCHITECTURES FOR REPORTING

None of the test information does any good if the monitoring team doesn't notice the problem. The earliest monitoring tools would announce that a website was broken by triggering a beeper or sending a terse email. These basic techniques are still important parts of any solution, but they're rapidly being displaced by more sophisticated tools that offer deeper analytics and the capability of spotting problems before they occur. On top of the greater precision, today's monitoring tools provide customizable dashboards that report a range of statistics and flag the ones that might indicate trouble.

For instance, when a website begins to fail after being overwhelmed by a wave of traffic, it often doesn't go dark at once. The first burst of traffic may be handled smoothly because the visitors may be requesting static pages that are easily cached. The trouble may come later when these visitors move on to more computationally heavy tasks such as searches or purchases that interact with a third-party API. This load follows the initial burst and overwhelms the servers.

Good reporting tools can make these flows apparent and give engineers the chance to limit the damage that may occur. A graphical dashboard can easily reveal that the front-end caches are encountering much higher demand and raise an alarm before the visitors move on to more computationally intensive queries. This may be a signal to bring more servers online.

The challenge is usually packing all of the information into a simple interface that takes up a small amount of space. This mixes good visual design with a sophisticated back end that can react appropriately to alerts generated by the testing tools. The dashboard must classify all events in a way that the most critical problems are obvious: Good statistics in green, problems in red, and suspicious events in yellow. Some dashboards rotate charts and lights to present more information in the same amount of space.

Naturally, mobile devices present a special challenge to the dashboard designers. Most of the mobile tools rework the standard desktop dashboards for the small screen, a task that requires more than shrinking the page. Better analytical tools are essential for editing the information that the admins must see. These can work behind the scenes and provide the intelligence to raise the right alarms for anyone at home, stuck in a meeting, in transit, or, alas, even asleep.

The critical work is done by the algorithms operating beneath the dashboard. These are able to classify the incoming events and decide if the events are dangerous enough to raise an alert. The basic algorithms use static thresholds and create an alert if the statistics lie outside the acceptable values. More ambitious algorithms adapt to the incoming data and "learn" when events may be unusual. These can be as complex as the machine learning algorithms that can notice a dangerous pattern or as simple as flagging a problem if any number moves more than two standard deviations away from the values collected over time.

Good monitoring tools can also organize the repair process. Each alert can include areas for the team to keep notes on how they were resolving the issue. These notes can coordinate the response and provide a record in case the problem occurs again.

The best systems offer a multistep workflow that coordinates the team's response. A serious event may be cleared from the system only after each team performs its checks and decides that the problem is solved. These systems often imitate workflow tracking systems and sometimes integrate directly with them.

FUTURE TEST ARCHITECTURES

In the beginning, the monitoring systems were deliberately kept independent of the actual website to avoid corrupting the results. They were designed to simulate what a user experienced, and they relied on the same external tests used by the consultants or independent testers that businesses hired to evaluate their websites.

Many of today's solutions maintain this separation, but others are blurring the distinction by relying on the cloud's internal monitoring telemetry to watch the production servers. The cloud's internal tools often dig deep into the virtual stacks on the machines, and the load averages recovered from these can provide excellent low-level data on application performance. Of course it also means trusting the cloud to report on the cloud, a convenient connection that can lead to awkward interdependencies and errors. For this reason, some shops choose to maintain the traditional separate monitoring server on their own premises to keep an eye on the cloud stack.

As the monitoring software vendors debate just how much to bridge the gap between test software and the working system, the lines will continue to blur as they automate the responses to the tests. The monitoring system is morphing into a management system. The most common change is adding or subtracting servers as the load changes. If the tests show that response times are slowing, the test systems can trigger the creation of new servers from the cloud without waiting for an administrator to make a decision.

Some of the vendors are experimenting with triggering computationally intensive jobs for times when the load is lower. The most sophisticated machines can take information about the load, combine it with historical data, and trigger back-end processing when it seems the machines have the spare capacity. While this may often happen late at night, the flexible, test-centered approach makes it possible to run on holidays and other moments when the load promises to remain light.

Some public clouds run active marketplaces where companies bid for the time of spare servers. The smartest monitoring systems can include the current cost of server time while making decisions about how to accommodate too much load. If server time is cheap, it can purchase more machines. If computation time is expensive, it can start shutting down extra features or curtailing their quality.

Elastic cloud infrastructures and more complex application stacks are outstripping the capabilities of traditional monitoring systems. New approaches to application monitoring are blurring the distinction between test system and system under test, producing hybrid, self-diagnosing systems that can make intelligent choices about how to help the site operators at the lowest cost.