

Transforming the Load Test Process

*How Continuous APM Automates Performance and Scalability
Diagnostics while Increasing Test Center Throughput*

A Whitepaper from dynaTrace software Inc.



Executive Summary

This whitepaper discusses the shortcomings of traditional load testing and problem diagnostic approaches with today's complex applications. These shortcomings lead not only to guessed fixes and multiple test iterations, but also prohibit agility, preventing performance testing from becoming properly integrated into iterative development approaches. Consequently, time of critical test and development resources is wasted with tricky problem-hunts shortly before the final software release is scheduled to occur, limiting test coverage, frequently resulting in missed release dates and poor quality. (continued...)



Executive Summary

This whitepaper discusses the shortcomings of traditional load testing and problem diagnostic approaches with today's complex applications. These shortcomings lead not only to guessed fixes and multiple test iterations, but also prohibit agility, preventing performance testing from becoming properly integrated into iterative development approaches. Consequently, time of critical test and development resources is wasted with tricky problem-hunts shortly before the final software release is scheduled to occur, limiting test coverage, frequently resulting in missed release dates and poor quality.

That is an untenable position for any enterprise that depends on high quality applications to be delivered on time in order to maintain position in today's competitive environment. Thus, an innovative, new application performance management approach – Continuous Application Performance Management – is needed that actually exposes the application black box to deliver visibility into the dynamics of modern enterprise applications that both testers and developers need – in the first test run.

In this way, testers will be able to increase test coverage, while developers can quickly hunt down and resolve performance problems. Key is that this new approach smoothly integrates into existing processes, meaning that testers can apply it easily even without knowing the underlying application architecture, and developers can analyze problems offline without having to attend the tests live on-site in the test lab.

Additionally, you will learn how Continuous APM enables you to take the next evolutionary step to become an expert in agile test automation, empowering test centers and development with an automated test infrastructure to drive quality upstream and prevent problems early on.



TABLE OF CONTENTS

1	Power of load testing not yet unleashed	1
1.1	Load testing reports only provide a black-box view	2
1.2	Traditional paths to root cause are time-intensive	2
1.3	Inefficiency of load testing.....	3
1.3.1	Problem diagnostics block test resources	3
1.3.2	Development spends too much time in root-cause analysis	3
1.4	Potential of load-testing is not unleashed	4
1.4.1	Many hidden problems not found in load-testing.....	4
1.4.2	Missing automation to allow for agile testing	4
2	Test more apps in less time and deliver precise diagnostic reports with dynaTrace Continuous APM	6
2.1	PurePath Technology	6
2.2	Integrating diagnostics data collection into load-testing	7
2.3	Improve and Automate Diagnosis.....	7
2.4	Improve Resolution Processes through Improved Communication with Development.....	7
2.5	Identify component level regressions early on.....	8
2.6	Load-Testing becomes Agile	9
2.7	Automated and Configurable Reporting	9
3	Compare for Yourself	10
4	Case Study: Novell.....	11
5	Conclusion.....	12



1 POWER OF LOAD TESTING NOT YET UNLEASHED

Load-Testing has become more affordable due to a growing list of available commercial and open source tools. Virtualization and cloud-based testing services also reduce the need for expensive test hardware. Creating and maintaining test scripts has improved as testing tools provide good scripting support in professional IDE's like Eclipse or Visual Studio. Enhanced support for Record & Replay as well as the ability to modularize scripts to make them reusable helps testers to cope with the growing demand on writing new test scripts for constantly changing applications.

Despite all the improvements on the testing tool side, analyzing and solving the identified performance problems takes most of the time in the overall testing phase of a project. According to Gartner, most application problems found during testing root from inside the application – not from the underlying system¹. A Yankee Group report states: “Today, generating a simulated load is table stakes. Most critical is the depth of code-level visibility.”² Most testing tools lack such application visibility resulting in up to 5x more test cycles and therefore higher costs. Without such code-level visibility, problem solving resembles this lengthy and expensive process:

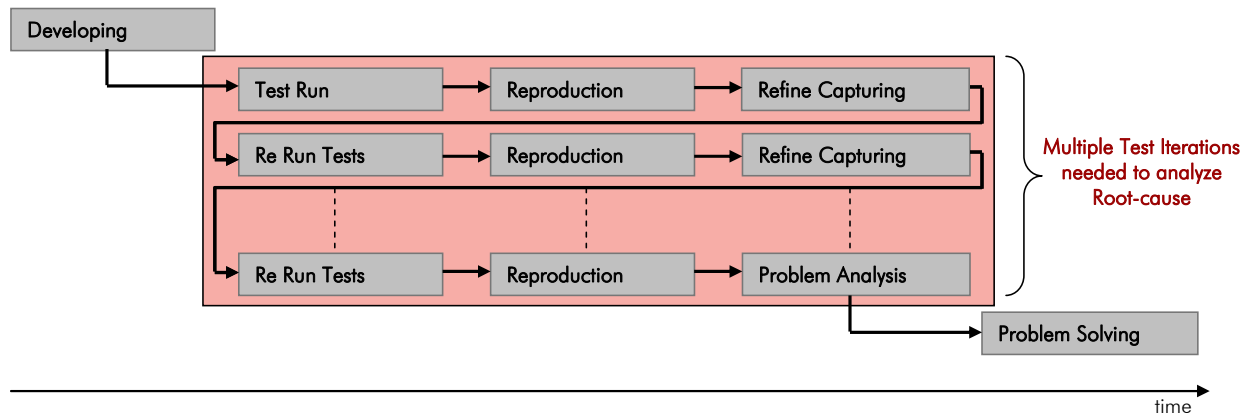


Fig. 1: Classical Performance Problem Solving Process with multiple Test Iterations

After the first test run is complete, the generated load-testing reports do not include enough diagnostics information for development to get straight to the root-cause of application issues – they more or less help identify system issues. In order to get to the root-cause of application problems, more test cycles are needed to capture additional code-level performance metrics such as logging, custom performance metrics, etc. These additional cycles involve time and effort of both testers (to run the tests) and developers (to refine data capturing). Not only is this process inefficient – the additional logging, use of debug code and special analysis tools also changes the performance characteristics of the tested application. This can lead to performance problems not being identified in testing because the tested application doesn't resemble the application being deployed.

Growing complexity of applications, heterogeneous and distributed architectures, 3rd party frameworks and virtualization add additional cycles to the test process as collecting meaningful analysis data is now also required for components that don't easily allow adding manual instrumentation for in-depth logging.

¹ In production, application issues still account for 60% of all downtime in modern service-based applications. Gartner, Managing Applications in the Age of SOA and Web Services, 2006.

² Yankee 2005 Report on Application Load Testing Market Is Poised for Growth



1.1 Load testing reports only provide a black-box view

Applications with distributed components (e.g. services) suffer from many moving parts with a non-deterministic chain of interactions and unpredictable performance side effects of components to other components.

Load testing tools interact with the application from the outside through its user interface. It does not make a difference for the tool how complex the application is on the back-end as this does not change the way test scripts are developed. Tracking down a performance problem and finding the root cause, however, is exponentially trickier in a highly-distributed, componentized application compared to traditional single or multi-tier application.

In addition to response times and transaction rates, performance counters from all individual servers are captured in order to provide some means of root cause analysis. These counters only represent average metrics of the individual services that are deployed on a machine. For example: high CPU usage on one application server hosting multiple services can be caused by either of the deployed services. There is no way to isolate the overall consumption to the individual services. In a distributed environment services deployed on different machines interact and impact each other. Further, captured CPU times on multiple machines cannot be correlated to individual service-to-service interactions.

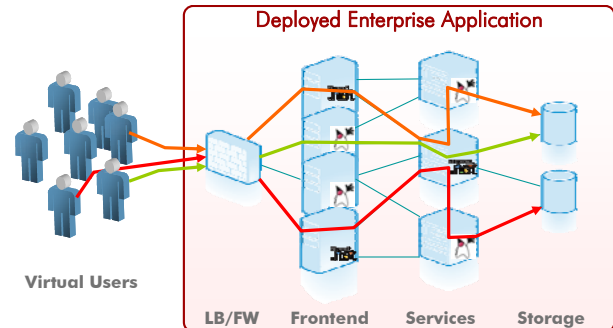


Fig. 2 Complex Application: a Black-Box for Testing

The problem traditional load-testing tools therefore face is the lack of visibility into the dynamics of the whole system, the inability to break down resource consumption into individual service calls, and the inability to track service-to-service interactions on an individual transaction basis.

The consequence of growing complexity combined with the lack of in-depth visibility is that load-testing tools only see the application as a big black box – an already-critical problem with traditional three-tier applications, but amplified with today's applications that often distribute services over multiple tiers. Capturing System Performance Counters and application logs simply don't provide the needed data for performance management and root cause analysis in complex and dynamic applications – even if it is nicely presented in a generated load testing report. Architects and Developers need more than average timings to quickly analyze and resolve problems.

1.2 Traditional paths to root cause are time-intensive

Development has various traditional approaches to get more information and some visibility into the application in order to isolate the root-cause of performance problems. However, they all come with serious drawbacks – such as excessive overhead, limited visibility, and lack of integration into the testing process in order to automate root-cause analysis:

Profilers reveal performance values down to individual code methods. Profilers, however, come with several penalties

- Profilers dramatically impact the performance characteristics of the application. As a result, less load can be handled by the application causing true performance problems to remain undiscovered
- They only provide average values on method execution times when using a sampling approach – thus missing outliers that lead to failing individual transactions
- They cannot capture context information (method arguments or SQL statements) needed for rapid root cause analysis
- Profilers only work on single Virtual Machines. In distributed enterprise applications it's impossible to follow individual transactions across tiers. Correlation – if possible – must be done manually by correlating individual profiling outputs

Application Logs & Debug Builds provide more in-depth log information from certain parts of the application. The typical process here is that the first iteration of a load test can pinpoint individual services that might have



a problem. Developers then provide specially-built Debug builds to capture more granular log-output in the identified problem area. The penalties of that approach are

- Multiple test iterations needed to pinpoint problematic components and refine log output
- Debug builds produce high I/O overhead due to heavy logging and therefore impact application performance characteristics
- Identifying problems with applications deployed across multiple tiers requires custom built logging infrastructure that allows correlation of log outputs (e.g. through a custom tracing approach that correlates log outputs from different tiers based on time stamps)

The impact on application performance, additional manual effort and change in implementation make such traditional approaches inoperative for accurate and efficient application performance management.

1.3 Inefficiency of load testing

The limited root cause analysis capabilities of load testing solutions make the overall testing and analysis process inefficient as resources are blocked on both testing and development side, minimizing overall test throughput and lowering developer productivity.

1.3.1 Problem diagnostics block test resources

In order to justify ROI on expensive test equipment (hardware and software) and the fact that more releases are being pushed by business, Test Centers need to test more applications in less time. Testing however doesn't end with forwarding load test reports to development which don't contain enough in-depth information. As problems also cannot be reproduced by developers in their local environment, Test Centers have to give developers access to the test environment in order to analyse the problems on the affected testing machines. This blocks scarce, precious hardware resources in the Test Center that should be used for more testing rather than diagnosing performance problems.

Manually collecting additional test results from various analysis tools (profilers, diagnostics tools, application logs) and correlating those results to those produced by the load-testing tool adds to the tasks that have to be done by testers - impeding them from testing applications.

All this additional effort increases cycle times and therefore prevents Test Centers from doing more with less.

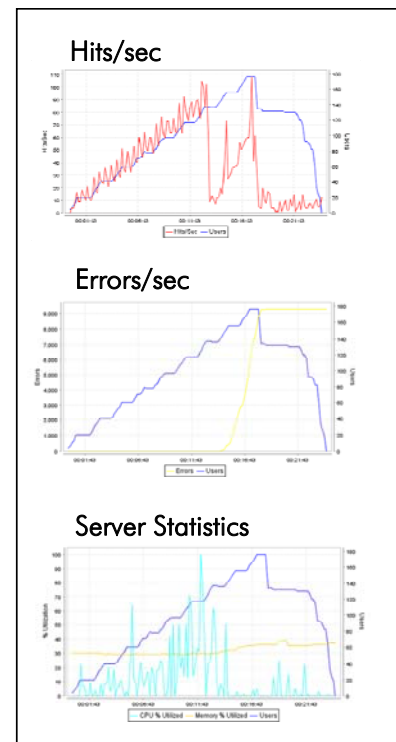
1.3.2 Development spends too much time in root-cause analysis

First iterations of load tests hardly ever result in: "The application could handle the simulated load within defined Service Level Boundaries". Testers report back problems identified to development in form of load-test reports.

As mentioned earlier, although load-testing reports like the one here show a lot of information, they do not highlight the root cause of an application issue such as a search transaction that occasionally spiked to 60 seconds throughout a long running test. Standard application logs often don't contain the in-depth information needed to analyze a newly-discovered problem.

As load-test reports lack in-depth diagnostics information down to component or even method level, development spends valuable time trying to reproduce the problem on a local environment in order to diagnose the issue with debuggers or profilers. This is often impossible because such performance problems usually don't occur in local, non-high-load environments.

The next logical step is adding additional log output to different components that might be involved in the problematic transactions. A test re-run collects the additional log information. This tedious and iterative process of refining application log output, manual log analysis and correlation, debugging and profiling is usually done by key developers or architects that have skills in performance analysis, and who have a good understanding of the overall system. Using an external or dislocated performance test center means extra communication overhead for delivering results as well as for getting access to





the test environment, which sometimes is needed to perform advanced analysis like debugging or profiling.

This time spent on root cause analysis by key technicians puts additional pressure on the development team as a whole, because they are hindered from implementing key new features – and these key technicians are usually the main resource needed to complete the project on time. This in turn puts even more pressure on other team members.

1.4 Potential of load-testing is not unleashed

It is well known that application problems caught later in the lifecycle are 10x – 100x more expensive than those found in development. Although not all problems can be identified before deployment, a high percentage of performance problems impacting the business in production can be found in testing. All you need is the right approach to Application Performance Management.

1.4.1 Many hidden problems not found in load-testing

Despite extensive testing, some problems will always escape to production and will first be identified when real users push the application. The reason is that not every single possible real-world test scenario can be tested. Most problems that “escape” testing could, however, be prevented.

An example of a common problem are individual failing components that cannot handle real world production load, but just scrape by the load in testing. Having a component-level performance and regression analysis in place will show how certain components scale compared to others. Analyzing response times with increasing load during load-testing down to component level enables predicting when components are likely to fail. They may not fail with the load during the load-test, but knowing that a certain component is almost at the breaking point allows handling this issue right away instead of waiting until the problem surfaces in production.

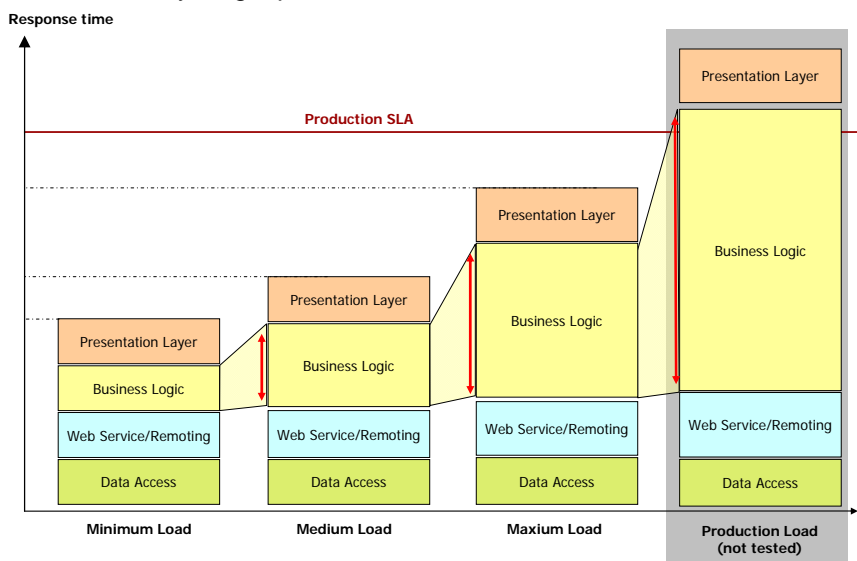


Fig. 4: Scalability Problem in the Business-logic Layer

Architectural issues – such as too many network roundtrips or database queries per transaction – are another example of component-related problems that may pass testing but will eventually result in a production failure. Analyzing individual transaction execution paths can identify these problems in testing by highlighting such architectural shortcomings.

Another example is the impact of additional logging, debug libraries or other environmental differences on the performance characteristics of the tested application. These differences compared to the real production environment hide or shift the real performance problems.

Regardless of best practices certain problems will never surface in testing. Once deployed in production - without the additional logging and with release libraries - the application exhibits different performance characteristics leading to real production problems never seen in testing.

1.4.2 Missing automation to allow for agile testing

Load Testing is usually done late in the project - disjoined from general development activities. This often seems unavoidable because the application needs to be in a certain state in order test its performance under realistic real-world load conditions. However, agile testing has proven to greatly reduce the cost of defects as tests are executed continuously as soon as code is checked-in, providing immediate feedback to developers to their most recent code changes.



Adding certain load tests – even on a smaller scale – to the mix of unit and integration tests helps to identify basic, but very common performance, scalability and architectural issues like too many data access roundtrips per request, problematic synchronization blocks, wasteful memory management, extensive I/O, etc. Identifying those basic problems early on also reduces the required load-testing cycles later in the project as the basic bottlenecks have already been removed.

Test early and continuously only works, however, if iterative load-testing activities can be automated and integrated into the development process. The lack of automation interfaces, unattended test execution and automated correlation of test results with diagnostic information either makes early load-testing impossible or requires costly manual iterations. As it stands today, some load-testing solutions offer automation interfaces and integrations into continuous integration processes via ANT, command line tools or HTTP based interfaces, which are key criteria for testing automation. These interfaces allow automatic test execution and result collection.

Due to the lack of in-depth root cause analysis, however, the captured data still misses the diagnostic information developers need, undermining its value for agile testing to a great extent, because problems must be reproduced and analyzed manually in a costly, time-consuming process.



2 TEST MORE APPS IN LESS TIME AND DELIVER PRECISE DIAGNOSTIC REPORTS WITH DYNATRACE CONTINUOUS APM

dynaTrace addresses the problems mentioned in the previous chapter by providing the required visibility into the dynamics of modern enterprise applications and embedding it into your existing load test process. Integrations to all major load-testing solutions like HP/Mercury LoadRunner, MicroFocus/Borland SilkPerformer, Microsoft Visual Studio Team System for Testers, OpenSTA, JMeter, Neotys, etc. already exist. dynaTrace provides the relevant information developers and system architects require to reconstruct the error directly from monitoring data offline, without having to reproduce it on their systems. This eliminates costly test re-runs, manual reproduction effort, and accelerates problem analysis. Instead of testers and developers spending time in multiple test iterations, the overall testing process can be reduced to the essential tasks:

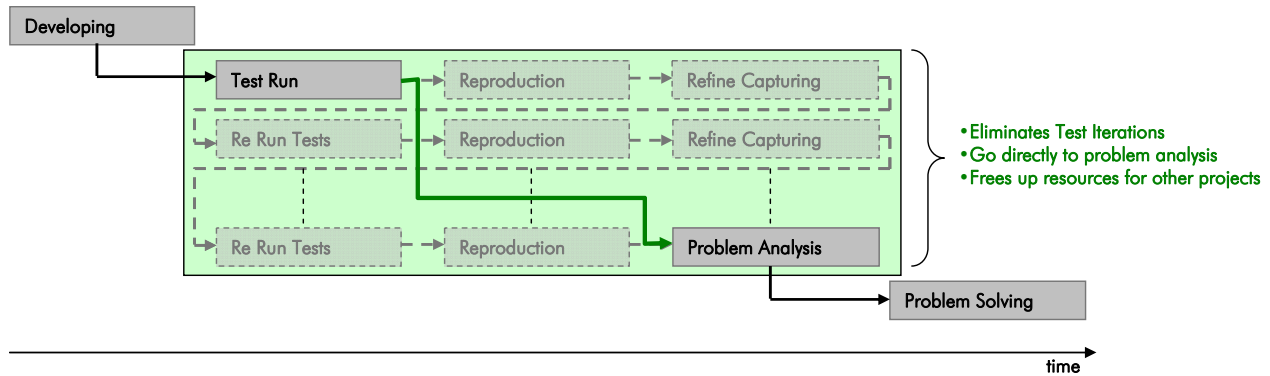


Fig. 5: Improved Performance Solving Process

The improved problem resolution process enables faster fixes, fewer and shorter test iterations - which then allows for more tests to increase test coverage and faster releases. Automating capture and analysis is the next evolutionary step. Testers become experts in agile test automation by empowering test centers and development with an automated test infrastructure.

2.1 PurePath Technology

The core of dynaTrace is its award-winning, patent-pending PurePath Technology. It integrates and extends load testing solutions to trace requests end-to-end at code-level from Virtual Users into backend application transactions. PurePath makes transparent the inner workings of applications under load, automating issue documentation and isolation, as well as reducing problem diagnostics and resolution times to minutes. It provides a common dataset that can be shared between testers, performance analysts and developers.

Additionally, dynaTrace also provides memory and thread dumps of the JVMs and CLR's running your applications - directly related to recorded PurePath transactions - to easily diagnose the root cause of memory and threading issues.

It helps you quickly and reliably find the answers you need to resolve performance problems that matter to the business:

- Assess the impact of a problem (WHAT?)
- Isolate the component causing the problem (WHERE?)
- Identify the root-cause of the problem (WHY?)

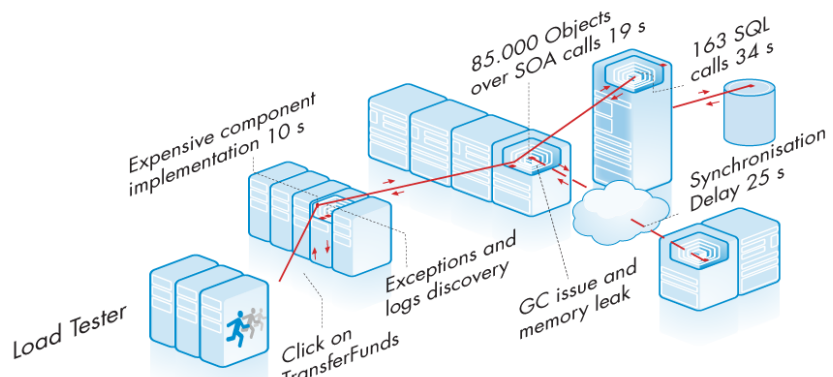


Fig. 6: PurePath traces all requests from load testing virtual users at code-level end-to-end into backend application transactions



2.2 Integrating diagnostics data collection into load-testing

Effective diagnostics data collection in test is essential to improve problem resolution processes. Test results linked directly with in-depth application performance data from dynaTrace allows you to seamlessly drill-down from total response times to code-level performance data on all individual transactions including outliers and on component-level monitoring information collected during testing. Information collection is triggered automatically by starting and stopping transactional trace data recording in testing through REST/SOAP automation interfaces. Actual test executions are directly linked to trace data using additional metadata like test case or timer name, which is passed from the test tool to the performance management solution. In the load testing script an additional HTTP header containing the transaction name is passed as part of the request. Server response also contains an additional transaction identifier enabling seamless integration of dynaTrace with the load-testing solution. This way, synthetic test transactions are directly linked to their captured dynaTrace PurePath. This enables immediate root-cause analysis of performance problems discovered in testing. In case of an error or unexpected response the unique response identifier facilitates root cause analysis. Non-web testing solutions achieve the same level of integration by providing meta-data such as test case name as part of test execution.

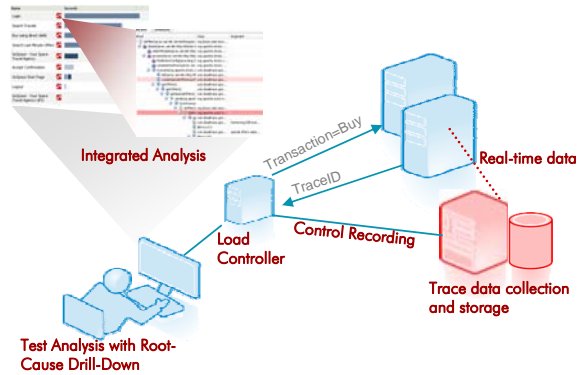


Fig. 7: Load Testing with Integrated Performance Data Collection

Additionally, data collected through dynaTrace in production can be fed back to support testing in creating more realistic load scenarios. Detailed transactional data about application behavior in production may be used to better simulate behavior of mocked components

2.3 Improve and Automate Diagnosis

In many cases application architecture and deeper code-level details are unknown by testers. Understanding the inner workings of an application, however, is a prerequisite for detecting code-level regressions and for problem diagnosis.

With dynaTrace, application components are discovered automatically by detecting relevant components and technologies such as Servlets, ASP.NET pages, EJBs, and remoting technologies. Sensor configuration for code-level tracing in custom application code is automated as part of system setup.

Additionally, dynaTrace itself analyzes all captured diagnostics data to automatically uncover most frequent application problems and their underlying root cause such as slowest web requests, slowest database statements or most frequently thrown exceptions. This enables testers to isolate and identify most common problem sources without involving development or other analysis tools.

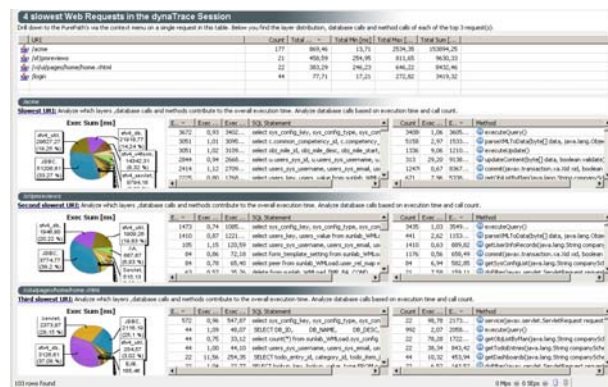


Fig. 8: Automated Transaction Analysis Unveiling Database Access Problems Causing Slow Web Request Response Times

2.4 Improve Resolution Processes through Improved Communication with Development

dynaTrace improves problem resolution processes by ending the trial-and-error approach requiring multiple test runs, and moving to a defined resolution process with a single load test run. dynaTrace enables this by automatically recording the root cause data development needs to efficiently resolve a performance problem into a diagnostics session, that they then can analyze offline on their local workstations without having to reproduce the problem in development. The responsible tester or performance engineer identifies the issue as

an application problem and forwards the diagnostics session to the responsible developer. The entire diagnostics process is executed without further tests including additional logging to gain required insight, thus eliminating the need for up to 80% of test iterations.

Development just needs to follow a defined process to pin down the problem, significantly accelerating problem resolution. First, the problematic transaction is analyzed regarding the main contributing components. When the offending component (e.g. the Web Services layer) is identified, the service calls of the transaction are analyzed – likewise with other technologies. Next, the calls are viewed in the context of the executed transaction to understand why these services are called. When the root cause has been identified the developer can easily apply the necessary code changes to fix the problem.

All this analysis is performed without additional interaction with the testing team and without additional test runs.

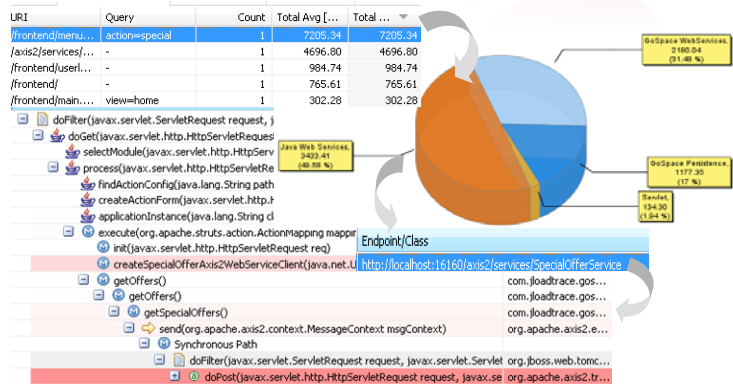


Fig. 9: Analysis of a Slow Web Request: Drill-Down to Component-level Break-down, Web Service Invocation and a Full Transaction-Trace

2.5 Identify component level regressions early on

Recording application transactions on code-level enables component-based performance analysis, e.g. based on individual layers of the application such as Data Access Layer, Web Service Layer, Business Logic and Presentation Layer. Analyzing the performance of individual components across different testable application builds as well as across different workload scenarios reveals performance and scalability issues in individual application components.

Identifying decreasing performance on component-level, even when overall response times are still acceptable, can prevent critical situations arising later in production when real-world user load is different than what has been tested in the load-testing phase.

Figure 10 highlights the Business Logic Layer's scalability problem with increased load. Even though the overall response time of the system is still acceptable it should be a warning sign to architects that there are hidden problems.

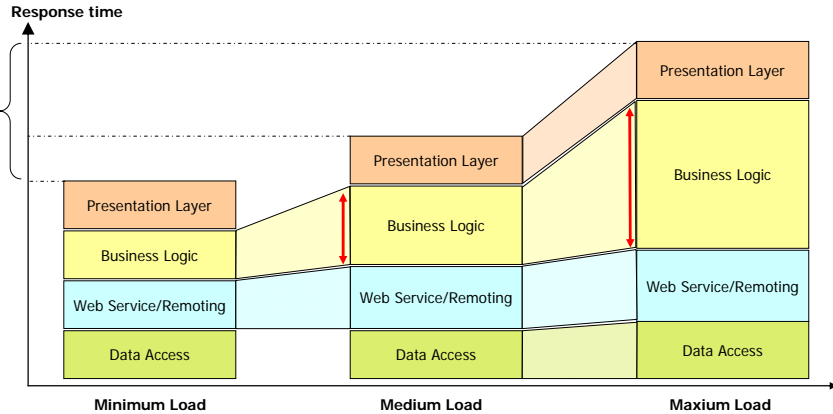


Fig. 10: Test Component Level Performance



2.6 Load-Testing becomes Agile

The earlier testing begins, the greater the test coverage – and the more problems are identified early on during the development cycle when they are most inexpensive to fix. An “Agile” approach to testing - testing early and continuously in the development lifecycle - brings many of benefits as applications being prepared for real load-tests are of better quality than applications tested for the first time.

dynaTrace provides automation interfaces to trigger an automated code-level regression analysis from the test runs of the Continuous Integration process. This way, regressions are automatically identified on a build-by-build basis. The root cause of such problems is captured and made accessible as part of the continuous testing results. Developers can act quickly on problems when memory of recent changes is still fresh. This keeps the overall quality of the software high, reduces testing cycles overall and shortens release cycles.

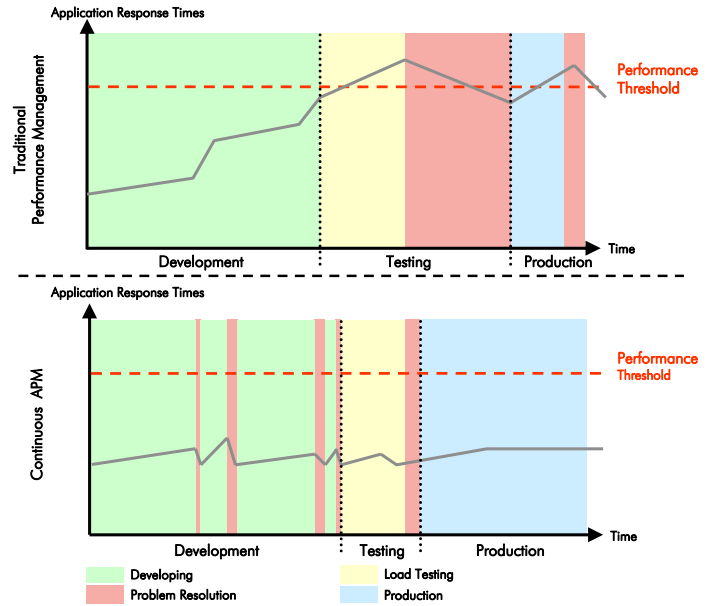


Fig. 11: Continuous Performance Management Shortens Release Cycles While Improving Quality

2.7 Automated and Configurable Reporting

dynaTrace’s dashboard technology allows easy configuration of reports to display only the information essential to a specific stakeholder. Additionally, it is also able to dynamically create dashboards based on contents of session data to further automate and simplify analysis (e.g. hit lists of slowest web requests, slowest database requests or most frequently executed database statements).

Driven by dynaTrace’s ISV’s and enterprise customers, an automation interface via REST is available to integrate report generation into the overall testing process. This ensures that interactive diagnosis reports, including automated analysis, are made available automatically to test engineers, architects or developers. Reports can also be exported as PDF, HTML or as CSV and XML for further processing with external tools.

Interactive diagnosis reports on individual sessions enable a highly efficient root cause analysis, as they allow you to dive into poor performing transactions down to code-level from any angle after the test run is completed, just as in a live session. As such reports even include contextual information like method arguments, SQL statements or servlet parameters, developers have all the data they need to resolve a problem available after the first test run – capturing additional data or reproducing the problem locally is no longer required.

Moreover, dynaTrace provides pre-built reports across performance data from subsequent test runs to automatically identify performance regressions on component-level introduced with new test builds, making proactive performance management simple to implement.



3 Compare for Yourself	dynaTrace	other solution
Reporting and Communication with Development		
Role-specific dashboard reports to display only the information essential for a specific stakeholder (e.g. overview reports for tester, interactive diagnostic reports for performance engineers)	✓	
Automated performance analysis to identify the root-cause of most frequent performance issues (e.g. too many SQL or remoting calls, frequently thrown exceptions, too chatty applications) without having to understand application internals	✓	
Performance regression reports on component-level to identify hidden performance degradations introduced with the new test builds that do not yet surface on end-user visible level and are thus invisible to load testing tools	✓	
Full automated reporting that is integrated via integration APIs into the overall testing process to automatically deliver reports to test engineers, performance analysts and software architects	✓	
Offline code-level diagnostics to enable developers and architects to diagnose performance issues of individual transactions interactively down to the offending line of code after the test has been completed, relieving developers from debugging issues online or to reproducing them locally	✓	
Reuse application instrumentation from development in QA enabling engineers and architects to define measurement granularity, so they get exactly what they need from QA.	✓	
Depth of Root-Cause Analysis		
Capture root-cause data for each individual transaction , and not averages or after-the-fact, enabling diagnosis of outlier transactions that just occur once, eliminating problem reproduction	✓	
Precisely trace transaction executions down to code-level incl. contextual data (e.g. method arguments, SQL-binds, exceptions/logs, etc.) to precisely identify the code-segments causing a problem in the first test run, eliminating the need to iteratively rerun tests with changing log-levels to isolate a problem	✓	
Low-overhead, small footprint to enable always-on tracing of every single application transaction without impacting application performance characteristics	✓	
Trace transaction end-to-end across multiple tiers (logical or physical) even across Java/.NET technology boundaries to understand remote communication overhead of distributed high performance applications	✓	
Analyze transaction metrics in context of server resource metrics to determine whether the performance problem is caused by configuration issues or programming issues.	✓	
Analyze Java and .NET virtual machines in terms of memory and thread activity even in high load environments to uncover and diagnose memory leaks, heavy garbage collector activity, high memory usage, synchronization and locking problems causing stability, performance and scalability problem	✓	
Deployment and Configuration		
Configuration-free agents for automated, centralized deployment for low maintenance	✓	
Automatically configures application instrumentation through wizards that automatically detect relevant components and recommend instrumentation depth to maximize visibility while minimizing overhead, without having to understand application internals in detail	✓	
Adapt instrumentation configuration on-the-fly without having to restart the application under test to react to test results while the test is running, accelerating performance analysis and tuning	✓	
Integration with Test Environments		
Integration with all load testing solutions for automatic performance data collection and analysis to speed up problem diagnostics and improve communication with development – without having to lock yourself into a single vendor.	✓	
Direct drill-down from symptoms in load testing reports into code-level performance diagnostics quickly revealing the problem's root cause	✓	
Open platform supported through a Community Portal for easy sharing and fast information access to extend out-of-the-box functionality with their own 3rd-party developed (open source) plug-ins such as custom monitors	✓	



4 Case Study: Novell

As one of the best-known enterprise software companies on the planet, Novell delivers the best-engineered Linux and IT management software to help enterprises lower cost, complexity and risk on virtually every platform.

Challenge: In the current difficult economy, Novell's Engineering group has been asked to do more with less while maintaining the company's standard for high-quality software. Novell's traditional approach for dealing with application performance issues was under increasing stress given the demands placed on Engineering. In the past, when QA uncovered application performance issues under load, they would have to find the relevant developers and bring them into the test environment to try to isolate the underlying causes. Lacking an integrated diagnostic solution, the geographically-dispersed teams were forced to endure a labor-intensive, inefficient approach that involved too many people and took too long to resolve problems, particularly given Novell's distributed agile development process.

Solution: After thorough review, Novell selected dynaTrace to define a new standard in test center productivity and collaboration with development. dynaTrace's continuous APM platform with PurePath gives Novell's globally-dispersed engineering team a common framework for collaboration to quickly and efficiently identify, diagnose, and resolve application performance issues.

Results: "By isolating problems faster with dynaTrace we're able to increase our performance test throughput by 2-3x without having to increase our staff. Before we might have had to run five or six test round-trips before we could isolate the issue, tying up not only the hardware, but tester's and developer's time. Now with dynaTrace's diagnostic capabilities, we can often isolate root-cause the first time around, package up the PurePath and send that directly to the developer so they know exactly where to find the issue."

These efficiency gains provided by dynaTrace are leveraged by both development and QA. With dynaTrace, testers can run fewer, more productive tests during debugging; running fewer tests during debugging means they have more time to run more productive tests during performance/regression testing and capacity planning phases. And all those tests return better results to development, enabling them to focus more on what they enjoy – creating new features and functionality – rather than chasing bugs.

Raise Productivity: With dynaTrace's PurePath technology preserving individual transactions for later review, problems can be analyzed whenever the team member has time, rather than when they are discovered.

Capturing and analyzing the data is now also split between Testers and Developers. Before using dynaTrace, Novell needed two people sitting in on every problem: the tester that drove the test to induce the problem, and the developer that then analyzed it, with both working on the same test machine. dynaTrace eliminated the need of having two team members – one of them mostly idle – working on the same problem.

dynaTrace's ability to run under load with virtually no overhead also frees up resources for more productive uses by automating away labor-intensive manual tasks. Because dynaTrace is on for the duration of a test, it eliminates the need to rerun certain tests to capture additional log and tracing information that developers put into the code.

Improve Lifecycle Collaboration: After issues have been diagnosed and resolved by development, Novell's QA engineers use dynaTrace's charting and diffing features to verify that code changes actually deliver the intended improvements. This feedback is valuable to developers for verification, strengthening their confidence in bug fixes. It also helps to ensure that bugs are killed as early as possible in the lifecycle, further reducing costs.



5 Conclusion

Although load testing has made great stride to become more efficient and affordable, it still lacks the visibility to efficiently diagnose the root-cause of performance problems and to make application behavior transparent. Consequently, conventional load testing and diagnostic processes require multiple iterations, drawing time away from both the tester to run the test and the developers to refine data capturing. Not only is this process highly inefficient, but the overhead introduced with classical methods like profiling or logging also changes the performance characteristics of the tested application.

This can lead to performance problems not being identified in testing because the tested application doesn't resemble the application ultimately deployed. This becomes even worse in modern application architectures with framework-based approaches, which do not allow adding manual instrumentation for in-depth logging. Finally, a cumbersome analysis process reduces agility, obstructing continuous performance testing in agile and iterative development processes to find and fix common performance issues early on, while they are still inexpensive to resolve.

dynaTrace addresses these problems by providing the required visibility into the dynamics of modern enterprise applications. Tight integrations with existing load testing solutions and straightforward deployment and configuration that do not require expertise in the architecture of the application under test, smoothly embedding dynaTrace into existing load test processes.

In this way, dynaTrace is first to equip developers and system architects with the information they truly require to reconstruct the error directly from the captured diagnosis data offline, without having to reproduce it on their systems. This eliminates costly test re-runs, reproduction effort and accelerates problem analysis. Instead of having testers and developers spending time in multiple test iterations, the overall testing process can be reduced to essential tasks. Additionally, performance reports down to component-level enable testers to discover new types of problems that are invisible to load testing tools, significantly increasing test coverage.

Automating capturing and analysis is the next evolutionary step. Testers become experts in agile test automation and empower test centers and development with an automated test infrastructure. This way, development teams are empowered to prevent problems early on, when fixing problems is least costly by integrating code-level performance analysis into their agile processes.

Bottom line, dynaTrace enables test teams and development to

Reduce Mean-Time-To-Repair by 90%

- Automate issue documentation and eliminate the need to reproduce them
- Automate problem isolation and diagnostics
- Accelerate final problem resolution

Reduce & accelerate required test cycles

- Eliminate test runs with additional logs to drill into certain problem areas
- Eliminate manual log file collection and problem description

Uncover hidden performance issues inside applications

- Find component-level issues invisible to load testing tools

Visit us at www.dynatrace.com to see how dynaTrace customers are delivering precise diagnostics and increasing test coverage in less time with Continuous Application Performance Management in Testing.

dynaTrace software Inc.

95 Hayden Avenue, Waltham, MA 02451, USA, T +1 781.674.4000 F +1 (781) 2075365

Headquarters EMEA: dynaTrace software GmbH

Freistädter Str. 313, 4040 Linz, Austria/Europe, T+ 43 (732) 908208, F +43 (732) 210100.008

E: takeaction@dynatrace.com

All rights reserved

dynaTrace software is a registered trademark of dynaTrace software GmbH. All other marks and names mentioned herein may be trademarks of other respective companies. (091007)