THE DZONE GUIDE TO

PERFORMANCE & MONITORING

VOLUME III

BROUGHT TO YOU IN PARTNERSHIP WITH

New Relic.

bmc

NASTEL
Driving Business Transaction Performance ®

APPDYNAMICS

ca technologies

catchpoint™

dynatrace

fusion reactor™

IBM

riverbed®

DEAR READER,

Developers face dozens of "impedance mismatches" every day. The most fundamental is perhaps the reduction of the non-sequential (non-procedural program design) to the sequential (execution). Most software isn't written in machine code. Most truly step-by-step descriptions of interesting systems are unusably inefficient.

This is the magic of Church-Turing, that the dimensional reduction from human-intelligible symbols to machine-usable symbols effectively loses nothing—that any computable function can be computed by a bunch of steps executed in linear order. But the *conceptual* mismatch puts the burden of optimal structure mapping squarely in the brain of the developer. In my head, I'm figuring out what InterfaceConsumerInterceptor is like and what it can do. But javac and the JRE or csc and .NET are doing...well who knows what. The operating system adds another layer of *well that's not obvious on the face of it*, and again the system architecture, and again the NIC, and then every packet-forwarder, until what seemed like truly beautiful code when you wrote it has become...

The epicycles don't end at these higher levels. Modern processors execute plenty of instructions out of order—even on a single core. Techniques as simple (or is that disconcerting?) in principle as branch prediction in practice further fuzzify the developer's sense of what the computer will actually do with their code. Low-level caches, pipelining, and other optimizations also make assumptions about probable execution dependencies, making even machine code less than fully deterministic. And then there's the abstraction of the virtual machine...

In short: *of course* designing for performance is absolutely essential; but runtime is so crazy a variable that we can reasonably blame too-early optimization for a non-negligible chunk of lousy UX and unmaintainable code.

So our latest Guide to Performance and Monitoring covers both the static and dynamic, the verifiable and the unknowable sides of building and maintaining performant applications.

Read it, monitor your results, and let us know what you think.

BY JOHN ESPOSITO

EDITOR-IN-CHIEF, DZONE RESEARCH@DZONE.COM

TABLE OF CONTENTS

EDITORIAL

JOHN ESPOSITO
RESEARCH@DZONE.COM
EDITOR-IN-CHIEF

CAITLIN CANDELMO
PUBLICATIONS MANAGER

ANDRE POWELL
EDITORIAL OPERATIONS MANAGER

G. RYAN SPAIN
ASSOCIATE EDITOR

MATT WERNER
ASSOCIATE EDITOR

MICHAEL THARRINGTON
ASSOCIATE EDITOR

TOM SMITH
RESEARCH ANALYST

BUSINESS

RICK ROSS
CEO

MATT SCHMIDT
PRESIDENT & CTO

JESSE DAVIS
EVP & COO

KELLET ATKINSON
VP OF MARKETING

MATT O'BRIAN
SALES@DZONE.COM
DIRECTOR OF BUSINESS DEVELOPMENT

ALEX CRAFTS
DIRECTOR OF MAJOR ACCOUNTS

CHRIS SMITH
PRODUCTION ADVISOR

JIM HOWARD
SR ACCOUNT EXECUTIVE

ANDREW BARKER
ACCOUNT EXECUTIVE

JIM DWYER
ACCOUNT EXECUTIVE

CHRIS BRUMFIELD
ACCOUNT MANAGER

ART

ASHLEY SLATE
DESIGN DIRECTOR

SPECIAL THANKS
to our topic experts, Zone Leaders, trusted DZone Most Valuable Bloggers, and dedicated users for all their help and feedback in making this report a great success.

WANT YOUR SOLUTION TO BE FEATURED IN COMING GUIDES?
Please contact research@dzone.com for submission information.

LIKE TO CONTRIBUTE CONTENT TO COMING GUIDES?
Please contact research@dzone.com for consideration.

INTERESTED IN BECOMING A DZONE RESEARCH PARTNER?
Please contact sales@dzone.com for information.

EXECUTIVE SUMMARY

As Tony Hoare notoriously observed, "Premature optimization is the root of all evil:" that is, the benefits of absolutely maximal optimization are usually much lower than the increased cost of maintenance and debugging that results from the brittleness caused by that optimization. On the other hand, the natural tendency of OOP to prioritize form over performance can generate a codebase that is highly readable but partitioned such that performance-oriented refactoring may prove extremely difficult. To help you steer between the Scylla of overeager optimization and the Charybdis of runtime-indifferent code structure, we've split this publication between ways to design performant systems and ways to monitor performance in the real world. To shed light on how developers are approaching application performance, and what performance problems they encounter (and where, and at what frequency), we present the following points in summary of the most important takeaways of our research.

Application code is most likely to cause performance problems frequently; database performance problems are most challenging to fix:

DATA Frequent performance issues appear most commonly in application code (43% of respondents) and in databases second most commonly (27%). Challenging performance issues are most likely to appear in the database (51%) and second in application code (47%).

IMPLICATIONS Enterprise application performance is most likely to suffer from higher-level, relatively shallow suboptimalities. Deep understanding of system architecture, network topology, and even pure algorithm design is not required to address most performance issues.

RECOMMENDATIONS Optimize application code first and databases second (all other things being equal). On first optimization pass, assume that performance problems can be addressed without investing in superior infrastructure. For common performance bottlenecks and key latency numbers, see our infographic on page 26 and checklist on page 29.

Parallelization is regularly built into program design by a large minority (but still a minority) of enterprise developers:

DATA 43% of developers regularly design programs for parallel execution. Java 8 Parallel Streams are often used (18%), slightly more frequently than ForkJoin (16%). ExecutorService was most popular by far, with 47% using it often. Race conditions and thread locks are encountered monthly by roughly one fifth of developers (21% and 19% respectively). Of major parallel programming models, only multithreading is often used by more than 30% of developers (81%).

IMPLICATIONS Enterprise developers do not manage parallelization aggressively. Simple thread pool management (ExecutorService) is much more commonly used for concurrency than upfront work splitting (ForkJoin), which suggests that optimization for multicore processors can be improved.

RECOMMENDATIONS More deliberately model task and data parallelization, and consider hardware threading more explicitly (and without relying excessively on synchronization wrappers) when designing for concurrency. For fundamentals of parallel algorithm design, see "Working in Parallel: On the Complications of Parallel Algorithm Design" on page 22 below.

Performance is still a second-stage design consideration, but not by much:

DATA 56% of developers build application functionality first, then worry about performance.

IMPLICATIONS Extremely premature optimization is generally recognized as poor design, but performance considerations are serious enough that almost half of developers do think about performance while building functionality.

RECOMMENDATIONS Distinguish architectural from code-level performance optimizations. Set clear performance targets (preferably cascading from UX tolerance levels) and meet them. Optimize for user value, not for the sake of optimization. For performance optimization of modern, highly modular architectures, see "Microservices Performance Patterns" on page 18 below. For performance insights at lower levels, see "Know When (and When Not) to Blame Your Network" on page 14 below.

Manual firefighting, lack of actionable insights, and heterogeneous IT environments are the top three monitoring challenges:

DATA 58% of respondents count firefighting and manual processes among the top three performance management challenges. 49% count lack of actionable insights to proactively solve issues. 47% count rising cost and complexity of managing heterogeneous IT environment.

IMPLICATIONS Performance management is far from a solved problem. Monitoring tools and response methods are not providing insights and solutions effectively, whether because they are not used adequately or need feature refinement.

RECOMMENDATIONS Measure problem location, frequency, and cost, and compare with the cost (both monetary and performance overhead) of an additional management layer. Consider tuning existing monitoring systems or adopting new systems (e.g. something more proactive than logs). For monitoring targets and tactics, see "Effective APM: Find and Fix the Things That Matter" on page 8 below. For the economics of web performance optimization, see "Why You Need to Know Your Pages' Conversion Impact Score" on page 46 below.

KEY RESEARCH FINDINGS

- 594 IT professionals responded to DZone's 2016 Performance & Monitoring survey
- The top three industries in which the survey takers work are Software Vendors (22%), Finance/Banking (14%), and E-Commerce/Internet (9%)
- The primary roles included Developer Team Leads (39%) and Development/Engineering roles (36%)
- 26% work at a company with more than 500 people; 22% work where there are more than 10,000 employees
- 41% of respondents work at companies whose headquarters are located in Europe; 32% in the USA
- Respondents have years of experience as IT professionals, with 51% having over 15 years' experience

SOFTWARE AND INFRASTRUCTURE STILL HAVE FREQUENT PERFORMANCE PROBLEMS

When asked about the last time they solved a performance problem in their software, most respondents (26%) answered that they had done so "this week," which was a similar result compared to 2015's survey. The second most popular answer this year was "in the last 3 months" at 23%, followed by "in the last two weeks" at 17%. All in all, 81% of respondents answered "in the last 3 months" or less, showing that software still has frequent performance problems that developers need to address. Application code (43%) remains the area of the technology stack that tends to have the highest frequency of performance issues, while malware remains the one with little to no issues, where 61% of respondents had either very few issues or none at all.

Respondents were also asked to note the last time they had to solve a performance problem in their infrastructure, and the majority (21%) said "in the last three months" followed by "this month" at 17%, and "this week" at 14%. Compared to 2015's survey results, where the most respondents (19%) noted "over a year ago" as the last time they worked on infrastructure performance problems, there is a clear shift to having more frequent performance problems that require immediate attention.

DATABASES POSE A CHALLENGE; FINDING THE ROOT CAUSE REMAINS AN ISSUE

There was another shift this year in the technology stack that tends to have the hardest-to-fix performance issues. In 2015, networks (now in 4th in 2016 at 46%) had the most challenging performance issues to fix, whereas this year 51% of the respondents noted that database performance issues were the toughest to fix. The survey takers also listed workload (49%) and application code (47%) as having hard-to-fix performance issues. In contrast, 32% of respondents said that failing/old hardware was the easiest to fix performance challenge, replacing last year's easiest issue to fix: malware.

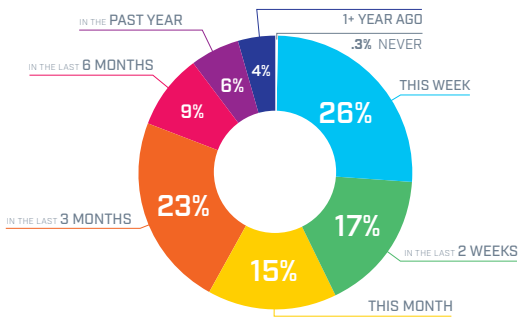
As applications become more advanced, so do the causes of the issues. Over half (52%) of the survey takers said that finding the root cause of an issue remains the most time-consuming part of fixing a performance-related problem. There does not appear to be any improvement in optimizing this process, as it was also the most time-consuming component in 2015's survey. Another time-consuming aspect includes collecting and interpreting various metrics (36%). On the other end of the spectrum, the least time-consuming component of fixing a performance-related issue remains communication/managing people.

Even though finding the root cause of a problem is the most time-consuming component of fixing a performance related issue, the majority (54%) of respondents noted that it takes an average of less than a week for their team to solve a performance related problem. These problems—particularly encountering database-related problems such as slow database queries (33%), and too many database queries (27%)—were, on average, encountered on a monthly basis. On the opposite end, 46% of the respondents said they rarely remain in the dark about the root cause of an issue.

MONITORING TOOLS AND APPLICATION LOGS ARE KEY

As a whole, respondents said that monitoring tools (32%) discovered the most

01. WHEN WAS THE LAST TIME YOU HAD TO SOLVE A PERFORMANCE PROBLEM IN YOUR SOFTWARE?



02. WHAT IS USUALLY THE MOST TIME CONSUMING PART OF FIXING A PERFORMANCE ISSUE?

	NOT	SOMEWHAT	VERY
COLLECTING AND INTERPRETING VARIOUS METRICS	9.6%	54.4%	36%
FINDING THE ROOT CAUSE OF THE ISSUE	4%	44.4%	51.5%
FIGURING OUT A SOLUTION TO THE ISSUE	11.3%	67.5%	21.2%
COMMUNICATION/MANAGING PEOPLE TO ADDRESS THE ISSUE	28.6%	46.5%	24.9%

performance issues. If systems are being monitored and reviewed consistently, then they tend to catch more performance issues than any of the other tools. Those who do not rely on monitoring tools run performance tests to discover performance issues, with 22% of respondents favoring this. Only 8% of the survey takers credited dumb luck for discovering performance issues.

Much like 2015, this year's respondents also favored application logs, as 89% of them said that these were one of the main tools their teams use to find the root cause of a performance problem. The second most commonly used tool for finding the root cause of a performance issue are database logs, with 68% of respondents relying on them. Monitoring, logging, and tests are three of the key components used to help discover problems early enough to fix them before they begin to negatively affect an application's performance.

SIMULTANEOUS USER LOADS VARY FOR APPS, FEW SERVERS ARE USED, AND PARALLEL EXECUTION IS AN AFTERTHOUGHT

The answers were pretty evenly split amongst the options when the survey takers were asked what the max simultaneous user load is for the main application their team works on. The majority—only 17%—said they use 1,001 – 5,000; 13% use 101 – 500; and 12% use 21 – 100. When asked how many servers they use at their organizations, 38% of the respondents said they they use fewer than 20 (this included IaaS and on-premises servers).

Over half (57%) of the developers surveyed do not regularly design their programs for parallel execution. When asked which parallel programming frameworks, standards, and APIs they use, 47% said they **often** used Executor Service (Java), while 33% *occasionally* use ForkJoin (Java) and 29% *occasionally* use Web Workers (JavaScript). As for parallel algorithm design techniques used, 63% most often use load balancing. 81% of respondents often use multithreading as their parallel programming model of choice. The respondents noted that they run into concurrency issues (race conditions, thread locks, mutual exclusion) only a few times a year.

APPLICATION PERFORMANCE IS STILL SECONDARY, THOUGH AWARENESS OF IT IS GROWING

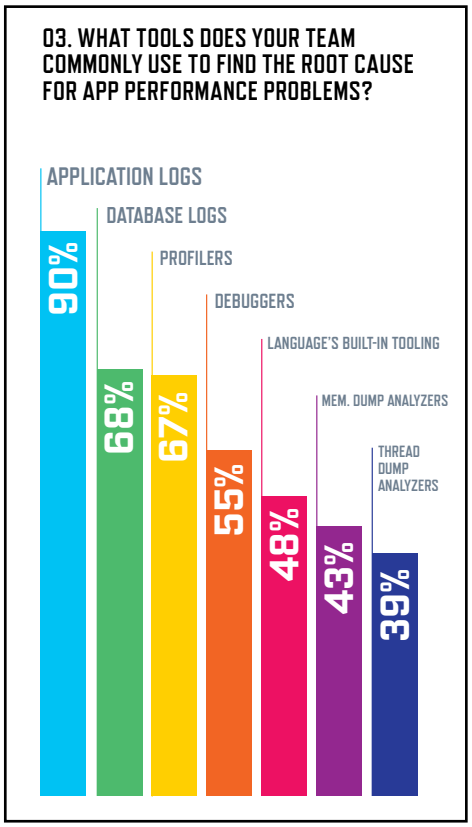
The majority of respondents (56%—though down from 62% in 2015) said that they build their application functionality first, and then they worry about performance. More people this year have performance in mind from the start when building applications, as 41% said that they build performance into the application from the start, which is up from 35% in 2015.

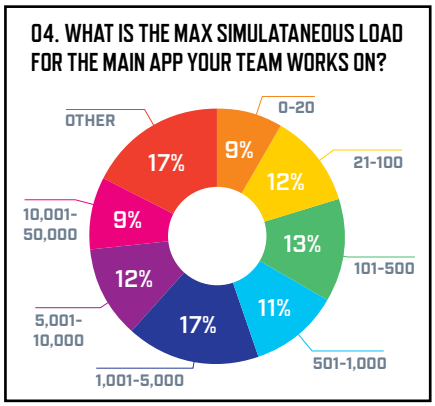
When it comes to monitoring tools used within their organizations, respondents noted that they use many different tools, with 36% using Nagios, 22% LogStash, and 21% using their own custom/homegrown tools. Furthermore, organizations are comfortable with the tools they currently use—60% of respondents said that they are not looking to switch to a new performance monitoring tool within the next 6 months. They also mainly use free and open-source tools, with 56% of respondents preferring this.

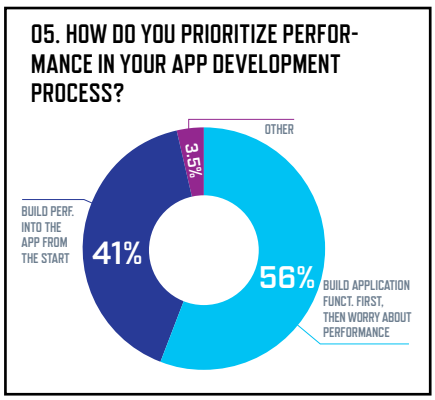
PERFORMANCE MANAGEMENT IS MOVING IN A POSITIVE DIRECTION

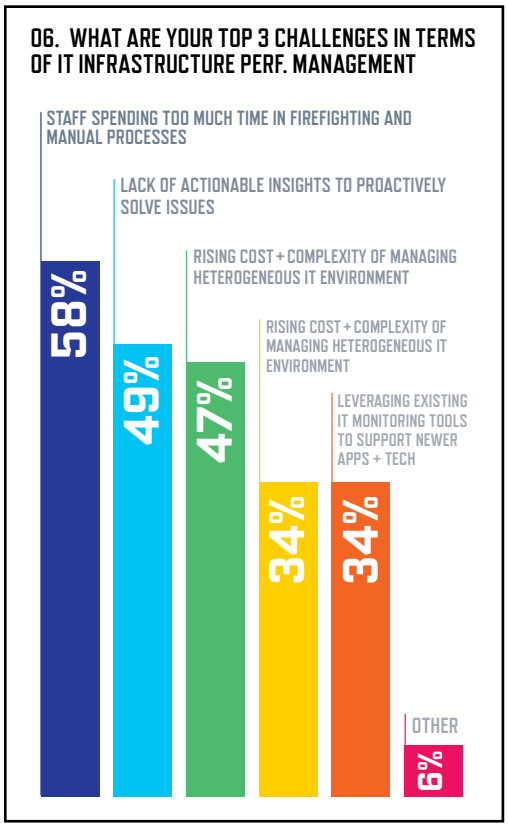
Although IT professionals have begun to put more emphasis on the importance of performance monitoring and testing, there are still some challenges that they face. The respondents said that the top challenge in IT infrastructure performance management (58%) is that the staff is spending too much time in firefighting and manual processes. The second biggest challenge, at 49%, is the lack of actionable insights to proactively solve issues. And the third biggest challenge, with 47% of respondents, was the rising cost and complexity of managing heterogeneous IT environments. With the increased and more streamlined use of performance monitoring tools, these challenges will be minimized.

Another component that can be adding to the abovementioned challenges is the separation of development and operations. According to the survey, 61% of respondents said that Dev and Ops have their metrics gathered in separate siloes. Though this is still the majority, this is down slightly from 64% in 2015. The more information is combined and shared amongst teams, the more streamlined performance management will be.



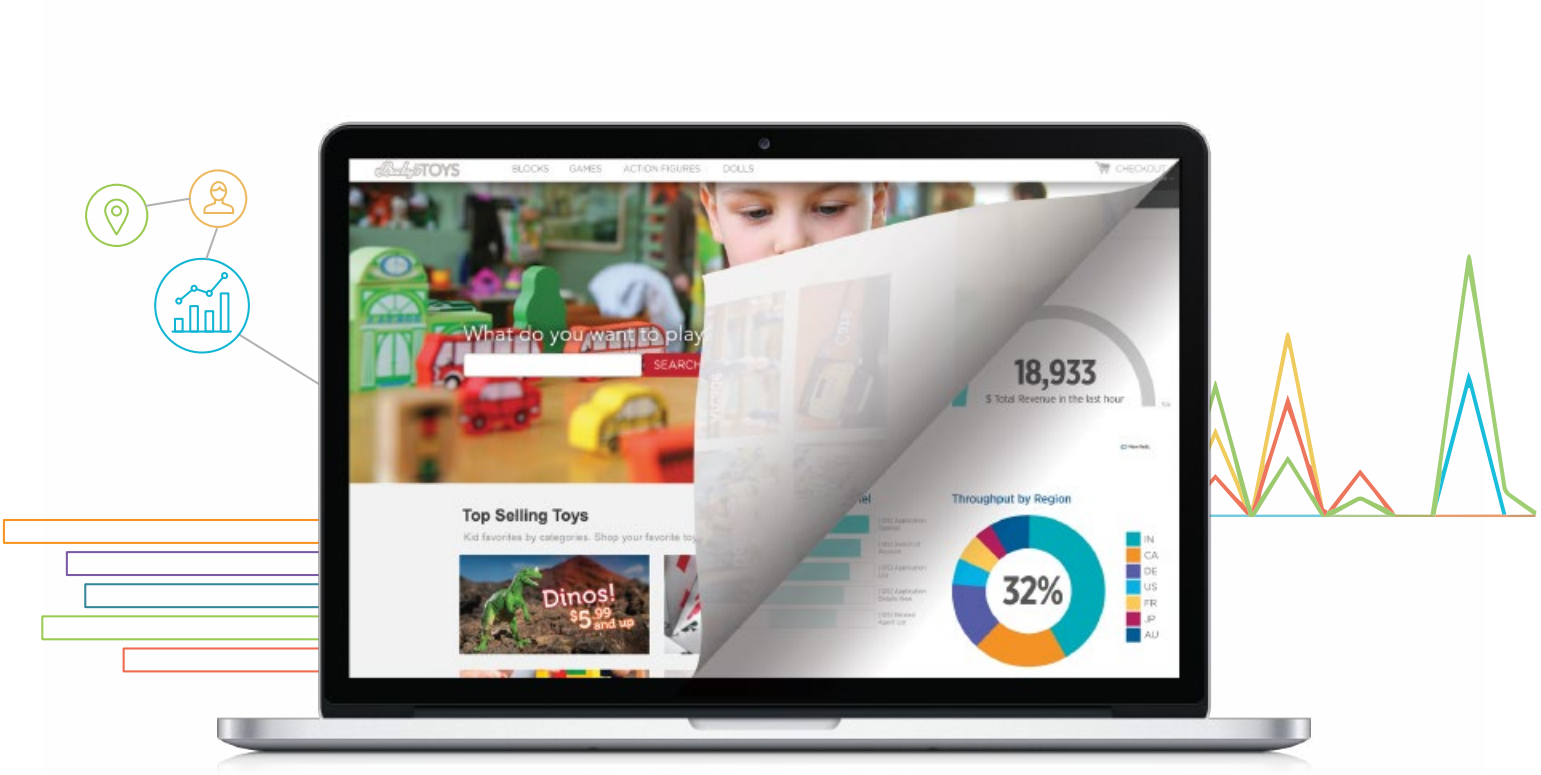






One source of truth.

See all your data. Boost performance. Drive accountability for everyone.



IT Operations
Faster delivery.
Fewer bottlenecks.
More stability.

Mobile Developers
End-to-end visibility,
24/7 alerting, and
crash analysis.

Front-end Developers
Reduce resolution times
and spend more time
writing new code.

App Owners
Track engagement.
Pinpoint issues.
Optimize usability.

Simple. Powerful. Secure.
Find out why New Relic helps you build and run great software at
newrelic.com/why

New Relic.

Reactive vs. Proactive Troubleshooting: Mastering the Art of Performance

Between all the languages, frameworks, containers, clouds, and other critical building blocks of today's applications, figuring out what's causing a bottleneck in performance can often turn into a time-consuming—and often frustrating—task. As long as you know where to look and have the right kind of visibility, finding and fixing problems doesn't have to be a painful process.

WHEN THINGS GO WRONG

When your app is broken or slow, the first thing you're going to want to know is the impact and severity of the issue. How many customers has it impacted? And for how long? In order to rapidly triage and reduce mean time to resolution, look at:

- *Backend processes:* Response times, errors, and transactions are all essential elements here. You want to quickly pinpoint if it was your code, backend server, or cloud service that caused an issue.
- *Frontend code:* With more and more code running on the client-side, you're going to want to see everything that happened from the first page load to the final click that completed a user's transaction.

In an ideal world, you want to avoid being in a reactive situation. You want to have a sense of confidence when your application is deployed—and the best way to do that is through pre-launch optimization.

WHEN THINGS *COULD* GO WRONG

Before any launch, you should proactively monitor the core metrics above, run test monitors, and set up advanced composite alerts that have context associated with their failures. All of this data should be feeding into a single analytics dashboard that can be used across developers, operations, and product teams, so there's a shared understanding of performance across the organization.

This way, whether you find yourself in a reactive or proactive scenario, you're well-equipped to resolve the issue quickly and go back to doing what you do best: writing new code, not troubleshooting it.

WRITTEN BY TORI WIELDT

DEVELOPER ADVOCATE, **NEW RELIC**

PARTNER SPOTLIGHT

New Relic Software Analytics Cloud BY NEW RELIC

New Relic

New Relic gives you deep performance analytics for every part of your software environment.

CATEGORY
APM

NEW RELEASES
Daily

OPEN SOURCE?
No

CASE STUDY

One of the fastest-growing digital properties in the U.S., Bleacher Report is the leading digital destination for team-specific sports content and real-time event coverage. To improve performance, the company embarked on a multi-year journey to turn its monolithic web application into a microservices-based architecture. New Relic has been there each step of the way, helping the Bleacher Report team stay on top of performance monitoring, proactive load testing, and capacity planning. Not only is the software analytics tool helping save time and money by making the team's code more efficient (and in turn, requiring fewer servers), but it also helps Bleacher Report respond more quickly and effectively to issues reported by users. "I use New Relic every day," says Eddie Dombrowski, senior software engineer. "It helps me find ways to make our applications perform better and prioritize which areas to address."

STRENGTHS

- Performance monitoring across applications, browsers, devices, and more
- Customer experience management for web and mobile channels
- Proactive root cause analysis anywhere in the stack
- Extensible platform offering partner integrations, open APIs, and 100+ plugins
- Secure, multi-tenant SaaS architecture delivering value out of the box within minutes

NOTABLE CUSTOMERS

- Hearst
- Trulia
- Lending Club
- HauteLook/ Nordstromrack.com
- MercadoLibre
- MLBAM
- Airbnb

BLOG **blog.newrelic.com** | TWITTER **@newrelic** | WEBSITE **newrelic.com**

Effective APM: Find and Fix the Things That Matter

BY JON C. HODGSON
APM SUBJECT MATTER EXPERT, **RIVERBED TECHNOLOGY**

QUICK VIEW

01
Data granularity is critical. Transaction & metric sampling can completely miss intermittent issues and may mislead you into solving symptoms instead of the root cause.

02
Beware the Flaw of Averages. The only way to truly understand the end-user experience of all users is by capturing all transactions and leveraging Big Data to analyze them.

03
Methodology is as important as the data. Ask the wrong questions, or ask the wrong way, and you'll waste time fixing the wrong things.

Over the past 20 years as an application performance specialist, I've witnessed APM evolve dramatically from its roots in simplistic server monitoring, to continually adding impressive (but now ubiquitous) capabilities such as code instrumentation, multi-tier transaction tracing, and end-user experience monitoring. Although the feature lists of APM tools continue to grow, organizations continue to have major performance issues, which they're unable to pinpoint and resolve even after months of effort. In helping to solve these problems, I noticed common themes as to why they eluded detection and resolution for so long.

The quality of the APM data is the number one reason why performance problems go unsolved. All tools claim to collect metrics about the environment and trace end-user transactions, but the way this data is captured, stored, and displayed ultimately dictates the value that data provides in detecting the presence of an issue, or accurately identifying its root cause. Many tools are fundamentally flawed in this regard.

The number two reason is the methodology of the troubleshooter. Even in cases where high-quality data exists, if you don't ask the right questions or look at that data the right way, you may not realize the true severity of an issue, or you may be blind to it altogether. In the worst cases you may mislead yourself into futilely chasing what I call a "Performance Phantom"—an issue that appears to be a root cause, but in actuality is a symptom of a larger issue.

Let's consider a common case that illustrates why these matter. Businesses want to ensure that their end users are happy so they can maximize productivity, loyalty, profits, etc. To that end they will often ask for KPIs to help them determine if key parts of an application are meeting SLAs, asking questions like *"What's the response time of MyAccount.aspx?"*

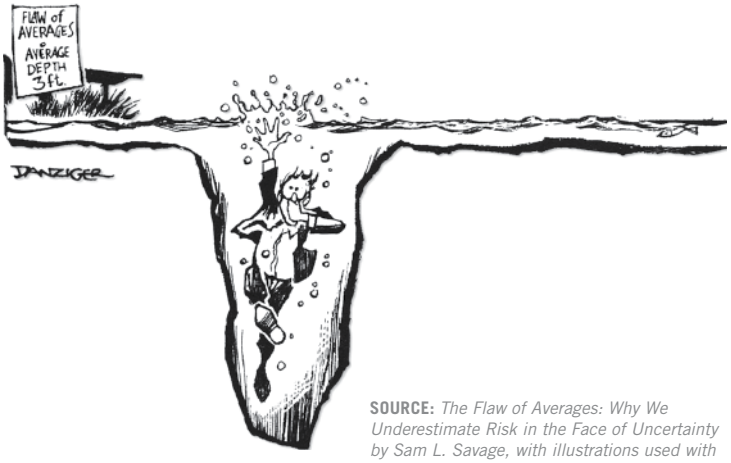
The answer is often provided by an APM tool in a report or business dashboard with a singular value like:

The value above is from a sample dataset I will use for the remainder of this article. That value represents the average of 10,000 calls to *MyAccount.aspx* over a 4-hour period. Here's a snippet of a log showing those calls:

#	Start Time	URL	Time
1	00:00:00.000	MyAccount.aspx	3.277
2	00:00:02.413	MyAccount.aspx	3.875
3	00:00:04.040	MyAccount.aspx	2.825
4	00:00:06.520	MyAccount.aspx	69.954
5	00:00:08.028	MyAccount.aspx	35.047
6	00:00:10.382	MyAccount.aspx	4.194
7	00:00:12.222	MyAccount.aspx	5.171
8	00:00:14.074	MyAccount.aspx	4.679
9	00:00:15.500	MyAccount.aspx	3.795
10	00:00:17.119	MyAccount.aspx	5.159
...			
9,993	04:02:05.774	MyAccount.aspx	3.778
9,994	04:02:07.170	MyAccount.aspx	34.376
9,995	04:02:08.433	MyAccount.aspx	24.971
9,996	04:02:10.480	MyAccount.aspx	4.004
9,997	04:02:12.082	MyAccount.aspx	3.552
9,998	04:02:14.869	MyAccount.aspx	10.735
9,999	04:02:17.336	MyAccount.aspx	3.686
10,000	04:02:19.266	MyAccount.aspx	5.200

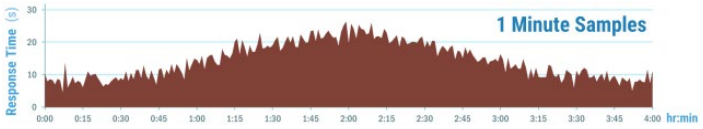
If you really think about it, you'll realize how ludicrous the initial question was in the first place. A singular value will never relate the range of experience for all of those users. There are actually over 10,000 answers to the question: one for each individual call, and others for subsets of calls like user type, location, etc. If you really want to know if ALL of your users are happy with ALL of their interactions with your application, you have to consider each user interaction as individually as possible, and beware the **Flaw of Averages.**

In this classic example, a statistician tried to cross a river that was, on average, 3 feet deep. Unfortunately, since he could not swim, the maximum value of his life became zero:

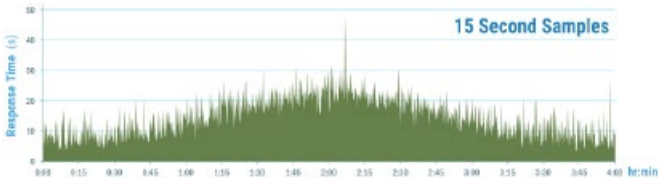


SOURCE: *The Flaw of Averages: Why We Underestimate Risk in the Face of Uncertainty by Sam L. Savage, with illustrations used with permission by Jeff Danziger – flawofaverages.com*

A common alternative to the singular value is a time series chart. Here we see the same data trended for the 4-hour period, revealing that it was much faster in the beginning and end, with a worst-case response time in the middle of 25 seconds:

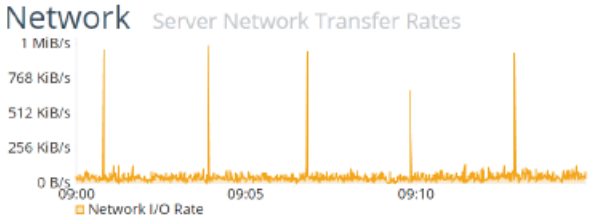


Although this 1-minute granularity chart has 240x more information than the singular answer, it still suffers from the Flaw of Averages. The same data at 15-second granularity tells a different story:

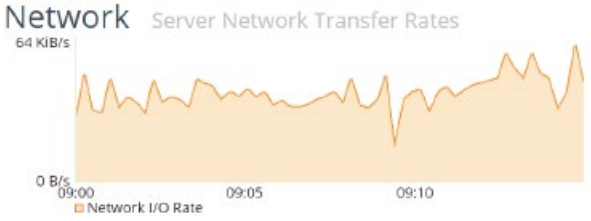


We see much more volatility in response times, with a worst case almost double what the previous chart suggested. As granularity improves, you'll get a more realistic understanding of the experience of your end users. If you consider that SLAs may be less than a second, you'll realize how inadequate even 15-second granular data is.

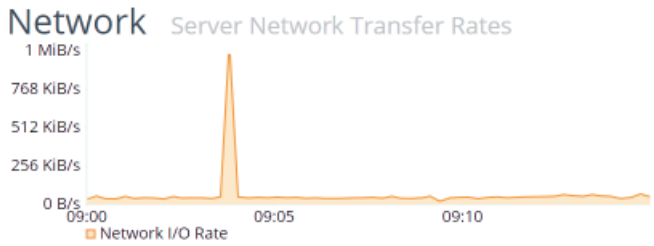
Many apps are plagued by periodic saturation of resources that only last for a second, but cause significant increases in response time during that second. Here's an example with five 1-second spikes in a 15-minute period:



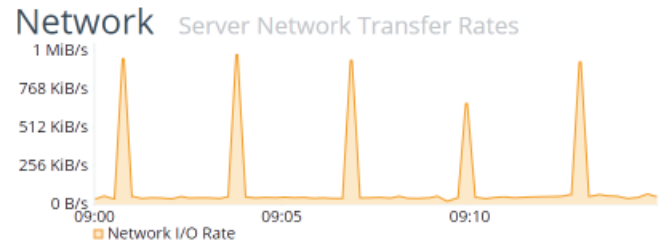
An APM tool will only catch the spikes if it coincidentally samples during the exact seconds the spikes occur in. If your tool samples every 15 seconds, you might be surprised at how low the odds are that it will catch those spikes. Statistically there's a 71% chance it won't see ANY of the spikes, so you wouldn't even know this behavior was occurring:



There's a 7% chance it will catch just 1 of the 5 spikes:



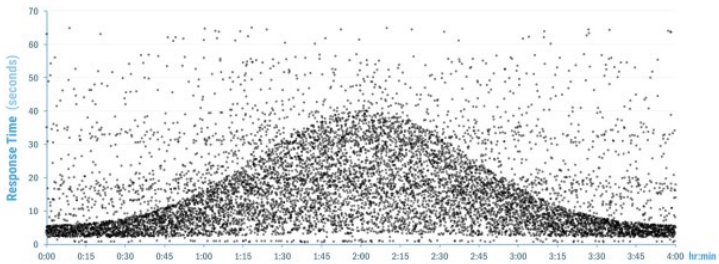
Here's where your jaw will drop: There is a 1 in 759,375 chance (0.0001%) that it will catch all 5 spikes!



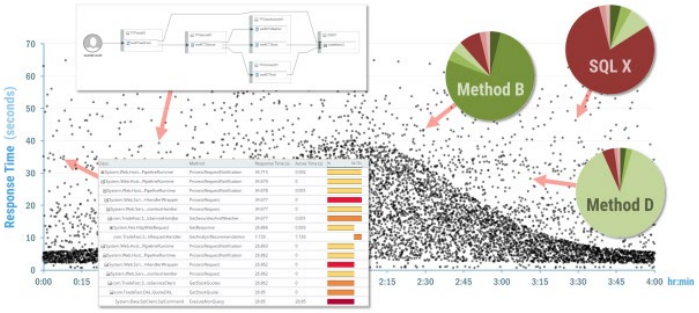
So even at a *seemingly* good 15-second granularity, there's almost no chance at all that you'd have an accurate understanding of this behavior. I often see coarse data granularity as the reason why organizations--even those with highly rated APM tools--are blind to these sorts of recurring issues. They don't even know the problem exists, so they don't even attempt to solve it.

Now let's get back to the previous *MyAccounts.aspx* example. I could show you how much better a 1-second sampled chart tells the story, but even that wouldn't tell the *full* story. Other statistics like min/max, percentiles, standard deviation, and histograms help reveal anomalies, but they too only paint a partial picture. The best option is to **not sample at all.** Capture everything. All transactions, all the time, down to the method & SQL level. With the right APM tool this is possible even in production under heavy loads.

But capturing that data is only half the battle, as you need to store that data in full detail and be able to nimbly analyze hundreds of thousands of transactions at once. Your APM tool needs to leverage Big Data to make sense of all that information and tell the complete story accurately. Here's our sample dataset as only Big Data can show it:

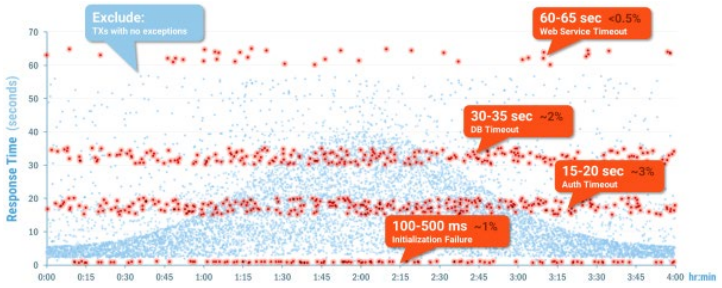


For 10,000 transactions you have 10,000 different answers to the initial question *"What's the response time of MyAccount.aspx?"*—this is a much different story than the simple line charts suggested. But even more importantly, you have the details as to why each of those 10,000 behaved the way they did:



For each *individual* transaction you can see what method or SQL is causing the majority of the delay. You can see multi-tier maps for each transaction independently, so if there is a certain pathway that's causing issues, it won't be hidden by a single one-size-fits-none application map. You can even get call-tree details for each transaction to provide the details developers need to solve the issue.

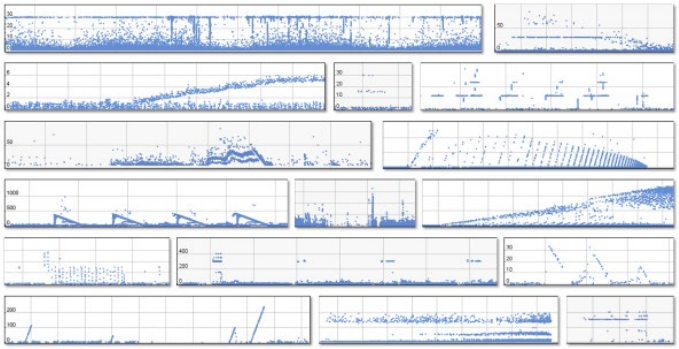
Big Data will allow you to filter out transactions with particular characteristics, and reveal clusters of behavior masked by aggregated line charts. By filtering out all the transactions that didn't contain exceptions, we see that there are 4 different sub-behaviors of the application:



The top 3 *bands* of response time are due to timeouts for 3 different dependencies: a Web Service, a Database, and the Authentication service. The bottom band is due to a catastrophic failure where the transactions failed before they even initialized, resulting in ultra-fast response times which would never be caught by sampling just the *slowest* transactions.

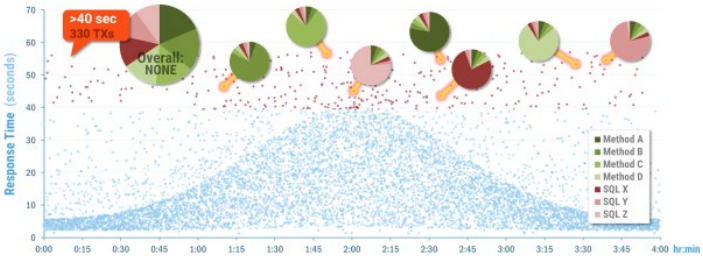
Just as there isn't a singular answer to the question *"What's the response time?"* there isn't a singular answer to *"Why is it slow?"*—which translates to *"What are the different things we need to fix to improve performance?"*

Since I've been using a sample dataset, I want to prove that this concept isn't just academic. Here are some real-world examples where Big Data revealed patterns of behavior that were previously hidden by other tools:



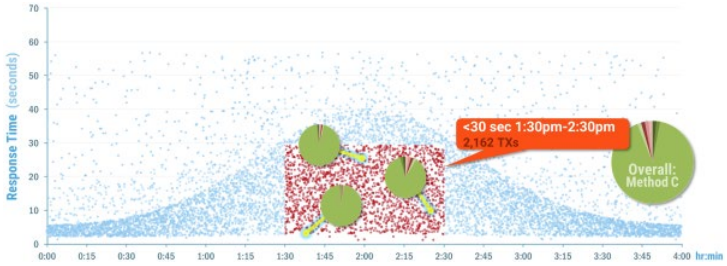
The horizontal lines represent timeouts. The vertical lines are microbursts after stalls. The diagonal lines are client or server side queuing depending on the direction. The ramps-beneath-ramps are compound issues. You will NEVER see patterns like these in line charts. If you've never seen patterns like these, then you've never seen an *accurate* representation of your data.

As I mentioned earlier, even with the best data, if you ask the wrong questions you'll get the wrong answer. It's very common for troubleshooters to ask *"Why are the slowest transactions slow?"* but quite often this isn't the reason why the application is slow *overall*. In our sample dataset, Big Data reveals that there isn't a *consistent* reason for slowness across the slowest transactions:



This is a clear indication of the "Performance Phantoms" I referred to earlier, where some environmental issue like garbage collection or hypervisor over-commitment causes delays in whatever pieces of code happen to be running at the same time. Trying to optimize these methods will waste countless hours with little reward. You can never solve a root cause by trying to fix the symptom.

The best way to make overarching improvements to application performance is to leverage Big Data to identify the overarching reasons for delay. Here we see a consistent reason for delay in this subset of transactions:



Method C is the overall largest contributor to delays, and individual transactions confirm that consistent root cause. Focusing on this one method will yield the greatest benefit for the least effort.

I worked with a large bank who had a major performance issue in one of their key pages. Using legacy APM tools, they identified the slowest methods in the slowest transactions, but even after optimizing them, performance issues persisted. They repeated this process for months to no avail. Once they leveraged Big Data APM, in one day they were able to identify a little method that on average took 53ms, but ran so frequently it wound up being the largest contributor to delay. Optimizing that single method improved the response time of 7 Million transactions per day by 95%, and reduced total processing time by 2,000 hours per day. This is not a corner case. Issues of this magnitude are very common—and hidden in plain sight—but with the right data and methodology they are easily revealed.

I challenge you to scrutinize your current tools to make sure they're capturing the right data in the right way. If your data is blind to an issue, or misrepresents it, then you'll fail before you even begin. Once you have the right data, I encourage you to step out of your comfort zone of just looking at averages and line charts, and harness the power that Big Data provides. Sift through the noise, identify the patterns in your behavior, and learn to distinguish inconsistent symptoms from consistent root causes. Be the hero that identifies the one little thing that yields hours of improvement for millions of users.

JON C. HODGSON is an APM subject matter expert for Riverbed Technology who has helped hundreds of organizations around the world optimize the reliability and performance of their mission-critical applications. When he's not obsessing about how to make things perform faster, he enjoys digging things up with his tractor at his home in Missouri.

IT is about decisions

BMC TrueSight transforms IT by turning data into actionable insights while eliminating the noise of traditional IT management tools

Bring IT to Life with TrueSight

Unprecedented Visibility - Noise = Actionable Insight

bmc

TrueSight performance and analytics

bmc.com/truesight

The New School of Applications and Infrastructure

The evolution of the performance and analytics space for IT operations has taken some really interesting turns over the last 12 months. As the enterprise begins to adopt modern, elastic technologies and practices, the traditional "stack" that we've all been tasked with supporting has changed. With that change comes a necessary evolution in both the tooling and the methodology of dealing with these new mutable environments. Big Data, Cloud, DevOps, web-scale, cloud-scale – these are no longer just buzz words bandied about by the analyst community and employed by nimble little startups. They are being adopted, in production, into many of BMC's largest customers.

Perhaps the most important consequence brought on by these new trends in IT is **application centricity**. The bottom line of every truly digital enterprise is customer experience. Whether it's B2B or B2C, the way in which the end-user interacts with the business service is the ultimate measure of success or failure for the business. To support this customer oriented stance, the once disparate IT Ops, application and development teams are beginning to overlap. They require the ability to access both deep and wide views of the new, modern stack. This is why it's so critical to provide end-to-end insight into both the application and the infrastructure that supports it.

The challenge in addressing both APM and infrastructure monitoring needs in one solution for the increasingly complex modern stack is **noise**. The more complex and fluid applications and infrastructure become, the louder they get.

So how do you parse out the signal from the noise?

At BMC we've long been known as a "MoM" (manager of managers) in the IT operations space. We collect, correlate and alert on everything that's gone awry in your infrastructure. This is still true – but we've evolved the concept to support the complexity and variety of the modern stack. We've added APM insight into the equation and tied infrastructure and applications together with Operational Analytics in the TrueSight portfolio. This ensures you see the right problems in the context of the applications and underlying infrastructure components that are affected – without the distraction of false or symptomatic alerts.

WRITTEN BY MARK RIVINGTON

SENIOR DIRECTOR OF STRATEGY, **BMC TRUESIGHT**

PARTNER SPOTLIGHT

TrueSight Performance and Analytics BY BMC SOFTWARE

BMC TrueSight applies behavioral analytics to a converged view of applications and infrastructure for less noise and more insights

CATEGORY

Application and Infrastructure Monitoring

NEW RELEASES

Continuous Delivery

OPEN SOURCE?

Some components

CASE STUDY

Northwestern University Information Technology (NUIT) delivers computing services to nearly 25,000 students, faculty and staff across three campuses in Illinois including over 300+ application owners with 99% system uptime. NUIT uses TrueSight App Visibility Manager to power a web-based capacity and availability service monitoring dashboard that keeps IT teams informed of potential performance problems and enable thousands of university end-users to check the status of their web applications. Putting the right data in the hands of the operations team and application owners has saved hours of staff time and improved collaboration on issues.

STRENGTHS

- Monitor performance of applications and infrastructure together
- Dynamically view application topology
- Reduce event noise with operational IT analytics
- Get proactive with log analytics and root cause analysis
- Understand the service impact of events
- Prioritize and route incidents automatically
- Automate event remediation and self-healing
- Real-time monitoring of modern stack

NOTABLE CUSTOMERS

- SEI
- Northwestern Univ.
- Harmony Information Systems
- Société Générale
- InContact
- Lockheed Martin
- HealthMEDX
- IKEA

BLOG **bmc.com/opsblog** | TWITTER **@truesightpulse** | WEBSITE **bmc.com/truesight**

Know When (and When Not) to Blame Your Network

BY NICK KEPHART
SR. DIRECTOR OF PRODUCT MARKETING, **THOUSANDEYES**

QUICK VIEW

01
In distributed, cloud-based environments, it's equally important to understand both application and network performance.

02
Active monitoring, often used for website performance, can also provide you with insights into cloud provider networks.

03
Active monitoring can provide you a stack trace for your network, showing the performance of each network that your traffic traverses.

04
Consider adding key network connectivity and service metrics to your arsenal in order to get ahead of cloud outages.

Most of us are familiar with APM tools. They are broadly deployed; a recent Gartner survey showed APM being used to monitor more than 25% of applications in 40% of enterprises. APM instruments our code to make it easier to troubleshoot application-related issues. You can trace transactions, check SQL performance, and track error rates. But sometimes your users complain about performance and your APM shows everything is swell. That's when it's typically time to push the ticket over to the network team.

And not only is it the case that APM tools tend to be pretty opaque as to whether the network is at fault, but they also aren't always well suited for the type of end-user and cloud environments that you are increasingly seeing. The same Gartner survey also found that a majority of APM users believe that current APM solutions are challenged by the prevalence of cloud-hosted applications and the Internet of Things (IoT).

So in this more distributed environment, where it's already difficult to pull apart whether a performance issue is application or network related, what do you do? The reality is that some of the same techniques you likely already use to monitor application experience can also help with network experience. Getting better visibility into application delivery may not be as hard as it seems.

SEEING APPLICATION AND NETWORK AS ONE

Active (or synthetic) monitoring is most associated with understanding page load and user transaction timings. But it can also help you understand when an issue is network-related, and when it isn't, so you can be confident when assigning your team to look into a problem.

Active monitoring can give you insight into the performance of networks and infrastructure, as well as your application. And, in addition to your data center, it works in cloud environments and across the Internet, where many of your applications are hosted and where your customers are clicking away on your app. That way, you can see network and application data lined up right next to each other; and not just some average latencies, but in-depth information about how each portion of the network, between your app and your users, is performing. Most active monitoring tools will give you perspectives both from Internet locations and from within your own infrastructure, so you can use this technique for customer-facing or internal-facing applications.

HOW IT WORKS

So how does it work? It starts with loading a page in a browser and monitoring user timing. Each object on the page is loaded, measuring load time (DNS, blocked, wait, response, etc.) wire size, and uncompressed size. These page loads can be linked together as entire user transactions with button clicks, form fills, and more. This can be particularly useful for JavaScript-heavy pages where user interactions

trigger additional network requests and object loads. Active monitoring can collect much of the same data you get in Firefox or Chrome developer tools en masse—at large scale—and crunch all of the results into easy-to-digest views.

In addition to all of this browser-level information, active monitoring can also send packets across the network that are specifically instrumented to give you a clear understanding of performance. By synchronizing it with browser-level data, you can get a ton of useful data about how your app is being accessed, not just loaded.

A STACK TRACE FOR YOUR NETWORK CONNECTIVITY

This isn't just round-trip loss and latency measurements to the web server. You can measure loss and latency to each layer 3 hop between your users and your web server or CDN edge. Plus, you can measure both the forward and reverse path, so you can understand performance of both downloads and uploads between your app and users. That's huge!

Why? Well, within your app you can solve problems a lot faster when you have an exact stack trace, right? It's the same with the network. By knowing which points in the network are having issues, you can much more quickly triage the issue and even tell who's responsible. All of a sudden you can answer questions such as:

- Is your cloud provider or hosted data center proving the throughput you expect?
- Is your IaaS provider having an outage, either regionally or more widespread?
- Is your CDN correctly caching and serving your content from an optimal edge location?
- Is your DNS service efficiently serving up DNS queries to your users?

As any DevOps Engineer knows, a lot can go wrong in a lot of places and exhibit strange behavior. For example, traffic between countries in Asia often peers in the United States or Singapore given the congested links with China, Vietnam, and the Philippines. The same thing happens in Latin America, Africa, and the Middle East. Adjusting routing or other network configurations can dramatically speed up app delivery in these cases.

Having the equivalent of a stack trace for your network will make it possible to detect issues you may not know exist and to fix problems fast as they arise.

KEY APPLICATION DELIVERY METRICS

So what will you do with this sort of data? First, you can start collecting key metrics about the delivery of your application that you may not currently have at your disposal. Here are some of the most important metrics to keep an eye on:

APP PERFORMANCE

- **Page Load and Transaction Time:** A standard metric in many APM tools, this can provide a good performance baseline.
- **Object Size (wire and uncompressed):** The size of objects on the wire can vary widely and is important to your app's sensitivity to throughput constraints.
- **Object Errors and Load Time:** Most apps and webpages have objects coming from a variety of third-party locations and CDNs. Understand whether availability of one object is causing part of your app to fail.
- **Web/App Server Availability and Response Time:** Most likely a metric you're already tracking, but a key one to correlate with network connectivity metrics to understand outages.

NETWORK CONNECTIVITY

- **Loss per Interface:** By tracking loss per interface, you can easily correlate network connectivity issues with specific service providers and devices.
- **Latency per Link:** With a link-level view, you can understand which portion of your CDN, ISP, or data center networks are congested or faulty.
- **Throughput per Location:** Understanding throughput by ISP, region, and city can inform decisions about how fast bulky objects can be loaded by users.

NETWORK SERVICES

- **CDN Latency:** Measure performance from users to edge locations as well as your origin to CDN ingestion servers.
- **DNS Availability and Response Time:** It doesn't go wrong often, but when it does, you're hosed. Keep an eye on your DNS service provider.
- **Routing Path Changes:** Keeping a pulse on routing changes can ensure that you know if there is network instability or suboptimal routing.

ADDING ACTIVE MONITORING TO YOUR ARSENAL

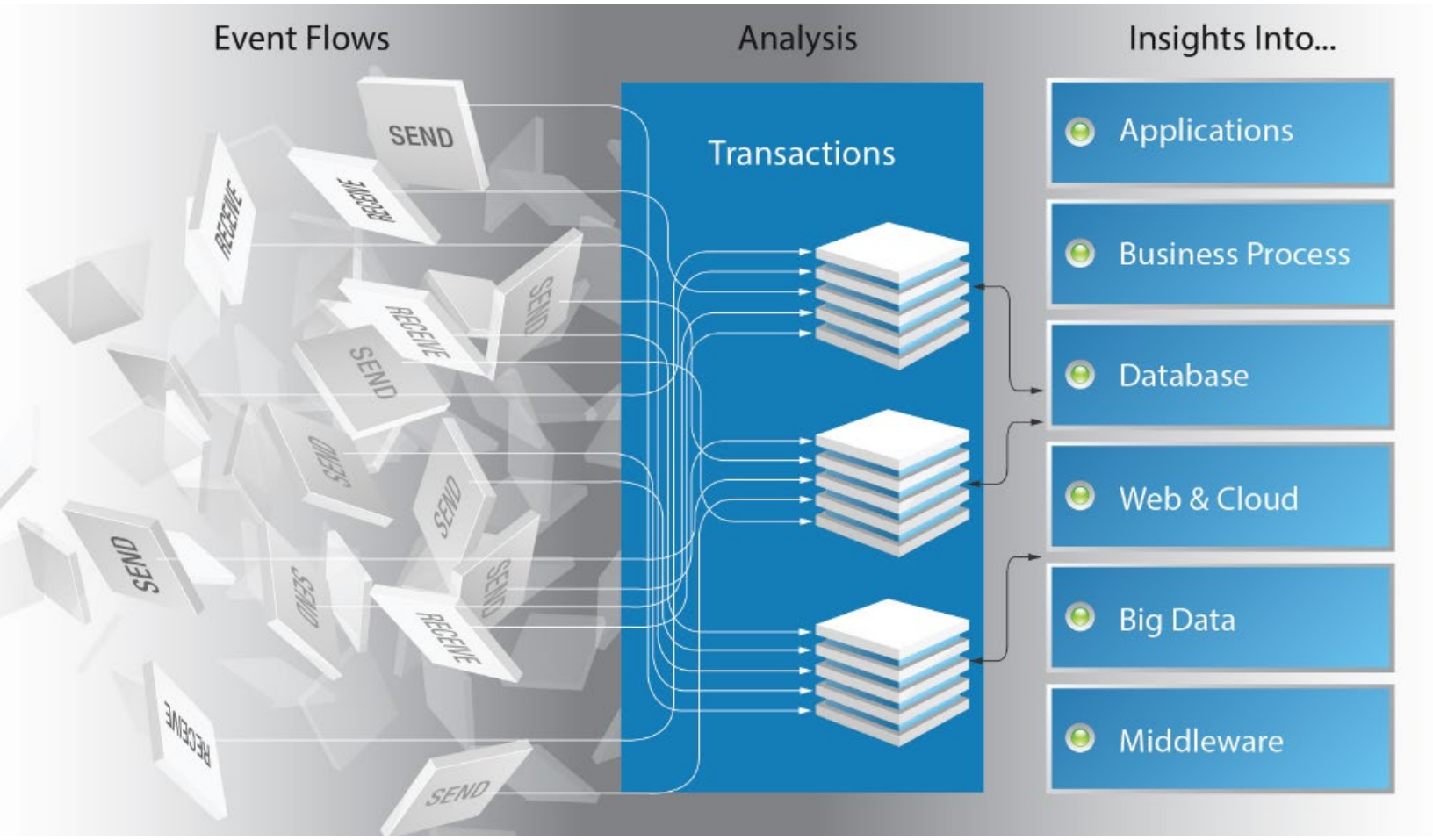
Active monitoring can save you from huge headaches. One major payment processor that I've worked with spent an entire holiday weekend, with multiple senior engineers trying to track down what they thought was a database transaction fault. Another team had just started deploying active monitoring in their environment, and upon reviewing the data, was able to track the problem to a routing issue that was causing unstable network connectivity. Upon seeing the data, the application development team became instant converts to adding active monitoring into the runbook for issue resolution.

As your applications are increasingly relying on IaaS, microservices, and APIs from far-flung parts of the Internet, your app is more reliant on the network than ever. That means in order to have a complete view of application experience, you should be adding active network monitoring to your application troubleshooting arsenal. With this data, your development team can avoid dead ends and be more confident the next time you need to ask the network guys to dive into an issue.

NICK KEPHART leads Product Marketing at ThousandEyes, which develops Network Intelligence software, where he reports on Internet health and digs into the causes of outages that impact important online services. Prior to ThousandEyes, Nick worked to promote new approaches to cloud application architectures and automation while at cloud management firm RightScale.

Deliver Great App Performance

with **AutoPilot® Insight**



Performance Monitoring
+
Powerful Analytics
=
Satisfied Customers

- ✓ Analyze logs, metrics, users, and perform total end-to-end transaction tracking
- ✓ Isolate and diagnose problems before they affect customers
- ✓ Integrate all infrastructure and performance monitoring on a single pane of glass
- ✓ Reduce support costs

NASTEL
Driving Business Transaction Performance ®

Begin your free product trial and business analysis today at **www.nastel.com/nastelzone**

The ABCs of Performance Monitoring

YOUR WORLD IS COMPLICATED

IT environments are much the same as they were 20+ years ago—except for everything. Along with OS, networks, middleware and business apps, today's IT pros must grasp the complexity of Big Data, Web- and cloud-based technology, mobile apps, business processes, and whatever is coming out next week. And now it all needs to be in real-time along with sophisticated analytics.

ASSUME YOUR DELOREAN IS PERMANENTLY IN THE SHOP

Forget about going back in a DeLorean to an earlier, simpler time. Concentrate on ***what you can do now*** to make your IT environments more understandable, controllable—and ultimately, more reliable. Start by:

- Determining the probable root causes of IT problems before they affect business service delivery
- Analyzing all of your related performance and infrastructure monitoring data together
- Understanding business transaction flows, the lifeblood of any business

Application performance monitoring (APM) focuses on understanding the health and throughput of apps within one or more system environments, from simple to the most complex enterprise settings. To paraphrase research compiled by Gartner over the last six years, there are five facets, or dimensions, of APM: Application topology discovery, Transaction profiling, Application component deep dive, IT operations analytics (ITOA), and End-user experience monitoring (EUEM).

Real-time visibility on application performance not only lets IT pros know how well their business services are performing at any given moment, but also provides the foundation for continuous optimization. If you pick the right tools you'll be able to focus on the right issues, at the right times. And being able to zero in with speed and precision as problems arise means bottlenecks are eliminated quickly and your company's reputation with app users is protected, which is the ultimate name of the game for APM.

Click here to see the rest of the article.

WRITTEN BY CHARLEY RICH

VP-PRODUCT MANAGEMENT, **NASTEL TECHNOLOGIES**

PARTNER SPOTLIGHT

AutoPilot Insight

BY NASTEL TECHNOLOGIES

NASTEL
Driving Business Transaction Performance

Nastel Technologies provides a unified suite of analytic and monitoring tools for end-to-end transaction tracking, logs, end-users, and apps

CATEGORY
Analytics & APM

NEW RELEASES
Every quarter

OPEN SOURCE?
No

CASE STUDY

Sky Mexico, a satellite television delivery service, needed a product to span its UNIX and Windows infrastructure and monitor the health of ERP, CRM, billing, IVR, provisioning, and other business and IT transactions. The lack of which was causing:

- Increasing backlog of service requests
- Failure to meet Service Level Agreements (SLAs)
- Loss or delay in order fulfillment
- On-going damage to company reputation and competitive stance

With AutoPilot Insight, Sky:

- Achieved a reduction of help desk tickets and costly Tier 3 support of 30 and 70 percent, respectively
- Slashed MTTR by 45 percent via advanced complex event processing (CEP)
- Could answer critical questions like: "What's my order status?"
- Turned an IT team from reactive to proactive

STRENGTHS

- Unified control of APM, Analytics, and Messaging Middleware/Admin functions
- Advanced Complex Event Processing engine enables elimination of false problem alarms
- Transaction, log, application flow, and end-user analysis, plus business process mapping
- Auto-discovery of apps, system components, transaction flow, and app dependencies
- Root-cause analysis; drilldown to offending components, SQL queries, and method calls

NOTABLE CUSTOMERS

- CitiBank
- BestBuy
- Fiserve
- BNY Mellon
- UnitedHealth Group
- Sky
- Dell
- NY Power Authority

BLOG **www.nastel.com/blog** | TWITTER **@nastel** | WEBSITE **www.nastel.com**

Performance Patterns in Microservices-Based Integrations

BY ROHIT DHALL
ENTERPRISE ARCHITECT, ENGINEERING AND R&D SERVICES, **HCL TECHNOLOGIES**

QUICK VIEW

01 Understand how integration with multiple systems poses potential performance issues.

02 Learn what performance patterns are and how these can help you avoid common potential performance issues.

03 Understand five different performance patterns and how they work.

04 Understand the importance of asynchronous communication/integration.

Systems based on microservices architectures are becoming more and more popular in IT environments. The rise of microservices increases the challenges associated with systems with multiple integrations. Almost all applications that perform anything useful for a given business need to be integrated with one or more applications. With microservices-based architecture, where a number of services are broken down based on the services or functionality offered, the number of integration points or touch points also increases massively. This can impact the system as a whole, since now overall system performance is also dependent upon external factors (the performance of an external system and its behavior).

PERFORMANCE PATTERNS AND THEIR BENEFITS

The concept of Design Patterns are well documented and understood in the software development world. These design patterns generally describe a reusable solution to a commonly occurring problem. Using design patterns can ensure good architectural design, but these alone are not enough to address performance challenges.

This is where performance patterns come into play. When implemented correctly, these can really help build a scalable solution.

PERFORMANCE CHALLENGES WITH RESPECT TO INTEGRATED SYSTEMS

Distributed computing has its own challenges, and all of these challenges are not only well documented, but are experienced by professionals working on distributed systems almost daily. While connecting to other microservices (within the same bounded context or of some remote, external system), many things can go wrong. Services and systems (both internal and external) you connect to may be slow or down. If your application is not designed to handle this scenario gracefully, it can have an adverse impact on the performance and stability of the overall application.

PERFORMANCE PATTERNS

In this section we will talk about some approaches and design decisions that can help us achieve better performance, resilience, and overall stability with respect to integration challenges in a microservices-based environment.

THROTTLING

Throttling is one technique that can be used to prevent any misbehaving or rogue application from overloading or bringing down our application by sending more requests than what our application can handle.

One simple way to implement throttling is by providing a fixed number of connections to individual applications. For example, there are two vendors who call our microservice to deduct money from one account. If one vendor has a big application (like Amazon), then it is likely to consume our service more often than a vendor which has a small user base. We can provide these two vendors two separate and dedicated "entry points," with dedicated throttled connection limits. This way, a large number of requests coming from Amazon will not hamper requests coming from a second vendor. Moreover, we can throttle individual partners so that none can send requests at a rate faster than what we can process.

Generally, synchronous requests from external services/systems are throttled at the load balancer/HTTP server or another such entry point.

TIMEOUTS

If a microservice is responding slowly, it can cause our application to take longer to complete a request. Application threads now remain busy for a longer duration. This can have a cascading

impact on our application, resulting in the application/server becoming totally choked/unresponsive.

Most libraries, APIs, frameworks, and servers provide configurable settings for different kinds of timeouts. You may need to set timeouts for read requests, write requests, wait timeouts, connection pool wait timeouts, keep alive timeouts, and so on. Values of these timeouts should be determined only by proper performance testing, SLA validation, etc.

DEDICATED THREAD POOLS/BULKHEADS

Consider a scenario where, in your application flow, you need to connect to five different microservices using REST over HTTP. You are also using a library to use a common thread pool for maintaining these connections. If, for some reason, one of the five services starts responding slowly, then all your pool members will be exhausted waiting for the response from this service. To minimize the impact, it is always a good practice to have a dedicated pool for each individual service. This can minimize the impact caused by a misbehaving service, allowing your application to continue with other parts of the execution path.

This is commonly known as the bulkheads pattern. The following figure depicts a sample scenario of implementing a bulkhead. On the left side of the figure, microservice A—which is calling both microservice X and microservice Y—is using a single common pool to connect to these microservices. If either service X or service Y misbehaves, it could impact the overall behavior of the flow, since the connection pool is common. If a bulkhead is implemented instead (as shown in the right side of the figure), even if microservice X is misbehaving, only the pool for X will be impacted. The application can continue to offer functionality that depends on microservice Y.

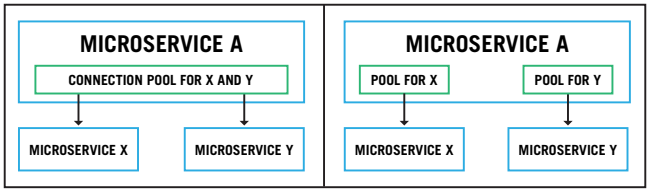


FIGURE 1: COMMON THREAD POOL VS. BULKHEADS

HOW BULKHEADS WORK

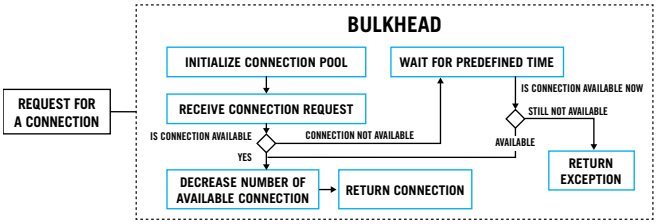


FIGURE 2: COMMON THREAD POOL VS. BULKHEADS

Critical Details

- Any application that needs to connect to a component will request a connection to that component.
- Connection to each of the components is controlled by the individual bulkhead.
- When a request for a new connection is made, the bulkhead will check if the connection to the requested component is available to serve the request.
- If the connection is available, it will allocate this connection to serve the request.
- In case no free connection is available, the bulkhead will wait for a pre-defined time interval.
- If any connection becomes available during this wait period, it will be allocated to serve the waiting request.

CIRCUIT BREAKERS

A Circuit Breaker is a design pattern, which is used to minimize the impact of any of the downstream being not accessible or down (due to planned or unplanned outages). Circuit breakers are used to check the availability of external systems/services, and in case these are down, applications can be prevented from sending requests to these external systems. This acts as a safety measure, on top of timeouts/bulkheads, where one may not want to even wait for the period specified by timeout. If a downstream system is down, it is of no use to wait for the TIMEOUT period for each request, and then getting a response of timeout exception.

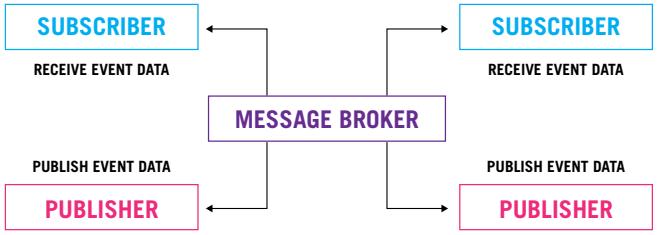
Circuit breakers can have built in logic to perform necessary health checks of external systems.

ASYNCHRONOUS INTEGRATION

Most performance issues related to integrations can be avoided by decoupling the communications between microservices. The asynchronous integration approach provides one such mechanism to achieve this decoupling. Take a look at the design of your microservices-based system, and give it a serious thought if you see point-to-point integration between two microservices.

Any standard message broker system can be used to provide publish-subscribe capabilities. Another way to achieve asynchronous integration is to use event-driven architecure.

The following figure shows a scenario, where decoupling between producers and receivers/subscribers is achieved with the use of a message broker.



CONCLUSION

In this article, we talked about some of the performance challenges we face while integrating microservices-based systems. It also presented some patterns that can be used to avoid these performance issues. We discussed throttling, timeout, bulkheads and circuit breaker patterns. Apart from these, an asynchronous integration approach is also discussed.

In a nutshell, asynchronous integration should be preferred, wherever possible. Other patterns can also be used in integration scenarios to avoid the ripple/cascading side effect of a misbehaving downstream system.

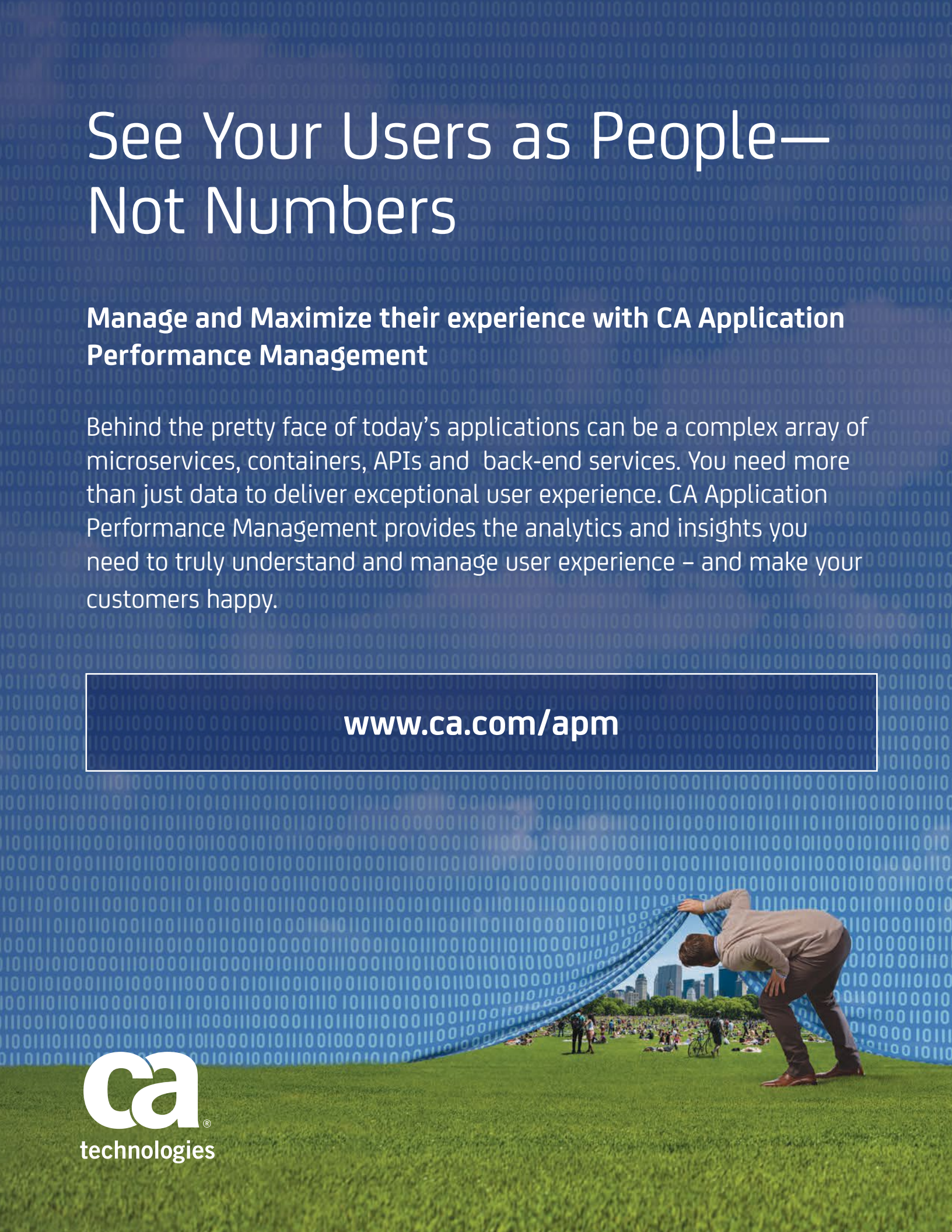
ROHIT DHALL is working as an Enterprise Architect with the Engineering and R&D Services division of HCL Technologies. He has more than 19 years of experience. He helps global clients build technical solutions to solve their complex business problems. His main area of expertise is architecting, designing, and implementing high-performance, fault-tolerant, and highly available solutions for leading Telco and BFSI organizations. He has worked with diverse technologies like Java/J2EE, client-server, P2P, DWH, SOA, Big Data, IoT, etc. He regularly writes white papers, articles and blogs and for various IT events, forums, and portals. He is also a coauthor of the IBM Redbook and Redpaper on 'ITCAM for WebSphere'.

See Your Users as People—Not Numbers

Manage and Maximize their experience with CA Application Performance Management

Behind the pretty face of today's applications can be a complex array of microservices, containers, APIs and back-end services. You need more than just data to deliver exceptional user experience. CA Application Performance Management provides the analytics and insights you need to truly understand and manage user experience – and make your customers happy.

www.ca.com/apm



ca technologies

Five Tips to Successfully Manage User Experience with Application Performance Management

When implementing an application performance monitoring strategy it can be tempting to just grab some tools and start using them. This can ultimately lead to choosing one or more disparate tools that are not integrated or holistic in their approach. Too many tools and too much data can actually lead to not enough insight into what is really going on with your apps or your users' experience. Here are five tips for success.

First, understand all of your customers. Monitor apps across mobile, web and wearables and include synthetic monitoring to find and fix problems even at times where you have no users. Leverage passive monitoring when security or other concerns prohibit direct end-user monitoring.

Second, make sure you can follow transactions from front-end to back-end. Transactions can cover a lot of ground from your app to APIs, security layers, middleware all the way to the back-end. Make sure your monitoring covers the same ground.

Third, get continuous feedback across DevOps by integrating monitoring across all parts of the SDLC. This is as much cultural as it is technical. Collaboration across Dev and Ops is critical to delivering great user experiences.

Fourth, understand how changes impact performance. Being able to roll back time to see what changed before an issue helps you find "patient zero" and resolve problems faster.

> TOO MANY TOOLS AND TOO MUCH DATA CAN ACTUALLY LEAD TO NOT ENOUGH INSIGHT INTO WHAT IS REALLY GOING ON WITH YOUR APPS

Finally, simplify the complex! Modern apps can have a lot going on under the covers. Views and perspectives that remove layers of complexity help you see what is important more clearly, without a distracting data deluge.

Consider these tips and you'll be more successful in managing the performance of your applications - and help keep your customers happy.

WRITTEN BY DAVID HARDMAN

DIRECTOR, PRODUCT MARKETING, **CA TECHNOLOGIES**

PARTNER SPOTLIGHT

CA Application Performance Management

BY CA TECHNOLOGIES

ca technologies

CA APM speeds and simplifies the triage of application performance issues, helping you deliver loyalty-building user experiences

CATEGORY
APM

NEW RELEASES
Quarterly

OPEN SOURCE?
No

CASE STUDY

Orange has been offering communication services for more than 20 years. Today it provides mobile and landline telecommunications and broadband services to 244 million retail and business customers around the globe. An excellent customer experience is a strategic priority for Orange. But the performance of some applications on Orange.com was not up to par. CA APM plays a critical role in ensuring the overall quality of Orange's applications. It helps Orange assess the risk associated with an application prior to its release into a given environment. Orange can deliver the excellent online experience expected by today's increasingly connected customers with better reliability, availability and faster response times.

STRENGTHS

- **Easy**—Simplify the triage process through role based views and integrated timeline
- **Proactive**—Recognize problems as they develop and focus on the most critical issues
- **Intelligent**—Detect and monitor application processes and transactions automatically
- **Collaborative**—Enable better communication between Dev and Ops to resolve problems faster

NOTABLE CUSTOMERS

- Lexmark
- Vodafone
- Itau Unibanco
- Blue Cross Blue Shield of Tennessee
- U.S. Cellular
- Innovapost
- Produban
- Expeditors

BLOG bit.ly/ca-apm | **TWITTER** @cainc | **WEBSITE** ca.com/apm

Working in Parallel:

ON THE COMPLICATIONS OF PARALLEL ALGORITHM DESIGN

BY ALAN HOHN
SOFTWARE ARCHITECT, **LOCKHEED MARTIN MISSION SYSTEMS AND TRAINING**

QUICK VIEW

01
In parallel programming, "work" is all the steps you have to do, and "depth" is how much work you can do at once. Both use Big O notation.

02
Available parallelism is work divided by depth.

03
Sometimes you have to waste work to improve parallelism.

04
Sometimes the algorithm with the best available parallelism is not the best algorithm.

05
After you find a good parallel algorithm, the next challenge is tuning it to run efficiently on real hardware.

Moving from a sequential to a parallel implementation of an algorithm usually means that something changes, and it may mean a completely different approach.

SIMPLE PERFORMANCE

We talk about functions in terms of the "order" (typically called Big O). This is how it behaves as the input size changes, without worrying too much about specifics of how long it takes to actually do the work.

For example, if we have data stored in an unsorted list structure, and we need to find out if a particular value is in the list, we must check each item in the list until we find the item or reach the end. In Big O notation we call this `O(n)`, indicating that as the length `n` of the list grows, we should expect the time it takes to search it to increase in a linear way.

Note that we don't care how long it takes to step through and look at each element, and we don't care that an early match is very fast. We only care about the general relationship between the size of the list and the time it takes to run. In this case, if the list gets twice as long, the average run time will get about twice as long.

Similarly, if we had an unsorted list, and we were searching for duplicated elements, we would call this `O(n^2)`, because we are going to have to do `n` searches through the list, each of which we already said is `O(n)`. Regular math works here, and `O(n)` times `O(n)` equals `O(n^2)`. Again, the details don't matter; we just care that if the list gets three times as long, average run time will be about nine times as long.

WORK AND DEPTH

When we move from sequential to parallel, we still think about Big O, but also about doing multiple things at the same time. For example, in searching an unordered list, while we have to step through the whole list, every single comparison is independent of every other, so if we had that many processors we could do them all at once.

As a result, instead of having a single Big O value, we use the terms "work" and "depth." Work we saw earlier; it is how the run time grows as the input size grows. Depth also uses Big O notation, but it uses it to express how easy it is to run in parallel.

We use the term "depth" because we are thinking in terms of "divide and conquer." We expect to have a recursive

function that hands off smaller and smaller pieces of the problem to new versions of itself. The flatter (shallower) the recursion, the better, because it means we can spread out across multiple processes more easily. In our search of an unordered list, the depth is `O(1)`, or "constant time." No matter how many extra items there are in the list, we can, in theory, break it up into that number of pieces.

In our unsorted list duplicate search, we compare each item in the list with every other item. This is no problem, for we just create `n^2` separate tasks, each with a different "left" and "right" index for the comparison, and we can do all the comparisons in one step. So the depth is still `O(1)`.

At this point, alarm bells will be ringing about how feasible this is, but I'm not quite ready to let the real world intrude yet.

AVAILABLE PARALLELISM

Putting work and depth together, we can define "available parallelism" (where bigger is better):

```
Available Parallelism = Work / Depth
```

With our search through an unsorted list, the work was `O(n)` and the depth was `O(1)`, giving an available parallelism of `O(n)`. This means that as the size of the input increases, the amount of work increases linearly, but our ability to do it in parallel also increases linearly. So as long as we have more processors the problem will take about the same amount of time (ignoring for a moment the overhead of splitting the work).

In a marginally more realistic example, let's say that instead of just identifying duplicates, we wanted to count the number of duplicates for each duplicate we find. Now, instead of just comparing each item in the list to every other item, we also need to keep track of how many matches we've found. So we can't split up the comparisons completely. Let's take a simple approach. We will split up the "left" side of the comparison, then just iterate over the list. This way we count the number of matches in parallel for each item in the list. Of course, this is a very poor approach, because we are finding the same duplicates many times, which is a lot of wasted work.

For this example, while the work is still `O(n^2)`, the depth is now `O(n)`. This means our available parallelism is `O(n)`. This is still quite good, because we still see linear speedup from adding more processors.

Of course, it would be nice to avoid that wasted work. Those experienced with map and reduce may have noticed that a map can emit a value for each item, then a reducer can add them up. In fact, this is Hadoop's WordCount example. The work in this case is `O(n)`, and if the reducer is written correctly the depth is `O(log n)`.

Our available parallelism is `O(n / log n)`, which is slightly less than linear.

Note that while the work is much worse in the first example, because of all the wasted comparisons, it has slightly better available parallelism than the map/reduce example, because it fully preserves the independence of all the comparisons. That is not enough reason to choose it, but it does illustrate an important rule in parallel programming, which is that sometimes it is necessary to waste work in order to improve parallelism.

THE REAL WORLD WILL NOT STOP HASSLING ME

So far, making a good parallel algorithm has meant trying to increase our available parallelism, because then we can just throw more hardware at the problem to get it to run faster. Unfortunately, while that can be true, it isn't the full story.

First, servers and electricity cost money. There is some limit on buying more hardware or spawning more cloud instances. At that point, no matter what the theoretical speedup of our algorithm is, we won't see any actual advantages, because we'll just be queuing up more tasks than we have cores to run them on.

Second, Big O notation hides a lot of important differences between algorithms. There's a cost in creating a thread or even a Goroutine. In most real-world implementations, tuning means we spawn many fewer parallel tasks than the theoretical maximum. For example, Hadoop lets you carefully configure split size (amount of data given to each worker) and block size (amount of data stored separately on disk). Our duplicate search with `n^2` tasks was absurd; the overhead is going to be many times greater than the time it takes to do a single comparison of two items.

Third, as we saw above, to get higher available parallelism we sometimes have to do extra work, not just incur extra overhead. Sometimes that extra work is justified by the speedup we get; sometimes it is not.

CONCLUSION

This is a pretty basic discussion of how parallel algorithms are analyzed and compared to each other. If you'd like to see how parallel code might work in practice, I have a GitHub repository that runs a Net Present Value simulator using Java fork/join's `RecursiveTask` that might be of interest.

ALAN HOHN is a software architect with Lockheed Martin Mission Systems and Training. Much of his recent work has been with Java, especially Java EE (JBoss) and OSGi (Karaf), but he's worked with C, C++, C#, Ada, Python, MATLAB, Hadoop, and a few other things over time. He had a great professor at Georgia Tech for a high performance computing class, which is lucky because he stumbled into using it at work soon after.

Digital Disruption, DevOps and the Importance of Gap-free Data

Studies show that high-performing companies release software more often. This digital disruption means less time to write new code and faster deployments with shorter testing time. This need for speed leads to gaps in application performance monitoring (APM), which can undermine the user experience.

Gap-free APM data means that every activity for every operation is captured and available for as long as needed — from every single method in your application infrastructure, from one end to the other. There are three gaps organizations should eliminate:

- **Sampling and Aggregation.** Some monitoring solutions rely on aggregating data to find issues. This could work IF they kept the data so that the actions leading up to problems could be traced back - but they don't. No matter how intelligent the aggregation or analytics, if the details saved from the transactions leading up to the issue are only sampled, or the triggering event is missed, you have to wait for the issue to happen again.
- **Methods.** Today's applications are distributed, compound and complex. This leads to complicated, end-to-end application logic as developers string together databases, business logic, web services, real-time analytics, etc. To ensure visibility into every moving part, you need insight at the method level across every tier for all business operations, or the root causes of problems are invisible.
- **Topology.** Complexity escalates when pieces of the IT infrastructure are combined e.g., virtualized servers and storage can float across physical machines in the enterprise data center, in multiple providers' clouds, and even in a partner's environment. The gaps and inability to coordinate the inter-tier transactions from sampling and snapshot methods are exacerbated dramatically by the complexity of even the most optimal choreography.

CONCLUSION

If you're responsible for the performance of your company's code from development and test and the transition to production, gap-free APM data helps isolate and resolve every issue quickly and efficiently - with no finger pointing since no data point is missed. In a world ruled by complexity, gap-free data not only creates a strong IT foundation but also confidence in the digital experience delivered.

WRITTEN BY ASAD ALI

PRODUCT SPECIALIST DIRECTOR, **DYNATRACE**

PARTNER SPOTLIGHT

Dynatrace Application Monitoring

dynatrace

"It ties together data across multiple servers to show exactly where a problem may be—across an entire complex enterprise." - S. ENNAB, KRONOS INC.

CATEGORY

Application Performance Management

NEW RELEASES

Monthly

OPEN SOURCE?

No

CASE STUDY

Nordstrom, Inc. depends on the web to fuel growth and serve customers. Performance Architect Gopal Brugalette explains: "Our customers are always looking for something new and better—so we have to experiment and innovate." Their APM solution had to be the technology leader, support a cultural shift to keep ahead of customer expectations and provide strategic business information. "Dynatrace facilitated that change by giving us the insight into customer experience throughout our organization…We used to test code at least three times to be sure it would work," Brugalette says. "Now, when Dynatrace doesn't flag any issues, we have confidence that everything is OK the first time. This keeps our team moving quickly and focused on improvement."

STRENGTHS

- Gapless data from end-to-end, for every transaction with incredibly low overhead
- Outside-in approach: Start with the customer experience, not the app
- Largest APM braintrust: 130k community members, 750 R&D experts, 400 solution experts
- The only cloud to legacy, multi-modal performance management suite
- 8,000+ customers, the largest market share in the APM industry

NOTABLE CUSTOMERS

- Verizon
- Panera
- AAA
- Costco
- Volkswagen
- LinkedIn
- Fidelity Investments
- Best Buy

BLOG **dynatrace.com/en/blogs/**

TWITTER **@dynatrace**

WEBSITE **dynatrace.com**

BOTTLENECKS + LATENCIES:

HOW TO KEEP YOUR THREADS BUSY

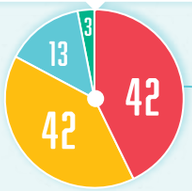
YOU AREN'T BUILDING SILICON, SO AS A DEVELOPER YOU CAN'T CHANGE HOW LONG *SOME* THINGS TAKE. BUT THERE ARE PLENTY OF BOTTLENECKS YOU *CAN* FIND AND FIX.

WHAT PERFORMANCE BOTTLENECKS ARE MOST COMMON? WE SURVEYED OVER 600 DEVELOPERS AND RESULTS SUMMARIZED THEIR RESPONSES ON THE LEFT. WHAT LATENCIES ARE JUST WHAT THEY ARE? WE TOOK DATA GATHERED BY PETER NORVIG AND JEFF DEAN AND VISUALIZED THEIR RESULTS BELOW.

BOTTLENECKS

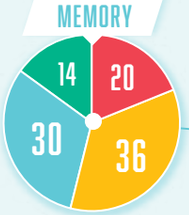
KEY — FREQUENT ISSUES — SOME ISSUES — RARE ISSUES — NO ISSUES

APPLICATION CODE

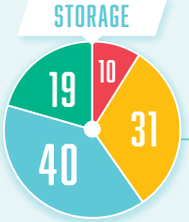


CPU

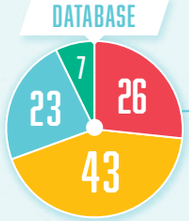
MEMORY

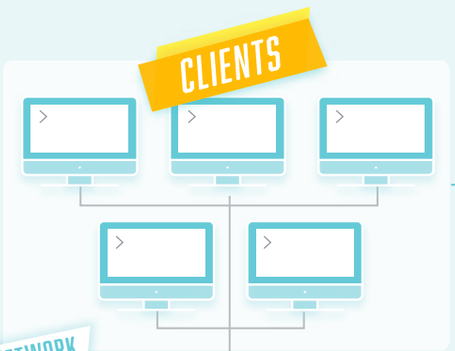


STORAGE



DATABASE





LATENCIES

SCALED WHERE ONE CPU CYCLE [.3NS] = 1 SEC

ONE CPU CYCLE
=.3NS — WHICH = 1 SEC, OR IS EQUAL TO CLAPPING YOUR HANDS

L1 CACHE ACCESS
=.9NS — WHICH = 3 SEC, OR IS EQUAL TO BLOWING YOUR NOSE

L2 CACHE ACCESS
=2.8NS — WHICH = 9 SEC, OR IS EQUAL TO BILL GATES EARNING $2,250

L3 CACHE ACCESS
=12.9NS — WHICH = 43 SEC, OR IS EQUAL TO COMPLETING AN AVERAGE MARIO BROS. LEVEL 1-1 SPEED RUN

MAIN MEMORY ACCESS
=100NS — WHICH = 6 MIN, OR IS EQUAL TO LISTENING TO QUEEN'S "BOHEMIAN RHAPSODY"

READ 1M BYTES SEQUENTIALLY FROM MEMORY
=9µS — WHICH = 9 HOURS, OR IS EQUAL TO COMPLETING A STANDARD US WORKDAY

SSD RANDOM FEED
=16µS — WHICH = 14 HOURS, OR IS EQUAL TO TAKING A FLIGHT FROM NEW YORK TO BEIJING

READ 1M BYTES SEQUENTIALLY FROM SSD
=200µS — 8 DAYS, OR IS EQUAL TO, IF THERE WERE 8 DAYS IN A WEEK, IT WOULD NOT BE ENOUGH FOR THE BEATLES TO SHOW THEY CARE

READ 1M BYTES SEQUENTIALLY FROM A SPINNING DISK
=2MS — WHICH = 70 DAYS, OR IS EQUAL TO PLANTING AND HARVESTING A ZUCCHINI

INTERNET: SF TO NYC
=71MS — WHICH = 7 YEARS, OR IS EQUAL TO ATTENDING AND GRADUATING HOGWARTS (IF YOU'RE A WITCH OR WIZARD)

Practical Tips for Detecting and Fixing Common App Performance Problems

Today's complex infrastructures combined with increasingly distributed and multi-tiered applications make it hard for IT to nail down performance issues. Making matters more difficult, data and resources are often outside of a troubleshooter's immediate control. As a result, detecting and fixing application performance problems has never been more difficult. Sluggish end-user transactions may present themselves as being slow due to the code. However, that slowness is often not the root cause, but rather a symptom of an underlying infrastructural issue hidden from view.

We've compiled a field guide that examines common, yet elusive application performance problems that reveal themselves only when you look at them from the right vantage point. They include:

- **Part 1**, The Flaw of Averages, introduces the concept of performance problems hiding in plain sight, yet masked by inadequate monitoring.
- **Part 2**, Obliterating Haystacks, shows how a big data approach can help you quickly pinpoint the needle in a haystack by removing the haystack from the equation.
- **Part 3**, The Power of Correlation Analysis, explores a particularly vexing issue: seemingly random, intermittent slowness moving from one part of an app to another.
- **Part 4**, The Performance Trinity, shows that while response time and throughput get all the attention, understanding load is the key to avoiding misdiagnosis and solving many issues.
- **Part 5**, Eliminating Leaks, provides an overview of memory leaks and similar behaviors, and introduces some common approaches to troubleshoot leak-induced problems.
- **Part 6**, Troubleshooting Leak-like Behavior, expands on concepts from the previous section, discussing how to troubleshoot other types of leak-like behavior.

This guide is based on the real-world experiences drawn from helping hundreds of organizations optimize their applications. Download the DevOps field guide.

WRITTEN BY KRISHNAN BADRINARAYANAN

PRODUCT MARKETING, **STEELCENTRAL**

PARTNER SPOTLIGHT

SteelCentral AppInternals BY RIVERBED

riverbed

> We're now able to look inside of the developers' code – without having to modify the code – while it's running in our production environment. That's fantastic. I can't imagine someone running a site of any real size without this capability.
>
> - ERIC MCCRAW, GLOBAL WEB SYSTEMS MANAGER, NATIONAL INSTRUMENTS

CATEGORY
Application Performance Management

NEW RELEASES
Quarterly

OPEN SOURCE?
No

CASE STUDY

National Instruments' public-facing website, ni.com, is updated frequently. The web systems team, which is charged with keeping the site running optimally, spent thousands of hours each year troubleshooting issues caused by new releases. This caused tension between the web systems team and the developers, and impacted customers as well.

The web systems team now uses AppInternals to find and fix root causes of application performance problems. Developers use it as well to test their code in QA. As a result, the team has:

- Reduced troubleshooting time by 90%
- Improved site stability and customer satisfaction
- Reduced bugs in production by 20% to 30%
- Reduced MTTR and have fewer incident calls
- Increased site updates from 16 to 120 per year

STRENGTHS

- Never miss a problem: Monitor user experience, and performance of code, SQL, infrastructure and web services
- Get detailed insights: Trace all transactions from user to backend while measuring system performance every second
- Understand app behavior: Visualize dependencies, derive insights or plan for capacity
- See the complete picture: Integrate with AppResponse to blend in network insights
- Set up in 15 minutes; use on and off the cloud seamlessly; no special skills needed

NOTABLE CUSTOMERS

- ABB
- Allianz
- Asurion
- Hertz
- Linkon
- Michelin
- National Instruments
- SLS

BLOG **rvbd.ly/20s7pW1** | TWITTER **@SteelCentral** | WEBSITE **www.appinternals.com**

LATENCY NUMBERS EVERYONE SHOULD KNOW

CHECKLIST BY DEEPAK KARANTH
SOFTWARE CONSULTANT, AGILE AND DEVOPS COACH AT SOFTWAREYOGA.COM

Latency, in general terms, is the amount of time between a cause and the observation of its effect. In a computer network, latency is defined as the amount of time it takes for a packet of data to get from one designated point to another. The table below presents the latency for the most common operations on commodity hardware. These data points are only approximations and will vary with the hardware and the execution environment of your code. However, their primary purpose is to enable you to make informed technical decisions to reduce latency.

OPERATION	NOTE	LATENCY	SCALED LATENCY
L1 cache reference	Level-1 cache, usually built onto the microprocessor chip itself.	0.5 ns	Consider L1 cache reference duration is 1 sec
Branch misprediction	During the execution of a program, the CPU predicts the next set of instructions. Branch misprediction is when it makes the wrong prediction. Hence, the previous prediction has to be erased and a new one must be calculated and placed on the execution stack.	5 ns	10 s
L2 cache reference	Level-2 cache is memory built on a separate chip.	7 ns	14 s
Mutex lock/unlock	This is the simple synchronization method used to ensure exclusive access to resources shared between many threads.	25 ns	50 s
Main memory reference	Time to reference main memory (i.e. RAM).	100 ns	3m 20s
Compress 1K bytes with Snappy	Snappy is a fast data compression and decompression library written in C++ by Google and is used in many Google projects like BigTable, MapReduce, and other open-source projects.	3,000 ns	1h 40 m
Send 1K bytes over 1 Gbps network	An approximation of time taken to send 1K bytes over the network, in the absence of special measures to improve the efficiency of sending data over the network.	10,000 ns	5h 33m 20s
Read 1 MB sequentially from memory	This includes the seek time as well as the time to read 1 MB of data.	250,000 ns	5d 18h 53m 20s
Round trip within same data center	We can assume that the DNS lookup will be much faster within a data center than it is to go over an external router.	500,000 ns	11d 13h 46m 40s
Read 1 MB sequentially from SSD disk	Assumes this is a SSD disk. SSD boasts random data access times of 100,000 ns or less.	1,000,000 ns	23d 3h 33m 20s
Disk seek	Disk seek is the method used to get to the sector and head in the disk where the required data exists.	10,000,000 ns	231d 11h 33m 20s
Read 1 MB sequentially from disk	Assumes this is a regular disk, not SSD. Check the difference in comparison to SSD!	20,000,000 ns	462d 23h 6m 40s
Send packet CA -> Netherlands -> CA	Round trip for packet data from U.S.A. to Europe and back.	150,000,000 ns	3472d 5h 20m

REFERENCES: DESIGNS, LESSONS AND ADVICE FROM BUILDING LARGE DISTRIBUTED SYSTEMS - PETER NORVIG'S POST ON *TEACH YOURSELF PROGRAMMING IN TEN YEARS*

IBM

IBM **API** Connect

API Connect integrates IBM StrongLoop and IBM API Management with a built-in gateway, allowing you to create, run, manage, and secure APIs and Microservices.

Unparalleled, integrated user experience.

ibm.biz/apiconnect

Scalable, Reliable, Performant: Building APIs in Node.js

APIs offer a fast, scalable way to expose just about anything, from data to services to that crazy legacy infrastructure deep in the guts of your company (no shame, everyone has it). Due to its almost extreme ability to handle high-concurrency, Node.js has become one of the most relied upon options for building APIs. Major enterprises like WalMart and Netflix have built or rebuilt major components of their platforms and services in Node for this reason.

So, Node is powerful, but its most common use cases often mean even small performance hits can add up fast. Here are a few things to remember when building APIs in Node.

Beware the event loop. The first thing most learn about Node is to stick to asynchronous I/O. But it isn't easy. For apps that handle tens or hundreds of thousands of requests per second, blocking for even fractions of a second can have a noticeable performance cost. Profiling and visualization tools like DTrace and FlameGraph are great for identifying where your app is getting hung up.

Use a reverse proxy. A well-built Node API can reliably handle a huge request volume, but everything has an upper limit. For applications that don't need to maintain state, Node scales well horizontally. This makes it a perfect fit for building RESTful APIs. To maintain API performance, put your cluster behind a reverse proxy to distribute load and handle backend calls asynchronously.

Start with an optimized framework. There are many great API frameworks written in Node that make it easy to build APIs optimized to perform at scale. Options like the open-source LoopBack framework from StrongLoop even offer easy-to-use add-on modules that manage other potential performance bottlenecks like transactions with SQL and NoSQL databases and object-relational mapping.

WRITTEN BY ALEX MURAMOTO

DEVELOPER ADVOCATE, **IBM/STRONGLOOP**

PARTNER SPOTLIGHT

API Connect

BY STRONGLOOP AND IBM

IBM

IBM API Connect is a complete solution that addresses all aspects of the API lifecycle - Create Run, Manage, Secure - for both on-premises and cloud environments.

CATEGORY
API Management

NEW RELEASES
Agile

OPEN SOURCE?
No

API LIFECYCLE

IBM API Connect offers features to manage the API lifecycle, including:

Create—create high-quality, scalable and secure APIs for application servers, databases, enterprise service buses (ESB) and mainframes in minutes.

Run—take advantage of integrated tooling to build, debug and deploy APIs and microservices using the Node.js or Java.

Manage—create and manage portals that allow developers to quickly discover and consume APIs and securely access enterprise data, and monitor APIs to improve performance.

Secure—Administrators can manage security and governance over APIs and the microservices. IT can set and enforce API policies to secure back-end information assets and comply with governance and regulatory mandates.

STRENGTHS

- Simplify discovery of enterprise systems of record for automated API creation
- Provide self-service access for internal and third-party developers through a market-leading gateway
- Ensure security and governance across the API lifecycle
- Unify management of Node.js and Java microservice applications
- Increase flexibility with hybrid cloud deployment

FEATURES

- Unified Console
- Quickly run APIs and microservices
- Manage APIs with ease
- Readily secure APIs and microservices
- Create APIs in minutes

BLOG **developer.ibm.com/apiconnect/blog/** | TWITTER **@ibmapiconnect** | WEBSITE **ibm.com/apiconnect**

How HTTP/2 Is Changing Web Performance Best Practices

BY CLAY SMITH
DEVELOPER ADVOCATE, **NEW RELIC**

QUICK VIEW

01 HTTP/2 is the successor of HTTP that was ratified in May 2015.

02 It is changing long-standing web performance optimizations.

03 Best practices for migrating and using it in production are still being finalized.

04 This article covers how HTTP/2 is different, how it improves latency, and how to debug it in production.

05 Measuring real-user performance is critical during a HTTP/2 migration.

The Hypertext Transfer Protocol (HTTP) underpins the World Wide Web and cyberspace. If that sounds dated, consider that the version of the protocol most commonly in use, HTTP 1.1, is nearly 20 years old. When it was ratified back in 1997, floppy drives and modems were must-have digital accessories and Java was a new, up-and-coming programming language. Ratified in May 2015, HTTP/2 was created to address some significant performance problems with HTTP 1.1 in the modern web era. Adoption of HTTP/2 has increased in the past year as browsers, web servers, commercial proxies, and major content delivery networks have committed to or released support.

Unfortunately for people who write code for the web, transitioning to HTTP/2 isn't always straightforward, and a speed boost isn't automatically guaranteed. The new protocol challenges some common wisdom when building performant web applications, and many existing tools—such as debugging proxies—don't support it yet. This article is an introduction to HTTP/2 and how it changes web performance best practices.

BINARY FRAMES: THE "FUNDAMENTAL UNIT" OF HTTP/2

One benefit of HTTP 1.1 (over non-secure connections, at least) is that it supports interaction with web servers using text in a telnet session on port 80: typing `GET / HTTP/1.1` returns an HTML document on most web servers. Because it's a text protocol, debugging is relatively straightforward.

Instead of text, requests and responses in HTTP/2 are represented by a stream of binary frames, described as a "basic protocol unit" in the HTTP/2 RFC. Each frame has a type that serves a different purpose. The authors of HTTP/2 realized that HTTP 1.1 will exist indefinitely (the Gopher protocol still is out there, after all). The binary frames of an HTTP/2 request map to an HTTP 1.1 request to ensure backwards compatibility.

There are some new features in HTTP/2 that don't map to HTTP 1.1, however. Server push (also known as "cache push") and stream reset are features that correspond to types of binary frames. Frames can also have a priority that allows clients to give servers hints about the priority of some assets over others.

Other than using Wireshark 2.0, one of the easiest ways to actually see the individual binary frames is by using the net-internals tab of Google Chrome (type `chrome://net-internals/#http2` into the address bar). The data

can be hard to understand for large web pages. Rebecca Murphey wrote a useful tool for displaying it visually in the command line.

Additionally, the protocol used to fetch assets can be displayed in the Chrome web developer tools—right click on the column header and select "Protocol":

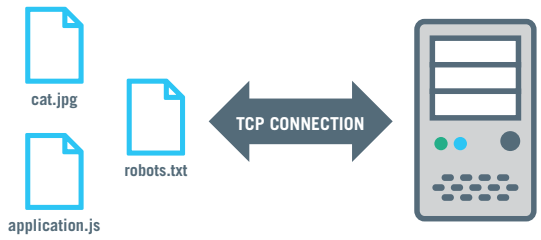
All major browsers require HTTP/2 connections to be secure. This is done for a practical reason: an extension of TLS called Application-Layer Protocol Negotiation (ALPN) lets servers know the browser supports HTTP/2 (among other protocols) and avoids an additional round trip. This also helps services that don't understand HTTP/2, such as proxies—they see only encrypted data over the wire.

REDUCING LATENCY WITH MULTIPLEXING

A key performance problem with HTTP 1.1 is latency, or the time it takes to make a request and receive a response. This issue has become more pronounced as the number of images and amount of JavaScript and CSS on a typical webpage continue to increase. Every time an asset is fetched, a new TCP connection is generally needed. This requirement is important for two reasons: the number of simultaneous open TCP connections per host is limited by browsers, and there's a performance penalty incurred when establishing new connections. If a physical web server is far away from users (for example, a user in Singapore requesting a page hosted at a data center on the U.S. East Coast), latency also increases. This scenario is not uncommon—one recent report says that more than 70% of global Internet traffic passes through the unmarked data centers of Northern Virginia.

HTTP 1.1 offers different workarounds for latency issues, including pipelining and the Keep-Alive header. However, pipelining was never widely implemented, and the Keep-Alive header suffered from head-of-line blocking: the current request must complete before the next one can be sent.

In HTTP/2, multiple asset requests can reuse a single TCP connection. Unlike HTTP 1.1 requests that use the Keep-Alive header, the requests and response binary frames in HTTP/2 are interleaved and head-of-line blocking does not happen. The cost of establishing a connection (the well-known "three-way handshake") has to happen only once per host. Multiplexing is especially beneficial for secure connections because of the performance cost involved with multiple TLS negotiations.



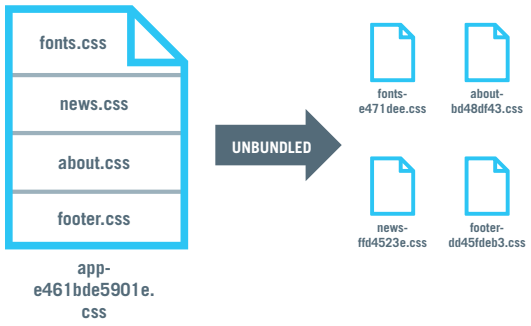
Requests for multiple assets on a single host use a single TCP connection in HTTP/2.

IMPLICATIONS FOR WEB PERFORMANCE: GOODBYE INLINING, CONCATENATION, AND IMAGE SPRITES?

HTTP/2 multiplexing has broad implications for front-end web developers. It removes the need for several long-standing workarounds that aim to reduce the number of connections by bundling related assets, including:

- **Concatenating JavaScript and CSS files:** Combining smaller files into a larger file to reduce the total number of requests.
- **Image spriting:** Combining multiple small images into one larger image.
- **Domain sharding:** Spreading requests for static assets across several domains to increase the total number of open TCP connections allowed by the browser.
- **Inlining assets:** Bundling assets with the HTML document source, including base-64 encoding images or writing JavaScript code directly inside `<script>` tags.

With unbundled assets, there is greater opportunity to aggressively cache smaller pieces of a web application. It's easiest to explain this with an example:



A concatenated and fingerprinted CSS file unbundles into four smaller fingerprinted files.

A common concatenation pattern has been to bundle style sheet files for different pages in an application into a single CSS file to reduce the number of asset requests. This large file is then fingerprinted with an MD5 hash of its contents in the filename so it can be aggressively cached by browsers. Unfortunately, this approach means that a very small change to the visual layout of the site, like changing the font style for a header, requires the entire concatenated file to be downloaded again.

When smaller asset files are fingerprinted, significant amounts of JavaScript and CSS components that don't change frequently can be cached by browsers—a small refactor of a single function no longer invalidates a massive amount of JavaScript application code or CSS.

Lastly, deprecating concatenation can reduce front-end build infrastructure complexity. Instead of having several pre-build

steps that concatenate assets, they can be included directly in the HTML document as smaller files.

POTENTIAL DOWNSIDES OF USING HTTP/2 IN THE REAL WORLD

Optimizing only for HTTP/2 clients potentially penalizes browsers that don't yet support it. Older browsers still prefer bundled assets to reduce the number of connections. As of February 2016, caniuse.com reports global browser support of HTTP/2 at 71%. Much like dropping Internet Explorer 8.0 support, the decision to adopt HTTP/2 or go with a hybrid approach must be made using relevant data on a per-site basis.

As described in a post by Kahn Academy Engineering that analyzed HTTP/2 traffic on its site, unbundling a large number of assets can actually increase the total number of bytes transferred. With zlib, compressing a single large file is more efficient than compressing many small files. The effect can be significant on an HTTP/2 site that has unbundled hundreds of assets.

Using HTTP/2 in browsers also requires assets to be delivered over TLS. Setting up TLS certificates can be cumbersome for the uninitiated. Fortunately, open-source projects such as Let's Encrypt are working on making certificate registration more accessible.

A WORK IN PROGRESS

Most users don't care what application protocol your site uses—they just want it to be fast and work as expected. Although HTTP/2 has been officially ratified for almost a year, developers are still learning best practices when building faster websites on top of it. The benefits of switching to HTTP/2 depend largely on the makeup of the particular website and what percentage of its users have modern browsers. Moreover, debugging the new protocol is challenging, and easy-to-use developer tools are still under construction.

Despite these challenges, HTTP/2 adoption is growing. According to researchers scanning popular web properties, the number of top sites that use HTTP/2 is increasing, especially after CloudFlare and WordPress announced their support in late 2015. When considering a switch, it's important to carefully measure and monitor asset- and page-load time in a variety of environments. As vendors and web professionals educate themselves on the implications of this massive change, making decisions from real user data is critical. In the midst of a website obesity crisis, now is a great time to cut down on the total number of assets regardless of the protocol.

4/4 MAJOR BROWSER VENDORS AGREE: HTTPS IS REQUIRED

Firefox, Internet Explorer, Safari, and Chrome all agree: HTTPS is required to use HTTP/2 in the first place. This is critical because of a new extension to Transport Layer Security (TLS) that allows browsers and clients to negotiate which application-layer protocol to use. When a TLS connection is established for the first time, the server broadcasts support for HTTP 1.1, SPDY, or HTTP/2 without an additional round trip.

Because of changes Google recently announced, it's critical that backend SSL libraries are updated before Chrome drops support for the older Next Protocol Negotiation. standard in favor of Application Layer Protocol Negotiation Unfortunately, for almost every modern Linux distribution, this means compiling web server software from source code with OpenSSL version 1.0.2 (not a trivial task).

With the latest version of OpenSSL installed on servers, however, it's possible to check hosts for HTTP/2 support from the command line:

```
me@ubuntu-trusty-64:~$ echo | openssl s_client
-alpn h2 -connect google.com:443 | grep ALPN
ALPN protocol: h2
DONE
```

A web-based tool from KeyCDN and the is-http2 package can also help determine host support.

The transition to the new protocol is relatively straightforward for sites that are already delivered securely. For non-secure sites, web servers (and potentially CDNs) will need to be correctly configured for HTTPS. New open-source projects such as Let's Encrypt aim to make this process as easy, free, and automated as possible. Of course, regardless of HTTP/2 support, moving to HTTPS is becoming more important. Some search engines now use secure sites as a positive signal in page ranking, and privacy advocates and industry experts strongly recommend it.

DETERMINING BACK END AND CONTENT DELIVERY NETWORK SUPPORT

If HTTPS is properly configured, the next step is determining if the server or proxy software supports HTTP/2. The IETF HTTP Working Group maintains a comprehensive list of known implementations on its website, and popular web servers have all released or committed to support. Most popular application development languages have HTTP/2 packages as well.

SERVER OR CLOUD PROVIDER	HTTP/2 SUPPORT
Apache	> 2.4.17
nginx	> 1.9.5
Microsoft IIS	Windows Server 2016 Technical Preview
Heroku	No (as of 1/16)
Google AppEngine	Available with TLS
Amazon S3	No (as of 1/16)

Support for the full suite of HTTP/2 features, especially server push, is not guaranteed. It's necessary to read the release notes to determine which features are fully supported.

If your site uses assets delivered by a Content Delivery Network (CDN), major vendors like CloudFlare and KeyCDN already support the new protocol even if your back end doesn't. With some providers, enabling HTTP/2 between your client and the edge locations can be as easy as toggling a radio button on a web form.

CDN	SUPPORTS HTTP/2 AS OF JAN. 2016?
Akamai	Yes
CloudFare	Yes
KeyCDN	Yes
Amazon CloudFront	No

USING WIRESHARK FOR DEBUGGING

HTTP/2 tooling still has a long way to go before catching up with HTTP 1.1. Because HTTP/2 is a binary protocol, simple debugging using telnet won't work, and standard debugging proxies like Charles and Fiddler do not offer support as of January 2016.

In the first part of this article, we discussed how to use Chrome Net Internals (`chrome://net-internals#http2`) to debug traffic. For more advanced analysis, using the low-level C (or the Python bindings) of the nghttp2 library or Wireshark 2.0 is needed. Here, we'll focus on Wireshark.

Configuring Wireshark to view an HTTP/2 frame requires additional setup because all traffic is encrypted. To view Firefox or Chrome HTTP/2 traffic, you have to log TLS session information to a file specified by the environment variable `SSLKEYLOGFILE`. On Mac OS X, set the environment variable before launching the browser from the command line (you can see Windows instructions here):

```
$ export SSLKEYLOGFILE=~/Desktop/tls_fun.log
$ open -a Google\ Chrome https://nghttp2.org/
```

Wireshark must be configured to use the SSLKEYLOGFILE in the preferences menu under the "SSL" protocol listing.

When starting Wireshark for the first time, a network interface needs to be selected. Filtering only on port 443 is a good idea since all HTTP/2 traffic in Chrome is secure. After clicking on the shark icon, recording begins for all traffic sent over that interface. The output can be overwhelming, but it's easy to filter HTTP/2-only traffic by typing "http2" into the filter text box. When HTTP/2 packets are captured, they can now be decrypted into individual HTTP2 binary frames:

Using the tabs at the bottom of the data panel, it's possible to see the decrypted frames. HEADERS frames, which are always compressed, can also be displayed decompressed.

THE TRANSITION IS NOT YET STRAIGHTFORWARD

For many web applications in early 2016, transitioning to HTTP/2 is not yet straightforward. Not only is HTTPS required in order to use the new protocol in browsers, it's likely that server software will also need to be upgraded. In some cases, particularly with Backend-as-a-Service providers or Content Delivery Networks, HTTP/2 support might not be available—or even promised—yet. Lastly, easy-to-use debugging tools are still being worked on.

As many teams have already discovered, it is likely that migrating any large site to HTTP/2 will contain surprises. Despite these challenges, many large web properties have successfully launched HTTP/2 support with significant performance benefits. Carefully measuring real-user performance and understanding the limitations of current tooling is helpful for making the transition as smooth as possible.

ADDITIONAL RESOURCES

Let's Encrypt

Why isn't HTTPS everywhere yet?

HTTP/2 on IIS

Moving to HTTP/2 on nginx 1.9.5

is-http npm module

Is TLS Fast Yet?

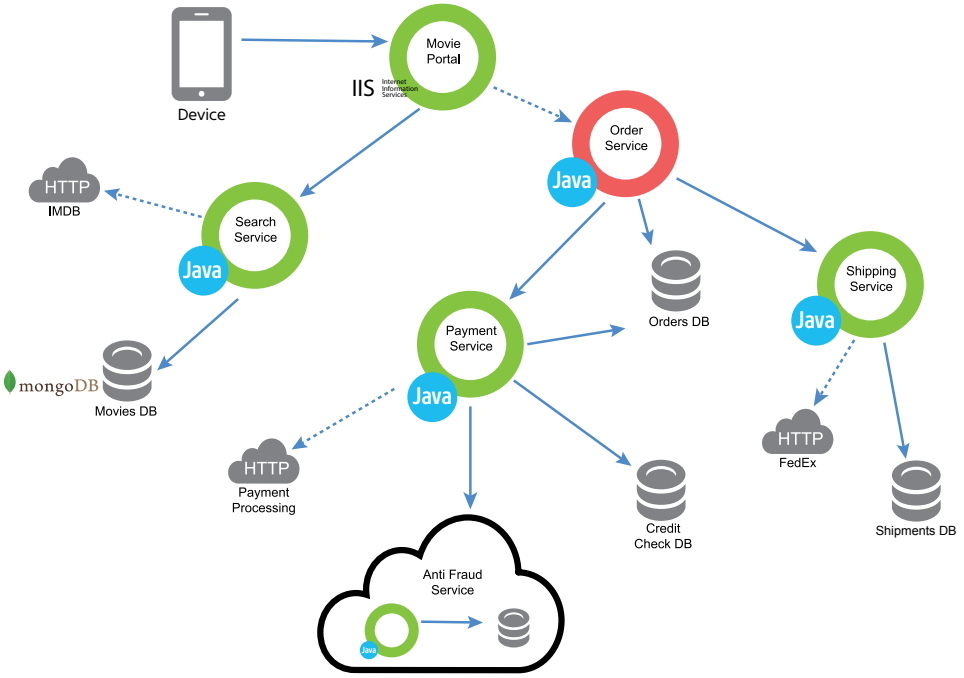
This article was written by Clay Smith, with contributions of technical feedback and invaluable suggestions by Jeff Martens, Product Manager for New Relic Browser, and web performance expert Andy Davies.

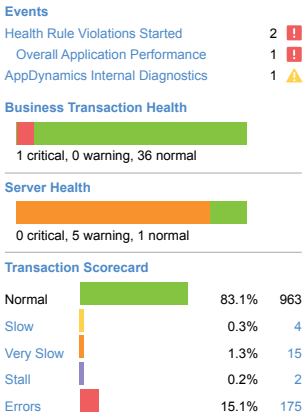
CLAY SMITH is a Developer Advocate at New Relic. He previously was a Senior Software Engineer at PagerDuty and has built many APIs and web applications at startups and large enterprises. As the author of one of the first apps written using Swift, he also likes to experiment with iOS development. Clay studied Computer Science with an emphasis on Artificial Intelligence and Linguistics at the University of Chicago.

APPDYNAMICS

There's nothing about Java that AppDynamics doesn't see.

AppDynamics gives you the visibility to take command of your Java application's performance, no matter how complicated or distributed your environment is.





Start your Free Trial

When your business runs on Java, count on AppDynamics to give you the complete visibility you need to be sure they are delivering the performance and business results you need — no matter how complex, distributed or asynchronous your environment, live 'in production' or during development.

See every line of code. Get a complete view of your environment with deep code diagnostics and auto-discovery. Understand performance trends with dynamic baselining. And drastically reduce time to root cause and remediation.

See why the world's largest Java deployments rely on the AppDynamics Application Intelligence Platform. Sign up for a FREE trial today at www.appdynamics.com/java.

What's Exciting About Java 9 and Application Performance Monitoring

In today's modern computing age, constant enhancements in software innovations are driving us closer to an era of software revolution. Perhaps in the distant future, that may be how the 21st century is remembered best. Among the popular software languages out there, however, Java continues to have the largest industry footprint, running applications around the globe producing combined annual revenue in trillions. That's why keeping up on the JDK is a high priority. Despite having a massive API to improve programming productivity, Java has also grown due to its high performance yet scalable JVM runtime, building among the fastest computing modern applications. As Java's footprint expands, JDK innovations continue to impact billions of lines of code. As AppDynamics continues to grow, our focus towards supporting Java is only furthered by our customer use & industry adoption of the JVM.

WRITTEN BY AAKRIT PRASAD

HEADING CORE & APM PRODUCTS, PRODUCT MANAGEMENT, **APPDYNAMICS**

PARTNER SPOTLIGHT

Application Intelligence Platform BY APPDYNAMICS

APPDYNAMICS

If your business runs on apps, Application Intelligence is for you. Real-time insights into application performance, user experience, and business outcomes.

CATEGORY
Application Performance Management

NEW RELEASES
Bi-Yearly

OPEN SOURCE?
No

CASE STUDY

"AppDynamics has enabled us to move towards data-driven troubleshooting rather than 'gut-feels.' The solution gives us the application intelligence to know when things aren't functioning optimally."

- Nitin Thakur, technical operations manager, Cisco

STRENGTHS

Application Performance Management is a technology solution that provides end-to-end business transaction-centric management of the most complex and distributed applications. Auto-discovered transactions, dynamic baselining, code-level diagnostics, and Virtual War Room collaboration ensure rapid issue identification and resolution to maintain an ideal user experience.

NOTABLE CUSTOMERS

- NASDAQ
- eHarmony
- DIRECTV
- Cisco
- Citrix
- Hallmark

BLOG **blog.appdynamics.com** | TWITTER **@AppDynamics** | WEBSITE **appdynamics.com**

Benchmarking Java Logging Frameworks

BY ANDRE NEWMAN
SOFTWARE DEVELOPER

QUICK VIEW

01 In distributed, cloud-based environments, it's equally important to understand both application and network performance.

02 Active monitoring, often used for website performance, can also provide you with insights into cloud provider networks.

03 Active monitoring can provide you a stack trace for your network, showing the performance of each network that your traffic traverses.

04 Consider adding key network connectivity and service metrics to your arsenal in order to get ahead of cloud outages.

One of the most common arguments against logging is its impact on your application's performance. There's no doubt logging can cost you some speed; the question is how much. When you're armed with some real numbers, it's much easier to find the right amount to log. In this article, we'll compare the performance and reliability of four popular Java logging frameworks.

THE CONTENDERS

For this test, we investigated four of the most commonly used Java logging frameworks:

1. Log4j 1.2.17
2. Log4j 2.3
3. Logback 1.1.3 using SLF4J 1.7.7
4. JUL

We tested each framework using three types of appenders: file, syslog, and socket. For syslog and socket appenders, we sent log data to a local server over both TCP and UDP. We also tested asynchronous logging using each framework's respective AsyncAppender. Note that this test doesn't include asynchronous loggers, which promise even faster logging for Log4j 2.3.

SETUP AND CONFIGURATION

Our goal was to measure the amount of time needed to log a number of events. Our application logged 100,000 DEBUG events (INFO events for JUL) over 10 iterations (we actually did 11 iterations, but the first was discarded due to large startup times to warm the JIT). To simulate a workload, we generated prime numbers in the background. We repeated this test three times and averaged the results. This stress test also drives the logging frameworks harder than they would in a typical workload because we wanted to push them to their limit. For example, in a typical workload, you won't see as many dropped events, because events will be more spread out over time, allowing the system to catch up.

We performed all testing on an Intel Core i7-4500U CPU with 8 GB of RAM and Java SE 7 update 79.

In the interest of fairness, we chose to keep each framework as close to its default configuration as possible. You might experience a boost in performance or reliability by tweaking your framework to suit your application.

APPENDER CONFIGURATION

We configured our file appenders to append entries to a single file using a PatternLayout of `%d{HH:mm:ss.SSS} %-5level - %msg%n`. Our socket appenders sent log data to a local socket server, which then wrote the entries to a file (see this link for an example using Log4j 1.2.17). Our syslog appenders sent log data to a local rsyslog server, which then forwarded the entries to Loggly.

The AsyncAppender was used with the default configuration, which has a buffer size of 128 events (256 events for Logback) and does not block when the buffer is full.

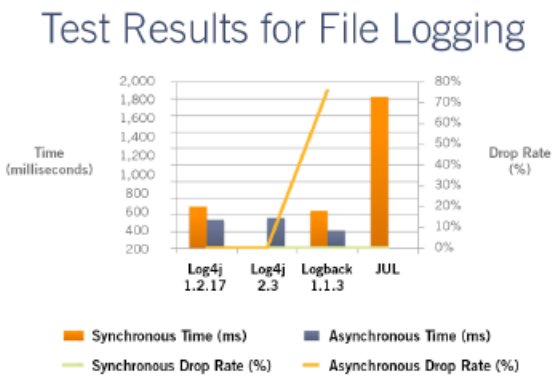
TEST RESULTS

FILE APPENDER

Logback came ahead in synchronous file logging, performing 9% faster than Log4j 2.3 and 11% faster than Log4j 1.2.17. All three

frameworks significantly outperformed JUL, which took over four times as long as Logback.

Using asynchronous appenders, run times decreased noticeably. Logback once again showed the highest performance but dropped most of its log events in order to do so—76%! None of the other frameworks dropped any events running synchronously or asynchronously. This is due to the behavior of Logback's AsyncAppender, which drops events below WARNING level if the queue becomes 80% full. Log4j 1.2.17 saw improved run times while managing to successfully record each event. Log4j 2.3 saw an increase in performance over the synchronous appender, but came third after Log4j 1.2.17.

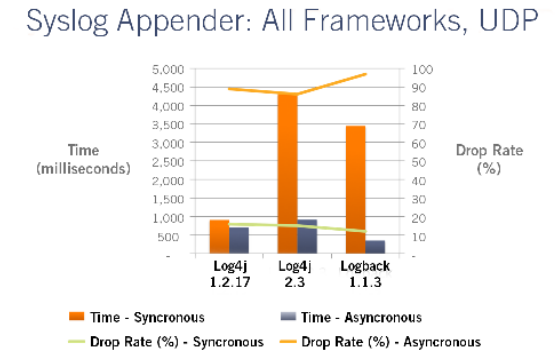


SYSLOG APPENDER

UDP

Using UDP, each framework experienced a similar rate of dropped messages due to packet loss. While Log4j 1.2.17 was the fastest, it also experienced the highest drop rate. Compared with Log4j 1.2.17, Log4j 2.3 saw a 1% improvement in dropped messages with a 9% drop in performance. SLF4J provided a somewhat more reliable experience for a substantial drop in performance.

Using an asynchronous appender resulted in a much shorter run time but also a much higher drop in reliability. The most striking difference came for Logback, which ran nearly 10 times faster but had eight times the number of dropped events.



TCP

As expected, TCP with Log4j 2.3 proved to be a much more reliable transmission method. (You can view the test results here.) We saw a small number of dropped messages, but it was negligible when compared with UDP. The cost of this higher reliability is a run time that's nearly twice as long.

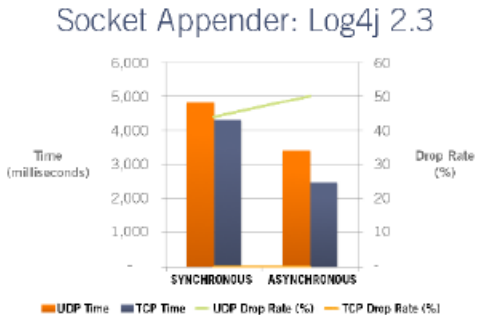
With an asynchronous appender, we saw a decent boost in performance with no drop in throughput.

SOCKET APPENDER

UDP

Log4j 2.3's socket appender was the slowest combination we tested. It was also one of the most error prone, dropping 44% of the events sent to it.

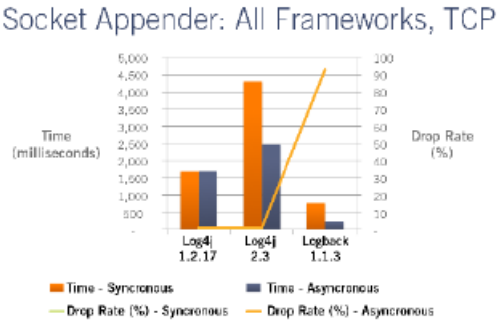
Using an asynchronous appender provided an almost 30% improvement in performance but with a 6% decrease in reliability.



TCP

Log4j 1.2.17 showed a nearly 3-second improvement over Log4j 2.3 when using TCP. However, the star of the show is Logback, which managed to perform in less than one-fifth the time of Log4j 2.3. You can see the Log4j 2.3 test results here.

When the application is logging asynchronously, Log4j 2.3 showed a marked improvement. Log4j 1.2.17 maintained its run time, but showed a small increase in the number of dropped events. Logback maintained its performance lead, but in doing so dropped over 90% of events.



CONCLUSION

The combination that we found to offer the best performance and reliability is Log4j 1.2.17's FileAppender using an AsyncAppender. This setup consistently completed in the fastest time with no dropped events. For raw performance, the clear winner was Logback's FileAppender using an AsyncAppender.

There's often a trade-off between fast and reliable logging. Logback in particular maximized performance by dropping a larger number of events, especially when we used an asynchronous appender. Log4j 1.2.17 and 2.3 tended to be more conservative but couldn't provide nearly the same performance gains.

ANDRE NEWMAN is a technical writer and regular contributor to Loggly. He is also a software developer specializing in enterprise application development. Andre has over a decade of experiencing developing in Java, VB.NET, C#, and C++. He has additional experience in systems administration and deployment. When Andre's not busy writing, he's either hacking away on an Arduino or building smartwatch apps.

Using HTTP/2 to Reduce Latency and Make Your Web Applications Faster

It's been a year since HTTP/2, the latest version of the network protocol the web runs on, was published as a spec by the Internet Engineering Task Force (IETF). One year after the spec was introduced, adoption of HTTP/2 has steadily increased to 7.4% of all websites, according to W3 Techs, an adoption percentage that has more than doubled in the last six months.

Why should you migrate your web applications to HTTP/2? The main reason is speed. An HTTP/2-based site will simply load faster than a site in HTTP/1.1, a nearly 20-year-old protocol that doesn't do a very efficient job of handling the network "handshake" between browser client and web server that happens every time a user tries to access a web page. As websites have grown larger and more complex, these inefficiencies have proved to be a drag on web performance. Organizations have had to adapt by using techniques such as domain sharding, in-line images, and file concatenation.

HTTP/2 largely remedies this. The new version of the protocol allows multiple requests to be sent from client to server, one after the other, on the same TCP connection, while responses to those requests can be received out of order – eliminating the need for multiple connections between the client and the server. This reduces network latency, which in turn makes web pages load faster. HTTP/2 also compresses HTTP headers, allows the server to push resources to the client that haven't been requested yet, and allows the client to indicate to the servers which resources are more important than others.

The end result is that a browser client can make faster and fewer connections to a web host, speeding up the time it takes to download content from that server. The smaller content payloads and optimized TCP connections of HTTP/2 are especially ideal for mobile applications and sites.

There are various ways to start using HTTP/2. You can upgrade your web server to the latest versions of Apache and Nginx. Your hosting or CDN provider can upgrade your site to HTTP/2 even faster. No coding changes are required. Then keep monitoring your sites to make sure they live up to their potential.

WRITTEN BY DENNIS CALLAGHAN

DIRECTOR OF INDUSTRY INNOVATION, **CATCHPOINT SYSTEMS**

PARTNER SPOTLIGHT

Catchpoint Synthetic

BY CATCHPOINT SYSTEMS

catchpoint™

Smarter, faster end-user monitoring for digital business

CATEGORY

EUM (End-User Experience Monitoring) / Performance Monitoring

NEW RELEASES

8x Annually

OPEN SOURCE?

No

STRENGTHS

- End-user experience monitoring (EUM) designed expressly for digital business
- Only EUM platform to simultaneously capture, index, and store object-level data
- The industry's most extensive monitor types
- The industry's most extensive global node coverage

CASE STUDY

Priceline.com relies on innovative proprietary architecture that combines internal and third-party partner components to offer high-performing websites and services to millions of customers. Speed, scalability, and consistency are keys to Priceline.com's continued success.

WHAT THEY USE

Catchpoint Synthetic

- Object Monitoring
- DNS Monitoring
- Hosts & Zone Monitoring

Real User Measurement

THE SOLUTION

Utilizing Catchpoint's monitoring locations to proactively monitor multistep transactions, DNS services, and API calls: Priceline continuouly benchmarks performance with industry peers to define appropriate goals to maintain its leadership position.

Catchpoint Insight: Priceline automatically correlated internal data with synthetic monitoring metrics to diagnose problems and rapidly find root causes across complex multi-tier architectures.

Zones and Hosts: Underperforming components (third-party vendors, internal components, etc.) were quickly troubleshooted.

Alerts: Problems and bottlenecks were immediatly communicated to Priceline.

Catchpoint Analytics: Priceline utilized Catchpoint's analytics to examine the impact of front-end code optimizations.

NOTABLE CUSTOMERS

- Business Insider
- Comcast
- Google
- Honeywell
- Kate Spade
- Trip Advisor
- Verizon
- Wayfair

BLOG **blog.catchpoint.com** | TWITTER **@catchpoint** | WEBSITE **catchpoint.com**

Executive Insights on Performance + Monitoring

QUICK VIEW

01
Performance and monitoring grow more challenging as more data and more layers of abstraction are added with no end in sight.

02
Customers need real user monitoring from the server to the application across all devices to understand the performance of their apps for the optimal UX.

03
Developers need to measure performance earlier in the development process and be sensitive to how latency can accrue as their application integrates with other apps.

BY TOM SMITH
MARKETING STRATEGIST AND RESEARCH ANALYST, **DZONE**

To gather insights on the state of performance and monitoring today, we spoke with 11 executives at nine companies providing performance and monitoring services to clients.

Specifically, we spoke to:

Dustin Whittle, Developer Evangelist, AppDynamics | **Michael Sage**, Chief DevOps Evangelist, Blazemeter | **Rob Malnati**, V.P. Marketing and **Pete Mastin**, Product Evangelist, Cedexis | **Charlie Baker**, V.P. Product Marketing, Dyn | **Andreas Grabner**, Technology Strategist, Dynatrace | **Dave Josephson**, Developer Evangelist, and **Michelle Urban**, Director of Marketing, Librato | **Bob Brodie**, CTO, SUMOHeavy | **Christian Beedgen**, CTO and Co-Founder, Sumo Logic | **Nick Kephart**, Senior Director Product Marketing, ThousandEyes

KEY FINDINGS

01 The keys to performance and monitoring are providing a holistic view of 1) what's happening from the infrastructure to the application, regardless of the device, and 2) the quality of the UX the end user is having. Tools are enabling companies to automate and scale monitoring so they can be notified of road blocks or traffic jams that may be negatively affecting the customer experience (CX). You want to know why people are not using your app—if it's slow or broken, or just not useful. Don't just see the performance but know the "why" behind the performance. Learn the best path to reduce latency. You need access to all of the data to be able to provide a thorough analysis. While artificial intelligence (AI) is moving us towards full post-deployment automation with no human intervention required, we're not there yet.

02 The biggest changes to performance and monitoring have been the movement from the data center to the cloud and the increase in the complexity of applications, which has made monitoring performance more challenging. Seeing the end-to-end user experience is the best way to monitor; however, new cloud infrastructures, new levels of abstraction, and distributed microservices are making it difficult to stay ahead and provide the visibility clients need. DevOps and Continuous Delivery have changed the way we create software as well as speed to market. Apps have changed and expectations have changed. New Relic came along in 2008 and enabled users to monitor for $150 per server, providing real-time statistics and visibility into applications, databases, browsers, and disks. Such tools have reduced mean time to innocence, enabling users to quickly identify if IP issues are taking place inside or outside their system for quicker problem resolution. Bleeding edge companies (Twitter, Facebook, Google, and Netflix) are pushing to machine learning and metacomputation to know

what's happening. We're also seeing a movement to unlock data between customer boundaries so clients with the same vendor can benefit from each other's data.

03 Everyone has created their own proprietary solution but tend to use additional resources like New Relic, Splunk, and open-source tools to build out the capabilities of their solution. While the solutions are proprietary, they are built to seamlessly integrate with other solutions and support all languages.

04 Scalability is the most consistently mentioned "real world problem to be solved," followed by the need to monitor across an ever broader range of platform formats and application types to ensure a good UX. Scalability is particularly important to e-commerce and companies expanding their presence from brick and mortar to digital. It's important to help companies know where to deploy in the cloud and how to automatically bce traffic loads, while providing a low mean time to innocence. Shifting the measurement of performance earlier in the build cycle will enable companies to proactively find and correct problems as early as possible. Multi-vendor and multi-path are keys to providing an outstanding UX given the vagaries of the internet. Understand and test how minor code changes can affect the performance and UX of the app since any issues will become amplified as use increases.

05 Businesses generally fail to understand the implications poor performance can have on their business and what can be done to improve performance post-ISP, which is the most common issue vendors see affecting their clients. In addition, given the changes with the use of microservices and multiple platforms, you have a distributed system that can be difficult to understand and which you have very little control over. There's a need to stay abreast of the changes and prepare to scale. All of these factors reinforce the need for trusted performance monitoring providers for the web and applications.

06 The biggest opportunities for improving performance is becoming more educated about the subject and injecting performance monitoring earlier in the development process. Real user monitoring (RUM) will be the standard in the future; however, users need to become aware of the tools available for monitoring and improving performance. As companies learn the effect of an improved UX, they will put more emphasis on improving performance. Moving to the cloud will eliminate certain infrastructure issues while new tools and technologies will enable companies to take advantage of elasticity without putting a burden on their IT staff.

07 The only concerns about the current state of performance and monitoring revolve around customers' lack of understanding of the importance of measuring performance in the development process, the increasing complexity of the tools needed to monitor, and the exponential increases in data and machines. There are still a lot of companies that do not view performance and monitoring as integral to the development process. Furthermore, as tools become more powerful, they become more complex, and you have to hire people to manage the tools.

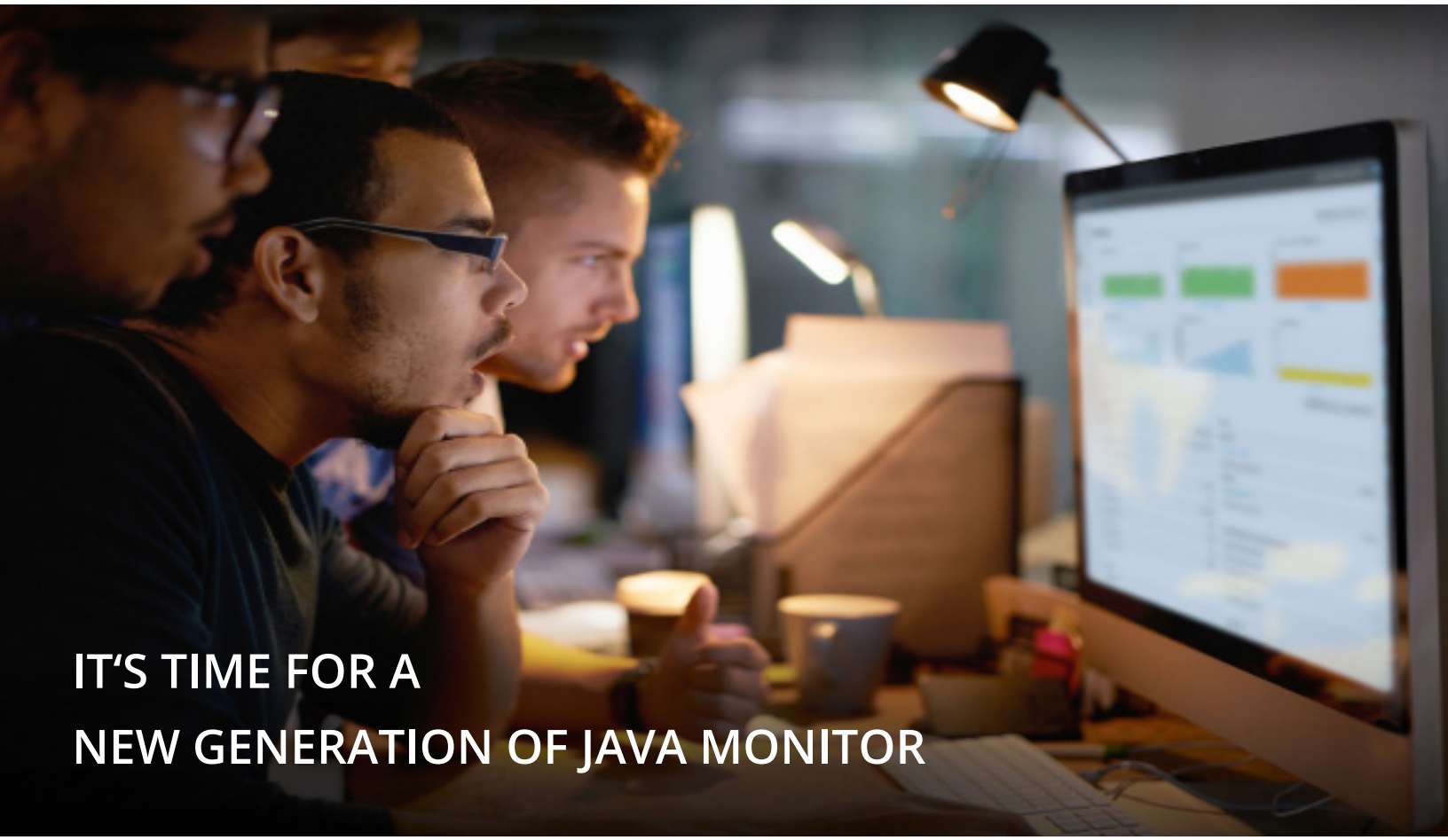
08 While there were a wide variety of suggestions for developers to optimize the performance of the applications they are developing, three were mentioned more than once: 1) have a DevOps mentality and be familiar with the DevOps process; 2) have a holistic view and understanding of performance and monitoring; and, 3) stay up to date with PHP—standards, coding, and messaging. Additional suggestions included: understand architectural concepts; understand automation since it is inevitable; become familiar with load testing; understand where bottlenecks are; know how to optimize performance; know infrastructure and cloud-based requirements; don't assume third-party services will always work; and be humble—don't get wed to a single way of doing something, as you will need to figure out workarounds to unanticipated problems.

09 Additional considerations, or "final thoughts," about performance and monitoring included:

- **Understand performance is cross-functional** and not the responsibility of a particular team. While someone has to take responsibility, don't let DevOps become a "center of excellence" silo.
- **Keep in mind what's free open source and what's expensive software.** Try before you buy. A lot of software has 30-day free trials. Take advantage of these trials to get to know the software.
- **Remember the role of content delivery networks (CDNs) is important** with regards to performance. People need to understand that dynamic and static content are not all the same.
- **Think of ways to stay up to date** with the changes in technologies.
- **Get educated about the space and the challenges** that exist with regards to using the internet as the primary vehicle for interacting with businesses and customers.
- **Remember that moving to the cloud is adding a different level of abstraction.** You think you can trust what's underneath; however, you lose the visibility, and the ability to know, as the layers of abstraction increase.
- **Consider that AI is where the major change is coming** to manage infrastructure in the future where we can let machines do more of the work.

TOM SMITH is a Research Analyst at DZone who excels at gathering insights from analytics—both quantitative and qualitative—to drive business results. His passion is sharing information of value to help people succeed. In his spare time, you can find him either eating at Chipotle or working out at the gym.

fusion reactor™



IT'S TIME FOR A NEW GENERATION OF JAVA MONITOR

FusionReactor goes beyond traditional APM tools to give you unrivaled insight into how your Java code performs and executes in production

FusionReactor doesn't "just monitor" and put the burden on you to figure things out. FusionReactor puts you in control, so that you can instantly isolate production issues and performance bottlenecks with our integrated low overhead Production Debugger and Profiler. Plus pro-actively improve application resilience with FusionReactor's unique Crash Protection capability.

No other monitoring solution gives you the same level of depth, insight or control of your Java applications 'in production'.

FusionReactor - Find it. Fix it. Prevent it.

Start Free Trial

www.fusion-reactor.com

Still using log files to isolate production issues? Perhaps there is an alternative...

It was interesting to read in the *2015 DZone APM Guide* that the #1 tool for finding production issues is application logs (94%). Wow – that's sad – because delving through logs is painstaking, time-consuming and generally not very much fun. How can this be? We've got all these wonderful APM tools available – yet developers still reach for log files! Is that the best we can do?

Traditional APM tools provide some neat metric graphs and can alert you when something seems wrong, but they don't tell you much at the level of detail software engineers need to get to the actual root of the issue. So we grep through our logs, dive into the heap, tally our object instances, run stack trace over and over, guess some breakpoints or include debug data into our code.

When something breaks in production, developers must go deeper than resource usage or business transaction fail rates – they need real-time insight and transparency into what the application is actually doing at the point that it's breaking – in production.

In order to pinpoint issues in production we believe developers need additional tooling which is actually closer to what they would use in their development/test environments. Developers need to see things like:

- stack trace + local variable visibility, at the exact point of failure or deadlock
- profiling information, when code is run against 'real production data'
- transactions, web & JDBC requests not just measured by time, but by memory consumed
- class loads/unloads & memory allocation (heap + non-heap) across time

We need a new generation of monitor, which not only provides core APM features, such as metrics and alerting, but also includes low-overhead production-grade tools to give access to detailed information needed to "deep-dive" & quickly figure out the hard stuff developers need to fix.

Before you reach for app logs next time, check out FusionReactor.

WRITTEN BY DAVID TATTERSALL
CEO - INTERGRAL GMBH – MAKERS OF **FUSIONREACTOR**

PARTNER SPOTLIGHT

FusionReactor BY INTERGRAL GMBH

FusionReactor goes beyond traditional APM tools to give you unrivaled insight into how your Java code performs and executes in production environments.

CATEGORY
APM for Developers

NEW RELEASES
3 Months

OPEN SOURCE?
No

CASE STUDY

Bullhorn provides cloud-based CRM solutions for relationship-driven businesses. Its zero-click data capture technology and relationship intelligence gives companies what they need, from insight to action, to win new customers and keep them happy.

"FusionReactor allows our team at Bullhorn to respond to issues quickly before they become customer impacting. It's short polling interval gives us a needed edge when it comes to ensuring an excellent experience for our customers. FusionReactor outshines the competition with its rapid response time, small hardware footprint and low total cost of ownership"

- Brad Witherell, Manager, Systems Engineering and Administration Bullhorn

STRENGTHS

- Deep dive Java APM
- Low-impact debugger and profiler (designed for production use)
- Full featured monitoring capability - alerting & metric analysis
- Crash protection capability to increase application resilience
- Seamless integration to run alongside other APM tools
- Hybrid APM - available as On Premise and Cloud (optional)
- Low cost, yet highly functional

NOTABLE CUSTOMERS

- Auto Europe
- Bullhorn
- Hasbro
- Allianz
- Primoris Services
- InVision

BLOG **blog.fusion-reactor.com** | TWITTER **@Fusion_Reactor** | WEBSITE **fusion-reactor.com**

Why You Need to Know Your Pages' Conversion Impact Score

BY TAMMY EVERTS
SENIOR RESEARCHER AND EVANGELIST, **SOASTA**

QUICK VIEW

01
Not all web pages are created equal. People react differently to slowdowns on different pages in the transaction path.

02
Knowing your pages' load times is just a first step. You need to correlate load time with other metrics that are meaningful to your business.

03
Conversion Impact Scoring keeps you from wasting limited performance optimization resources on the wrong pages.

04
Every site is different. Page groups that have high Conversion Impact Scores for another retailer may not generate the same scores for you. That's why you need to use your own user data.

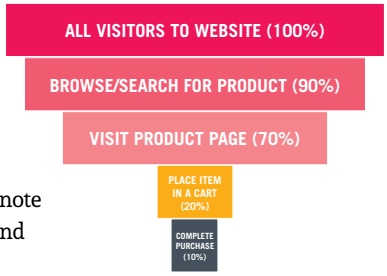
There's a widely held belief that the only people who need to care about conversions are people in sales and business development. Wrong. Conversions are the lifeblood of your business. If you touch your company's website in any way—be it design, marketing, or development—then your actions have an impact on conversions. You need to understand what that impact is.

I'm going to explain how to determine which pages you should focus on optimizing in order to increase conversions and, ultimately, deliver the highest ROI.

WHAT IS A CONVERSION?

A conversion is what happens when a person who's browsing a site converts to being a user or buyer of the service or product that site offers. So if you're a SaaS vendor, a conversion happens when a person signs up to use your service—or if you're an e-commerce shop, when a person buys something. Conversions can also include actions like signing up for a newsletter or making a donation.

The **conversion funnel** is the start-to-finish path that a user takes when they convert from browsing to buying/downloading/etc. A conversion funnel for an ecommerce site might look something like this (note that percentages are arbitrary and extremely optimistic):

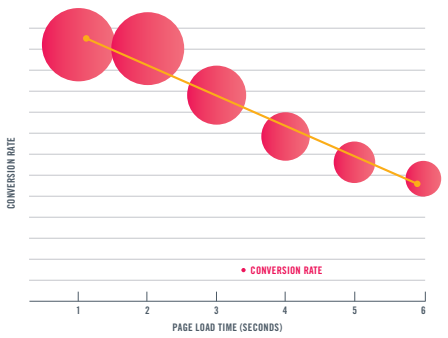


Conversion rate is the percentage of total user sessions that result in a conversion. In the conversion funnel graphic above, the conversion rate is the number of people who completed a purchase: 10%. Conversion rates are typically in the 2-5% range. Anything higher than that is amazing.

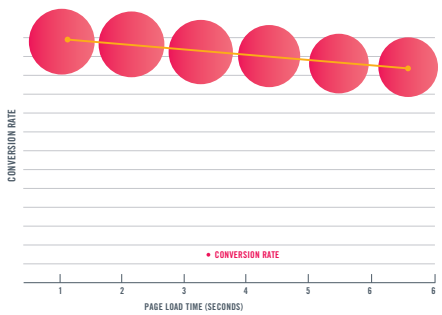
For a site that does hundreds of thousands of dollars worth of transactions in a day, even tiny changes in conversion rate—such as increasing from 2.1% to 2.2%—can have a huge impact on revenue.

PERFORMANCE SLOWDOWNS AFFECT CONVERSIONS DIFFERENTLY ON DIFFERENT PAGES

When pages get slower, conversion rates suffer. But some types of pages suffer more than others. For example, on retail sites, slow "browse" pages have a greater negative impact on performance than slow "checkout" pages.



Here you can see that, for one ecommerce vendor, the conversion rate shrank by about 50% when the load time for "browse" pages increased from 1 to 6 seconds *(right, top)*:



For the same retailer, the impact on conversion rate was much less when checkout pages degraded in speed *(right, bottom)*:

Looking at these two graphs side by side, you could be tempted to deduce that, because conversions were hurt more by slow "browse" pages than by slow "checkout" pages, the site owner should focus energy on optimizing the browse pages. This might be true—but it might not be true, too. This is where Conversion Impact Scoring comes in.

WHAT IS THE CONVERSION IMPACT SCORE?

The Conversion Impact Score is a relative score that ranks page groups by their propensity to negatively impact conversions due to high load times. For each page group, the Conversion Impact Score is calculated using the proportion of overall requests that are associated with that group, along with the correlation between its load times and number of conversions. The more negative the score, the more detrimental to conversions that high load times for that page group are, relative to the other page groups.

In other words, Conversion Impact Scoring answers this question: how much impact does the performance of this page have on conversions?

CASE STUDY: HOW TO USE CONVERSION IMPACT SCORING TO PRIORITIZE PERFORMANCE OPTIMIZATION

Now let's walk through how to use Conversion Impact Scores to make decisions about optimizing your pages.

In the table below, you can see the Conversion Impact Scores and load times for a set of page groups on a retail site. The second column represents the Conversion Impact Score for each page group, and the third column represents the median page load time for each group. For a complete graph of these details, click here.

The groups are ranked from those with the highest Conversion Impact Scores (such as product and category pages; in other words, pages viewed in the "browse" phase of the conversion funnel) to pages with the lowest scores (such as checkout and sign-in pages).

PAGE GROUP	RELATIVE CONVERSION IMPACT SCORE	MEDIAN FULL PAGE LOAD TIME (SECONDS)
Product Detail Page	-0.12	2.9
Category Browse 1	-0.085	3.0
Home	-0.08	3.8
Choose Your Country	-0.045	2.1
Shopping Bag	-0.01	2
Checkout – Send To	-0.005	4
Wishlist	-0.004	2.8
Checkout – Order Confirmation	-0.003	3.2
Account – SignIn	-0.0025	3.3

Some quick observations:

- Some of the fastest page groups—such as Shopping Bag and Wishlist—have relatively low Conversion Impact Scores. This means that page speed isn't a significant factor in how well these pages convert.
- The page groups with the highest Conversion Impact Scores—such as Product and Category pages—have acceptable load times in the 3-second range.
- The slowest group is Checkout – SendTo, followed by Home, Account – SignIn and Checkout – Order Confirmation.

Without knowing the Conversion Impact Scores for these page groups, you might focus on optimizing pages according to how slow they load. Looking at load time, this is the order in which you'd prioritize fixing these groups:

1. Checkout – SendTo
2. Home
3. Account – SignIn
4. Checkout – OrderConfirmation
5. Category Browse 1

Now here's how some of these assumptions are incorrect:

ASSUMPTION #1: PRIORITIZING THE CHECKOUT – SENDTO PAGE GROUP BECAUSE IT'S THE SLOWEST

If you looked only at page load times, you might believe that you need to prioritize the Checkout – SendTo group because its performance is dramatically poorer than the other groups. But if you knew its Conversion Impact Score, you'd realize that page speed doesn't have much impact on conversion rate, so making this group faster wouldn't be the best use of your limited optimization resources.

ASSUMPTION #2: TACKLING THE ACCOUNT – SIGNIN PAGE GROUP NEXT

Also, if you were to look exclusively at load times, you might believe that when you're done with optimizing the Checkout – SendTo group, you should focus next on addressing performance issues on the Account – SignIn group. While these pages have a high enough Conversion Impact Score that they merit addressing, they shouldn't rank high up on your list.

ASSUMPTION #3: NOT WORRYING ABOUT THE CATEGORY BROWSE 1 AND PRODUCT DETAIL PAGE GROUPS BECAUSE THEY SEEM RELATIVELY FAST

Still looking solely at load times, you might also guess that, because these pages look fairly speedy, you don't need to worry about them. This is where you'd make your biggest mistake. Because these groups have the highest Conversion Impact Scores, they have the potential to deliver the most benefit to you if you make them faster.

CONCLUSION

Knowing the Conversion Impact Scores for this set of page groups, this is the order in which you might actually want to prioritize their optimization to give you the best ROI:

1. Home
2. Category Browse 1
3. Product Detail Page
4. Choose Your Country
5. Shopping Bag

As senior researcher and evangelist at SOASTA, **TAMMY EVERTS** studies the technical, business, and human sides of web/application performance and shares her findings via countless blog posts, presentations, case studies, articles, and reports. She manages the popular industry blog the Performance Beacon.

SOLUTIONS DIRECTORY

This directory of monitoring, hosting, and optimization services provides comprehensive, factual comparisons of data gathered from third-party sources and the tool creators' organizations. Solutions in the directory are selected based on several impartial criteria, including solution maturity, technical innovativeness, relevance, and data availability.

PRODUCT NAME	PRODUCT TYPE	FREE TRIAL	HOSTING	WEBSITE
Akamai Ion	CDN, Network & Mobile Monitoring & Optimization, FEO	Traffic (15TB) and other limits	SaaS	akamai.com
Alertsite by Smartbear Software	APM, Synthetic Monitoring, Infrastructure Monitoring, Middleware Monitoring	Available by request	On-premise or SaaS	smartbear.com/product/alertsite/overview/
Apica Systems	APM, Infrastructure Monitoring	Limited by usage	SaaS	apicasystems.com
AppDynamics	APM, Mobile and Web RUM, Database Monitoring, Infrastructure Visibility	Free forever (Lite); 15-day free trial (Pro)	On-premise or SaaS	appdynamics.com
AppFirst	APM, Infrastructure Monitoring, ITOA	30 days	SaaS	appfirst.com
AppNeta APM Platform	APM, Synthetic Monitoring, Network Monitoring, ITOA, Real User Monitoring	Available by request	SaaS	appneta.com
AppNomic AppsOne	ITOA	Upon request	On-premise or SaaS	appnomic.com
Aternity	APM, ITOA, Real User Monitoring	Upon request	On-premise	aternity.com
BigPanda	ITOA, Alert Software	21 days	SaaS	bigpanda.io
BMC TrueSight Pulse	APM, Network Monitoring, ITOA, Database Monitoring	14 days	SaaS	bmc.com/it-solutions/truesight.html
BrowserStack	FEO	Limited by usage	SaaS	browserstack.com
CA App Synthetic Monitor	APM, Synthetic Monitoring	Free Trial	SaaS	ca.com/us/products/ca-app-synthetic-monitor.html
CA Mobile App Analytics	Mobile APM	Free version available	SaaS	ca.com/us/products/ca-mobile-app-analytics.html
CA Unified Infrastructure Management	Infrastructure Monitoring	Free Trial	On-premise	ca.com/us/products/ca-unified-infrastructure-management.html
Catchpoint Suite	Synthetic, RUM, UEM	14 days	On-premise or SaaS	catchpoint.com/products/
Censum by jClarity	JVM Garbage Collection Optimization	7 days	SaaS w/ on-premise option	jclarity.com

PRODUCT NAME	PRODUCT TYPE	FREE TRIAL	HOSTING	WEBSITE
Circonus	Infrastructure Monitoring, ITOA	Free tier available	SaaS	circonus.com
CloudFlare	CDN, Network, Mobile, and Web Monitoring and Optimization, FEO	Free tier available	CDN	cloudflare.com
Correlsense SharePath	APM, Network Monitoring, Middleware Monitoring	Upon request	On-premise or SaaS	correlsense.com
CoScale	APM, Infrastructure Monitoring, ITOA, Real User Monitoring	30 days	SaaS	coscale.com
Datadog	Performance Metrics Integration and Analysis	14 days	SaaS	datadoghq.com
Dotcom Monitor	APM, Infrastructure Monitoring, FEO	30 days	SaaS	dotcom-monitor.com
Dyn	Infrastructure Monitoring, Network Monitoring, ITOA	7 days	On-premise	dyn.com
Dynatrace Application Monitoring	APM, ITOA	30 days	On-premise	dynatrace.com/en/application-monitoring/
Dynatrace Data Center RUM	RUM (web and non-web), synthetic, ITOA	Demo on request	On-premise	dynatrace.com/en/data-center-rum/
Dynatrace Ruxit	APM (cloud-native optimized) + AI	30 days / 1000 hours	On-premise or SaaS	dynatrace.com/en/ruxit/
Dynatrace Synthetic	Synthetic monitoring, managed load testing	Demo on request	SaaS	dynatrace.com/en/synthetic-monitoring/
Dynatrace UEM	Real user monitoring (web and mobile)	30 days	On-premise	dynatrace.com
eg Innovations Monitors	APM, Infrastructure Monitoring, ITOA	14 days	SaaS	eginnovations.com
Evolven	ITOA	Upon request	On-premise	evolven.com
Extrahop Networks	ITOA	Free tier available	SaaS	extrahop.com
F5 Big IP Software	APM, Network Monitoring	30 days	On-premise or SaaS	f5.com
Foglight by Dell	APM, Database Monitoring, RUM, ITOA	Available by request	On-premise	software.dell.com
Fusion Reactor	Java server monitor, production debugging, crash protection	14 days	On-premise	fusion-reactor.com
HPE APM	APM, ITOA, Real User Monitoring	30 days	On-premise	hp.com
IBM API Connect	API Management Platform	Free tier available	On-premire or SaaS	ibm.com/software/products/en/api-connect
IBM Application Performance Management	APM, Infrastructure Monitoring, Real User Monitoring	30 days	On-premise or SaaS	ibm.com/software/products/en/ibm-application-performance-management
Idera SQL Diagnostic Manager	DB monitoring	14 days	SaaS	idera.com
Idera UpTime Software	APM, Infrastructure Monitoring, Network Monitoring	14 days	SaaS	idera.com

PRODUCT NAME	PRODUCT TYPE	FREE TRIAL	HOSTING	WEBSITE
Illuminate by jClarity	JVM Performance Diagnosis and Optimization	14 days	SaaS w/ on-premise option	jclarity.com
Impact by Cedexis	Infrastructure Monitoring, FEO, ITOA	Upon request	SaaS	cedexis.com
Inetco Insight	APM, Middleware Monitoring	Upon request	On-premise	inetco.com
Infovista 5view Applications	APM, Network Monitoring, Real User Monitoring	Upon request	On-premise	infovista.com
JenniferSoft	APM	14 days	On-premise	jennifersoft.com
Keynote Platform by Dynatrace	Mobile APM (Synthetic Monitoring, Test Automation)	7 days	SaaS	keynote.com
Librato	Performance Metrics Integration and Analysis	30 days	SaaS	librato.com
Logentries	Log Management and Analytics	30 days; free tier available	SaaS	logentries.com
Loggly	Log Management and Analytics	30 days	SaaS	loggly.com
LogMatrix NerveCenter	ITOA, APM, Infrastructure Monitoring, Network Monitoring, Database Monitoring	Available by request	On-premise	logmatrix.com
ManageEngine Applications Manager	APM, Network Monitoring, Infrastructure Monitoring	Available by request	On-premise	manageengine.com
Microsoft System Center 2012	APM	180 days	On-premise	microsoft.com
Moogsoft	Performance Metrics Integration, Analysis, and Response	Available by request	On-premise or SaaS	moogsoft.com
Nagios XI	APM, Infrastructure Monitoring, Network Monitoring, FEO, ITOA	Open source	On-premise	nagios.com
Nastel Autopilot	APM, Infrastructure Monitoring, FEO, Middleware Monitoring	Upon request	SaaS	nastel.com
NetScout nGeniusOne	APM, Network Monitoring, ITOA	Upon request	On-premise	netscout.com
Netuitive	APM, Infrastructure Monitoring, ITOA	21 days	SaaS	netuitive.com
Neustar Website Monitoring	FEO	30 days	SaaS	neustar.biz
New Relic APM	APM, Database Monitoring, Availability & Error Monitoring, Reports, Team Collaboration, Security	Free tier availablel 14-day Pro trial	SaaS	newrelic.com/application-monitoring
op5 Monitor	APM, Infrastructure Monitoring, Network Monitoring, FEO, ITOA	Free tier available	SaaS	op5.com
OpsGenie	Alert Software	Upon request	On-premise	opsgenie.com
OpsView	APM, Network Monitoring, ITOA	30 days	On-premise	opsview.com

PRODUCT NAME	PRODUCT TYPE	FREE TRIAL	HOSTING	WEBSITE
PA Server Monitor	Infrastructure Monitoring, Network Monitoring	30 days	On-premise	poweradmin.com
PagerDuty	ITOA, Alert Software	14 days	SaaS	pagerduty.com
Pingdom	APM, FEO	30 days	SaaS	pingdom.com
Rackspace Monitoring	Cloud monitoring	Included with cloud account	SaaS	rackspace.com/cloud/monitoring
Riverbed SteelCentral	APM, Infrastructure Monitoring, Network Monitoring, ITOA	30-90 days	On-premise	riverbed.com
SauceLabs	FEO, Automated Web and Mobile Testing	14 days	SaaS	saucelabs.com
ScienceLogic Platform	APM, Infrastructure Monitoring, Network Monitoring	Upon request	SaaS	sciencelogic.com
SevOne	Infrastructure Monitoring, Network Monitoring	Upon request	SaaS	sevone.com
SIEM by AccelOps	ITOA, Network Monitoring	30 days	SaaS	accelops.com
Site24x7 by ManageEngine	APM, FEO, Infrastructure Monitoring, Network Monitoring	Limited by usage	SaaS	site24x7.com
Soasta Platform	Real User Monitoring, Load Testing	Up to 100 users	SaaS	soasta.com
Solarwinds Network Performance Monitor	Network Monitoring, ITOA, Database Monitoring, Log Management	30 days	On-premise	solarwinds.com
SpeedCurve	FEO, ITOA	None	SaaS	speedcurve.com
Spiceworks	Network Monitoring, ITOA	Free	On-premise	spiceworks.com
Stackify	APM, Network Monitoring, Database Monitoring, ITOA	Upon request	SaaS	stackify.com
TeamQuest	ITOA	Upon request	On-premise	teamquest.com
Telerik Analytics	End-User Monitoring and Analytics	Free	On-premise	telerik.com
ThousandEyes	Network Monitoring, ITOA	15 days	SaaS	thousandeyes.com
TINGYUN App	APM, FEO, Real User Monitoring	Available by request	SaaS	tingyun.com
VictorOps	Alert Software	14 days	On-premise	victorops.com
Zabbix	Network Monitoring	Open source	On-premise	zabbix.com
Zenoss Service Dynamics	Infrastructure Monitoring, Network Monitoring	Open source version available	On-premise or SaaS	zenoss.com

DIVING DEEPER

INTO PERFORMANCE + MONITORING

TOP 10 #PERFORMANCE TWITTER FEEDS

 @Souders

 @brendangregg

 @tameverts

 @paul_irish

 @bbinto

 @mdaoudi

 @ChrisLove

 @firt

 @Perf_Rocks

 @duhroach

DZONE PERFORMANCE-RELATED ZONES

Performance Zone
dzone.com/performance

Scalability and optimization are constant concerns for the developer and operations manager. The Performance Zone focuses on all things performance, covering everything from database optimization to garbage collection, tool and technique comparisons, and tweaks to keep your code as efficient as possible.

DevOps Zone
dzone.com/devops

DevOps is a cultural movement, supported by exciting new tools, that is aimed at encouraging close cooperation within cross-disciplinary teams of developers and IT operations/system admins. The DevOps Zone is your hot spot for news and resources about Continuous Delivery, Puppet, Chef, Jenkins, and much more.

Java Zone
dzone.com/java

The largest, most active Java developer community on the web, with news and tutorials on Java tools, performance tricks, and new standards and strategies that keep your skills razor sharp.

TOP PERFORMANCE REFCARDZ

GETTING STARTED WITH
Real User Monitoring
bit.ly/dz-userm

Java Performance Optimization
bit.ly/dz-javaperf

Scalability & High Availability
bit.ly/dz-scale

TOP PERFORMANCE WEBSITES

Planet Performance
perfplanet.com

ResponsiveDesign.is
responsivedesign.is

Brendan Gregg's Blog
brendangregg.com/blog

TOP SPEED TEST TOOLS

webpagetest.org

tools.pingdom.com/fpt

developers.google.com/speed/pagespeed/insights/

gtmetrix.com

GLOSSARY

ACTIVE MONITORING Also known as *synthetic monitoring*, this is a type of website monitoring where scripts are created to simulate an ordered series of actions that an end-user might take (as opposed to comparatively atomic functional or integration tests). Tests overall site functionality and response time and helps identify any problems that hinder overall site performance.

APPENDER In logging systems, specifies destination, message format, behavior (non/blocking, response timeouts, retry intervals, exception handling), accept/reject filters, compression details, etc.

APPLICATION-LAYER PROTOCOL NEGOTIATION (ALPN) An extension of Transport Layer Security (TLS) protocol negotiation that helps client and server figure out, within the TLS handshake, which application-layer protocols to use. Handles HTTP/1.1 vs. HTTP/2 selection but is also app-layer protocol indifferent.

APPLICATION PERFORMANCE MONITORING (APM) Combines metrics on all factors that might affect application performance (within an application and/or web server, between database and application server, on a single machine, between client and server, etc.); usually (but not always) higher-level than stack trace.

AVAILABLE PARALLELISM In terms of Big O notation: work (relation between run steps and input count) divided by depth (number of branches in execution tree). Intuitively: how much time we can save by running an algorithm in parallel, discounted by the delay introduced by splitting up the workload.

BIG-O NOTATION Describes the rate of change in runtime steps required by an algorithm given a specified change in input count. Used to capture efficiency of an algorithm. O(n) will scale linearly, O(n2) will become quadratically slower, and O(1) will not lose any efficiency over time.

BINARY FRAMES The basic unit of communication in HTTP/2 (constructed by analogy to frames in link-layer protocols); replaces human-readable header+body in HTTP/1.1 request/response instances (where headers are not compressed).

BOTTLENECK Occurs when an entire system is slowed by one key component that has reached capacity; the result is that non-bottlenecked system components waste resources waiting.

CIRCUIT BREAKER A wrapper around a resource to check the availability of that resource and return an error message to a requesting process if the resource is unavailable; prevents cascading failures (e.g., if additional resources are waiting on a requesting resource that receives an 'unavailable' message, then the requesting resource can enter fallback mode).

CONTENT DELIVERY NETWORK (CDN) Geographically and topologically distributed servers that cache content (often high-bandwidth static content, like videos, documents, or other large binaries) to minimize unnecessary transport and backbone overload.

CONVERSION FUNNEL A start-to-finish path that a user follows when they convert from looking/browsing to downloading or making a purchase.

CONVERSION IMPACT SCORE A measure of how much something (e.g., a page load time increase of 500ms) affects conversion (see above).

DESIGN PATTERN A reusable solution to commonly recurring problem; more abstract than a best practice, more concrete than a design principle (object-oriented examples: Iterator, Factory, Observer).

FLAW OF AVERAGES A phrase coined by statistician Sam Savage to capture the notion that serious misrepresentation of data often occurs when averages are used to represent uncertain outcomes.

GARBAGE COLLECTION A part of automatic memory management; the process of reclaiming the memory reserved for objects that are no longer in use.

KEY PERFORMANCE INDICATOR (KPI) A set of indicators to measure data or performance against a particular set of standards or requirements

LATENCY The time delay between an input and the desired output in a software system.

MICROSERVICES An application deployment model consisting of small, loosely coupled services that each perform a single function according to the domain's bounded contexts; sometimes seen as "SOA done right," or "another version of the UNIX philosophy."

MULTIPLEXING A method used to send more than one message or data stream (which in practice often means bidirectionally) in the form of one complex signal over a single link.

RECURSIVE FUNCTION A system of solving a problem in which the solution depends on breaking down the problem and solving smaller instances of it. This function can keep looping back to the beginning of itself until all the problems are solved.

SERVER PUSH A method of information delivery on the Web that is originated by the publisher/information server versus the client/information user (the usual process).

THREAD POOL A number of threads reserved in advance; avoids ad-hoc task creation overhead, thread-linked spin-up expense (e.g., if stopping a thread would require closing a socket that will be reopened soon), and resource bottlenecks caused by OS resource management.

THROTTLING A mechanism to deliberately regulate the rate at which data is transferred or processed.

TRANSPORT LAYER SECURITY (TLS) A protocol designed to protect the privacy between communicating applications and their users on the Internet. Usually considered to deprecate SSL v3.0 (which has serious vulnerabilities).

USER DATAGRAM PROTOCOL (UDP) A lightweight, connectionless alternative to TCP, this is a messaging protocol in which computer applications can send messages across an IP network without needing prior communication to set up data paths. It is also used to set up loss-tolerating and low-latency internet application connections.