



THE DZONE GUIDE TO PERFORMANCE & MONITORING

2015 EDITION

BROUGHT TO YOU IN PARTNERSHIP WITH

APPDYNAMICS



NGINX

SCASTA

Dear Reader,

In CS101 they ask: how much computation is required to produce that output, given this input? In “The Real World” they tell you: *your application is too darn slow!* You’re dealing with business processes and impatient people, not mathematical functions. Big-O only tells you what an abstract tape is doing. Users just tell you they’re not happy.

Well, maybe there’s a bottleneck. Find the slowest thing and fix it. Maybe as a developer you can’t fix it on your own—iron can only spin so fast, CPUs and GPUs can only execute so many processes per second, and maybe the budget says we’re stuck with this service level for now.

But as client machines grow more capable, network latency decreases, and the metal itself keeps riding Moore’s Law to the limits of two- or three-dimensional silicon. The chances that you can do *something* on your own keep increasing. Trouble is: as capabilities increase (especially virtualization) and systems become more complex, it gets proportionally more difficult to find the bottleneck and to fix it.

So we’ve worked with hundreds of professionals in various development and monitoring roles to publish our new 2015 DZone *Guide to Performance & Monitoring*. Our goal: help you write more performant applications; learn where your applications are slowing down; and fix those problems quickly.

We’ve covered tools and techniques for application monitoring, infrastructure monitoring, IT Operational Analytics, CDN monitoring, load testing, client-side web performance, and more. We offer tips for monitoring in virtualized environments; introduce a load of W3C-generated web performance APIs; and suggest ways to find whether software or hardware is the problem. We’ve even tried to figure out whether monitoring is having any success catching up to DevOps practices (spoilers: there’s room for improvement; read on to learn where).

We hope the 2015 DZone *Guide to Performance & Monitoring* will help you write faster code, find problems faster, and fix problems more easily. Enjoy! And as always, feel free to let us know what you think.



JOHN ESPOSITO

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

Table of Contents

- 3 SUMMARY & KEY TAKEAWAYS**
- 4 KEY RESEARCH FINDINGS**
- 8 DOES MONITORING STILL SUCK?**
BY DAVID GILDEH
- 11 TRY OPTIMIZING MEMORY CONSUMPTION FIRST**
BY PETER LAWREY
- 14 MECHANICAL SYMPATHY: UNDERSTANDING THE HARDWARE MAKES YOU A BETTER DEVELOPER**
BY DAVE FARLEY
- 17 DIVING DEEPER INTO PERFORMANCE & MONITORING**
- 20 THE WEB PERFORMANCE APIS REFERENCE**
BY BARBARA BERMES
- 22 THE SCALE OF COMPUTING LATENCIES**
INFOGRAPHIC
- 26 CHASING PERFORMANCE PHANTOMS: LOOKING BEYOND “WHY THE SLOWEST THING IS SLOW”**
BY JON C. HODGSON
- 32 HOW TO DEFINE PERFORMANCE REQUIREMENTS**
BY NIKITA SALNIKOV-TARNOVSKI
- 36 FRONT-END OPTIMIZATION CHECKLIST**
- 37 SOLUTIONS DIRECTORY**
- 43 GLOSSARY**

Credits

EDITORIAL

John Esposito
research@dzone.com
EDITOR-IN-CHIEF

Jayashree Gopal
DIRECTOR OF RESEARCH

Mitch Pronschinske
SR. RESEARCH ANALYST

Benjamin Ball
RESEARCH ANALYST

Matt Werner
MARKET RESEARCHER

John Walter
CONTENT CURATOR

Ryan Spain
CONTENT CURATOR

BUSINESS

Rick Ross
CEO

Matt Schmidt
PRESIDENT & CTO

Kellet Atkinson
GENERAL MANAGER

Alex Crafts
sales@dzone.com
VP OF SALES

Matt O'Brian
DIRECTOR OF BUSINESS
DEVELOPMENT

Chelsea Bosworth
MARKETING ASSOCIATE

Chris Smith
PRODUCTION ADVISOR

Brandon Rosser
CUSTOMER SUCCESS
ADVOCATE

Jillian Poore
SALES ASSOCIATE

Jim Howard
SALES ASSOCIATE

ART

Ashley Slate
DESIGN DIRECTOR

Special thanks to our topic experts Alexander Podelko, Alex Rosemblat, David Farley, Sergey Chernyshev, Colin Scott, and our trusted DZone Most Valuable Bloggers for all their help and feedback in making this report a great success.

WANT YOUR SOLUTION TO BE FEATURED IN THIS OR 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.

Summary and Key Takeaways

THERE ARE FEW THINGS MORE FRUSTRATING THAN A SLOW website or application. Performance is not just a development concern; it is also a component of design because the user experience and the usability of features depend heavily on response time. A faster application makes the experience feel lighter and simpler, whereas slow applications feel cumbersome and untrustworthy [1]. Many companies have gathered user data suggesting that it only takes a few seconds of loading time for users to abandon a site or lose faith in a brand [2]. While performance is the most important aspect of software besides its primary objective to work properly, many organizations treat it as an afterthought. DZone's *Guide to Performance & Monitoring* will help you take a hard look at your software design and monitoring strategies so that you can build a performant foundation for your applications and better identify the root causes of performance problems. The resources in this guide include:

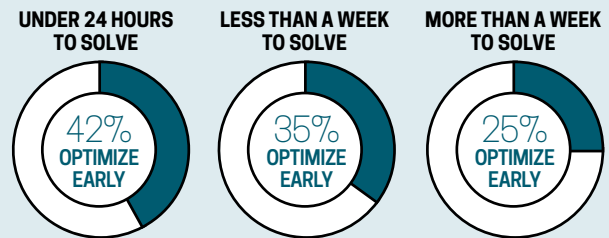
- Performance statistics and strategies from 400+ IT professionals.
- An overview of how modern companies are handling application monitoring.
- Resources on harnessing new W3C web performance API standards.
- Step-by-step processes for finding root causes of performance problems in virtualized environments.
- A catalogue of APM, ITOA, FEO/WPO, network monitoring, and infrastructure monitoring tools.

“AVOID PREMATURE OPTIMIZATION” IS A MYTH

Premature optimization can mean different things depending on whom you ask, but the value of building performant code into your applications early in the development process is clear. There is a definite correlation between respondents who said they build performance into their applications from the start, and respondents who said that they solve performance problems faster. Along with this data, we have the expert advice of Dave Farley, who has worked with high performance systems for many years, and was the co-author of the seminal book, [“Continuous Delivery.”](#) In Farley's contribution to this guide, he agrees that the old adage “avoid premature optimization” is bad advice, and

that developers should always focus on performance early in the design process. Early optimization is only harmful if you optimize without knowing where the true bottleneck is occurring.

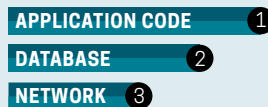
01. EARLY PERFORMANCE OPTIMIZATION FILTERED BY AVG TIME TO SOLVE PERFORMANCE PROBLEMS



PERFORMANCE PROBLEMS ARE A CONSTANT BATTLE

68% of respondents said they had run into performance problems in their software in the last month, and 32% say they had to solve a problem in the last week. On the operations side, 45% said they have had to solve a performance problem in their infrastructure in the last month, and 18% said they have had to solve one in the last week. Many factors can increase the frequency of performance issues, but as applications and infrastructure grow in size and user load, increased performance problems become inevitable. Our data shows that performance is a frequent concern for nearly every software company.

02. WHERE THE MOST FREQUENT PERFORMANCE PROBLEMS OCCUR



03. WHERE THE HARDEST-TO-FIX PERFORMANCE PROBLEMS OCCUR



APPLICATION CODE CAUSES THE MOST FREQUENT PERFORMANCE PROBLEMS; NETWORK PERFORMANCE PROBLEMS ARE THE HARDEST TO FIX

Application code, database, and network were ranked one, two, and three respectively by respondents as the areas where they most frequently encountered performance problems in their technology stack. The hardest-to-fix problems came from the network, while application code and the database were ranked second and third respectively. When the rankings were filtered by the average time taken to solve performance issues, we found that respondents who took longer to fix performance problems ranked application code as the most difficult area to fix. The respondents that took less than 24 hours on average to solve performance problems ranked application code as only the third hardest area to fix. This could suggest that the faster problem solvers have smaller, simpler, or more elegant codebases.

[1] <http://www.amazon.com/gp/product/0262134721/>

[2] <http://www.adweek.com/socialtimes/study-consumers-will-abandon-apps-with-greater-than-six-second-load-times-infographic/616318>

Key Research Findings

More than 400 IT professionals responded to DZone's 2015 Performance & Monitoring Survey. Here are the demographics for this survey:

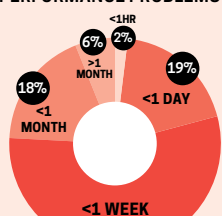
- **Developer (49%) and Architect (24%) are the most common roles.**
- **65% of respondents come from large organizations (100 or more employees) and 35% come from small organizations (under 100 employees).**
- **The majority of respondents are headquartered in Europe (43%) or the US (35%).**
- **Over half of respondents (70%) have over 10 years of experience as IT professionals.**
- **A large majority of respondents' organizations use Java (81%). JavaScript/Node.js is the next highest (61%).**

MOST PERFORMANCE PROBLEMS ARE SOLVED IN A WEEK OR LESS

From the detection of a performance problem to committing a fix to production, a majority of respondents said they take less than a week on average (55%) to solve a performance problem. 19% said they take less than a day, and 2% said they often do it in under an hour. While there were no significant correlations between the speed of performance fixes and the number of servers, organization size, or user load, other factors did correlate with speed of performance fixes. There is a 10% higher proportion of Java developers in the segments that took longer than a day to fix performance problems. This might have a connection to the frequency of memory leaks in some Java

technologies, because we found that memory problems are more common among respondent segments that took longer to fix performance problems. We also found that faster performance problem solving teams are more likely to use only free open source tools and to have monitoring tools that aggregate developer and operational metrics into one combined view.

01. AVERAGE TIME TO SOLVE PERFORMANCE PROBLEMS



FINDING ROOT CAUSE IS THE HARDEST PART OF PERFORMANCE MANAGEMENT

Taking a general look at performance management, the most time-consuming part of fixing performance issues is finding the root cause (score: 2114), according to respondent rankings. Closely behind is the task of reproducing the performance issue (score: 2068), another well known pain point for most debugging efforts. Collecting and interpreting various metrics (score: 1836) is the third most time-consuming task.

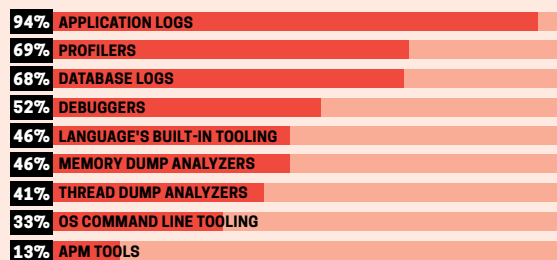
02. MOST TIME CONSUMING PART OF SOLVING PERFORMANCE PROBLEMS

- 1 FINDING THE ROOT CAUSE OF THE ISSUE
- 2 REPRODUCING THE PERFORMANCE ISSUE
- 3 COLLECTING & INTERPRETING VARIOUS METRICS

MOST PERFORMANCE PROBLEMS ARE PINPOINTED THROUGH SIMPLE APPLICATION LOG OBSERVATION

Respondents were asked to give a percentage breakdown of how often various methods discovered their performance issues. The highest average percentage came from monitoring tools (35%), with performance tests (load, stress, outage tests, etc.) and user support messages (email or social media) not far behind (28% and 25% respectively). We also discovered the most common tools for finding the root causes of performance problems. Application logs formed the primary basis (94%) for solving performance issues, while database logs (68%) and profilers (69%) were also common components for finding root causes. Surprisingly, only 13% said they commonly use APM tools to find the root cause of performance problems. The reason might be that the majority of performance problems are easy enough to solve by looking at application logs and deducing the root cause. However, it's also possible that the average performance problem solving speeds we're seeing could be improved if simple application log observation were not the primary tool for root cause analysis.

03. MOST COMMON TOOLS FOR FINDING ROOT CAUSE

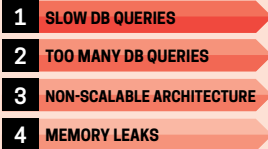


DATABASE ACCESS IS THE BIGGEST OFFENDER FOR PERFORMANCE PROBLEMS

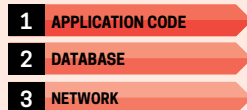
Slow database queries (rank: 1, score: 778) were the most common root cause for performance problems. Too many database queries (rank: 2, score: 533), non-scalable architecture (rank: 3, score: 490), and memory leaks (rank: 4, score: 412) were other common root causes. As we mentioned in the Key Takeaways section, the most frequent application problems came from application code, with

the database as a close second. Therefore, most performance issues are probably coming from application code that makes burdensome queries to the database, with the second most common problem possibly being inefficient code that runs too many database queries.

04. MOST COMMON PERFORMANCE ROOT CAUSES

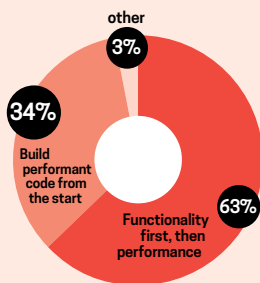


05. WHERE THE MOST FREQUENT PERFORMANCE PROBLEMS OCCUR



PERFORMANCE IS NOT USUALLY BUILT INTO APPLICATIONS FROM THE START

06. PERFORMANCE PRIORITIZATION

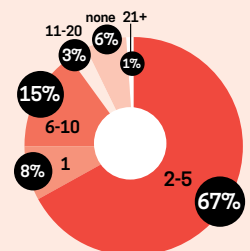


While premature optimization can be harmful if you try to fix the incorrect cause of a bottleneck, a software team can accrue a lot of technical debt if they don't build efficient code into the application from the start. For this reason, some might consider it a negative trend that 63% of respondents said they build application functionality first before worrying about performance.

METRICS ARE MORE OFTEN SCATTERED THAN COMBINED INTO SINGLE VIEWS

A strong majority (67%) of respondents use between 2-5 performance and infrastructure monitoring tools. Only 6% use none and 8% use one. The most common infrastructure monitoring strategy among respondents is to measure and report on multiple infrastructure performance indicators, but viewing each indicator separately (33%). Only 14% said they have these metrics unified into a single view. Surprisingly, 20% only monitor and report on infrastructure availability (statuses of servers, whether they're up or down), and 12% have no infrastructure monitoring strategy in place at all. 17% approach infrastructure monitoring from an end-to-end user experience perspective. The biggest challenge in infrastructure performance management is the time spent in firefighting and manual processes (56%). The second biggest challenge is a lack of actionable insights to proactively solve issues (44%).

07. # OF PERFORMANCE AND INFRASTRUCTURE MONITORING TOOLS USED

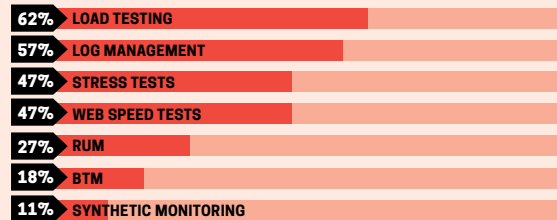


LOAD TESTS AND LOG MANAGEMENT ARE THE TOOLS OF CHOICE FOR PERFORMANCE MANAGEMENT

Load testing was the most common performance test that respondents used (62%), followed by log management/analysis (57%), stress tests (47%), and website speed tests (47%). Real user monitoring (27%), business transaction monitoring (18%),

and synthetic monitoring (11%), which are newer terms in the performance management industry, are at the lower end of usage. About one third of respondents are joining the IT operational analytics (ITOA) trend, with 34% saying they use an ITOA tool to run Big Data-level analytics on monitoring data.

08. MOST COMMON PERFORMANCE TOOL TYPES

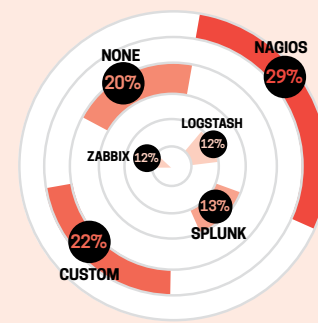


NAGIOS IS THE MOST USED MONITORING TOOL

With 29% of respondents using it, Nagios is the most commonly used monitoring tool among APM, infrastructure, and network monitoring tools. The second most common were homegrown custom solutions (22%). Splunk (13%), LogStash (12%), and Zabbix (12%) were other standouts. 20% said they use none of the 28 options we asked about. As respondents' max user load increased, Nagios and AWS

CloudWatch usage went up, and New Relic paid versions went up while New Relic Lite went down. LogStash had a jump in popularity around the 1K user load mark, and Pingdom and Splunk had jumps around the 100K mark. Respondents were less likely to choose "none" as their max user load increased. We also found that 57% of respondents have development and operations teams that share the same monitoring tools.

09. WHAT PERFORMANCE MANAGEMENT/ MONITORING TOOLS DO YOU USE?

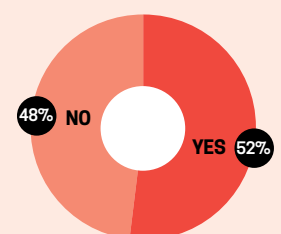


ABOUT HALF OF DEVELOPERS USE ONLY OPEN SOURCE

A slight majority of respondents (52%) said they only use free and open source tools for all of their performance management. This fits with the popularity of Nagios (a mature, open source tool) and custom tools in the survey data. David Gildeh, the founder of Dataloop.io, has also noticed this trend in the 60+ companies he surveyed about monitoring practices [1]. Many software companies want to create custom-fit performance management systems by building a "kit car" or APM toolchain out of several small open source projects that each perform very specific tasks. Whether or not these efforts are successful, in each case it takes a significant amount of time and planning to create these tools.

Developers are always trying to improve their systems, as evidenced by the 42% of respondents who said they are thinking of using a new performance monitoring tool in the next six months.

10. DO YOU ONLY USE FOSS PERFORMANCE TOOLS?



[1] DZone Guide to Performance & Monitoring 2015 Ed. pg. 8



DATADOG

SEE IT ALL IN ONE PLACE



Automatically track auto-scaling cloud hosts



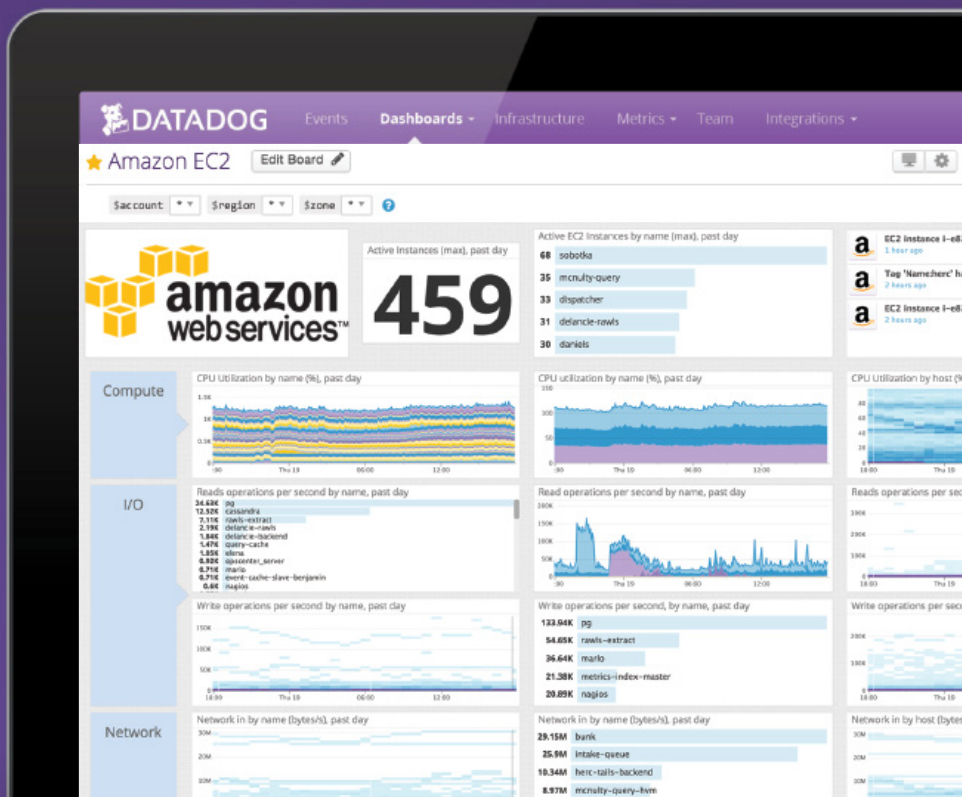
Monitor cloud and on-premise infrastructure



Alert team members about critical issues



Integrate with 100+ apps and services you're already using



A FEW OF OUR 100+ INTEGRATIONS



AWS



Google Cloud Platform



Docker



MongoDB



Redis



VMware



Python



Cassandra



Java



Google App Engine



Postgres



Github



Pagerduty



Nagios



PHP



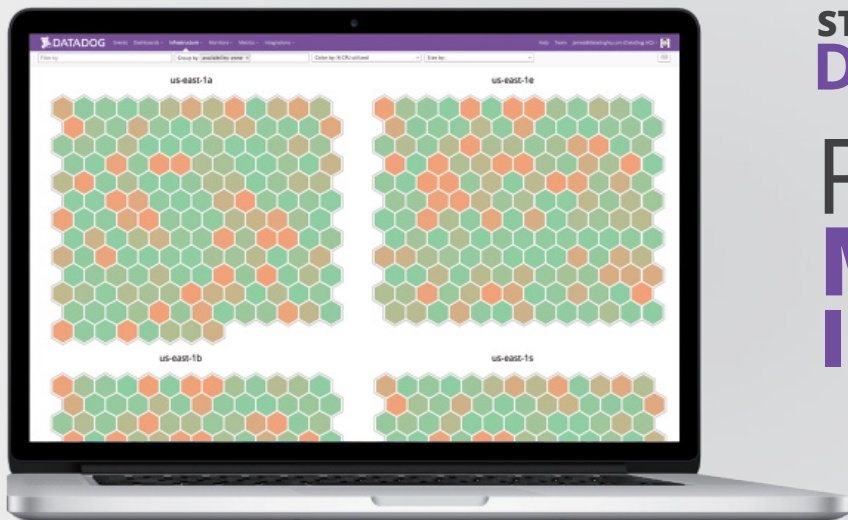
ElasticSearch



MySQL



New Relic



START YOUR FREE TRIAL AT
DATADOG.COM

READY TO MONITOR IN MINUTES.

Questions?
info@datadoghq.com

New Challenges in Infrastructure Monitoring

Today's applications often run in large, dynamic environments with hundreds or thousands of hosts. This is great for scalability and user experience, but it can be a nightmare for operations. The old ways of monitoring infrastructure just don't scale.

For starters, humans cannot reliably spot troublesome patterns in the massive amounts of data generated by modern infrastructures. What is more, the infrastructure itself changes all the time—it scales up or down in response to demand, and individual hosts routinely fail. It's not easy to tell which changes are problematic and which are not.

This type of infrastructure demands a modern monitoring platform. It must scale effortlessly along with your infrastructure and handle spikes in data volume. It must be able to discern the actionable from the routine. And it must minimize the false-positive alerts that fatigue ops teams and cause important issues to be overlooked.

The monitoring platform must also collect enough information to enable you to quickly diagnose the root cause of a problem, which means that your monitoring platform should interact with all your important technologies via vendor-supported integrations. It should offer both granular data collection for detailed analysis and long-term data retention for trend detection. And it should give you total visibility into all of your hosts with the ability to filter the hosts viewed by specific attributes.

Finally, since your monitoring platform will serve as your operational nerve center, it must help your team efficiently share information about potential issues. Annotated dashboards and seamless integrations with chat programs help facilitate quick, unambiguous team communication.

A modern monitoring platform must be able to scale effortlessly along with your infrastructure and to discern the actionable from the routine.

With these principles in mind, we built Datadog to serve the evolving needs of customers such as Netflix, Facebook, and Spotify. Our experience has been fed back into our platform so that today, world-class monitoring is available to companies of every size, working at any scale.



WRITTEN BY ALEXIS LÊ-QUÔC
CTO & CO-FOUNDER,
DATADOG

Datadog by Datadog



Datadog is the leading service for cloud-scale monitoring that bridges together data from over 100 commonly used tools and services.

CATEGORY

Infrastructure Monitoring

FEATURES

- 100+ integrations out of the box
- Data retained at 1-second granularity for an entire year
- Single-view metrics aggregation

CUSTOMERS

- | | | | |
|-----------|-----------|-----------|------------|
| • Netflix | • Zendesk | • Adobe | • Facebook |
| • Airbnb | • Spotify | • Samsung | • EA |

LANGUAGE SUPPORT

- C#
- Java
- Ruby
- Python
- PHP
- Go
- JavaScript
- SQL
- 4 others

ALERTS

- Threshold alerts
- Limit alerts
- Status alerts
- Conditional alerts
- Alert scheduling

CASE STUDY

SimpleReach provides real-time visibility and historical reporting into how content performs across metrics like reach, engagement, and social activity. Their platform runs within an AWS environment and as the infrastructure scaled the team was spending an increasing amount of time tracking and comparing performance metrics when updates occurred. They needed a monitoring tool to improve staff productivity by providing developers with a workflow-oriented operational monitoring system. Datadog provides developers early insight into how their changes are likely to impact other systems' performance by correlating seemingly disparate events and metrics which has dramatically improved productivity for the developers and operations team.

BLOG datadoghq.com/blog

TWITTER [@datadoghq](https://twitter.com/datadoghq)

WEBSITE datadoghq.com

Does Monitoring Still Suck?

BY DAVID GILDEH

In 2011, several champions of the DevOps movement started a Twitter campaign with the hashtag “#monitoringsucks.” Today, I wanted to revisit this sentiment in conjunction with a study I was doing before my team and I launched Dataloop.IO, a new monitoring tool for online services. We wanted to ensure that our new tool was solving real problems that other companies were facing, such as the monitoring pains we had experienced at our previous company, Alfresco.

We interviewed over 60 other online services, ranging in size from huge online services like the BBC to smaller startups in both London and the US. We focused only on online services and their specific needs, so our results are specific to the online services sector. Most services were running on public cloud infrastructure (like AWS) and fully committed to a DevOps approach.

Since we were trying to figure out how to develop a better monitoring tool, we needed to answer two questions. First, we needed to know what monitoring tools companies were currently using. Second, we needed to see how many servers these companies were monitoring with these tools. Finally, we wanted to see how these two variables related, so that we could get a sense of which tools were solving (or causing!) which problems at which company sizes.

QUICK VIEW

01

Nagios is still the most common tool in medium and large online service companies while New Relic and Icinga are more popular in small online services.

02

Many of the most popular monitoring tools were designed before DevOps organizational trends, cloud architectures, and microservices.

03

Most of the complaints about monitoring from 4 years ago persist because of siloed tools and metrics, alert desensitization, and insufficient technology for monitoring ephemeral instances in cloud architectures.

On a higher level, we learned two things. First, in some ways, monitoring still sucks, in ways we'll explain in more detail below. Second, the pain is actually going to get worse as more companies move to microservices.



WHAT MONITORING TOOLS COMPANIES ARE USING

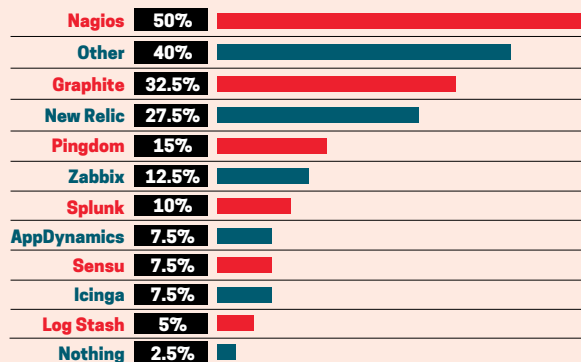
Although a lot of new tools have arrived since 2011, it's clear that older open source tools like Nagios, Zabbix, and Icinga still dominate the market; **70%** of the companies we spoke to are still using these older tools for their core monitoring and alerting (see Chart 01). Among DZone's enterprise developer audience, 43% used Nagios, Zabbix, or Icinga. Nagios was also the most popular monitoring tool at 29% marketshare [1].

Around 70% of the companies used more than one monitoring tool, with most using an average of two. Nagios and Graphite configurations were the most common, with many also using New Relic. However, only two of the companies we spoke to actually paid for New Relic. Most users did not upgrade from the free version because they thought the paid version was too expensive.

There were a lot of different tools in the “Other” category, but no particular tool stood out. The tools in the “Other” category were mainly used by startups

and included some of the newer SaaS monitoring tools such as Librato and Datadog. Older open source tools like Cacti and Munin were also well-represented in this group, along with AWS CloudWatch. However, a significant number of tools in this category were home-grown solutions. Some of the larger services had invested significant resources into building custom monitoring solutions from the ground up.

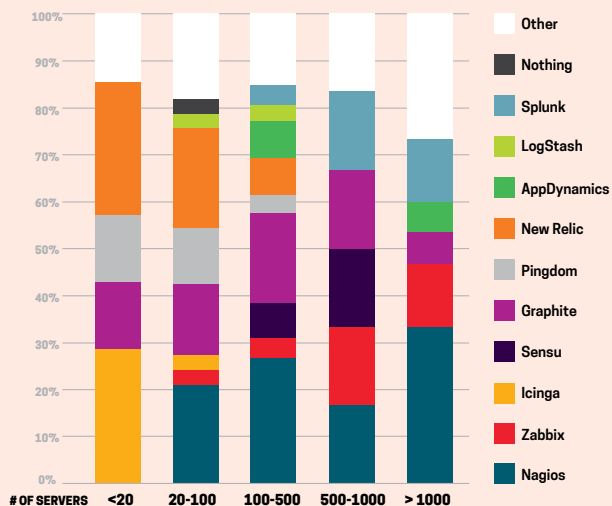
01. THE MONITORING TOOLS THAT COMPANIES DEPLOYED



HOW MANY SERVERS COMPANIES ARE MONITORING

If we look at tool usage versus the number of servers the companies manage (ranged from < 20 servers being a startup service to >1000 servers for the large online services), the proportion of older open source tools like Nagios and paid on-premise tools goes up as the service gets larger, whereas the smaller services are more likely to use developer-focused tools like Graphite and New Relic.

02. THE MONITORING TOOLS THAT COMPANIES DEPLOYED



This makes sense insofar as many of the larger services are older (> 5 years old) and have legacy monitoring infrastructure. They also have the resources to hire a dedicated operations team that tends to bring in the tools they're most familiar with, namely Nagios and its alternatives. They also have the budget to pay for the enterprise versions of monitoring tools like Splunk and AppDynamics.

In one company's case, they were receiving around 5000 alert emails a day.

On the other hand, the smaller services often don't have any DevOps or dedicated operations people in their company, so their developers tend to use simple-to-install SaaS monitoring tools or popular tools in the developer community such as Graphite and LogStash. There seems to be a tipping point between 50-100 servers where the company has the resources to bring in a DevOps/operations person or team, and they in turn tend to bring in time-tested infrastructure monitoring tools that they trust, like Nagios.

KEY TRENDS

Here are four of the major trends we uncovered from interviewing more than 60 online service companies about their monitoring strategies.

1. BUILDING & SCALING THE "KIT CAR"

With 78% of online services running their own open-source monitoring solution, many spent between 4-6 months building their monitoring solution with open-source components and then tuning it to work well with their environment. The key issue is that many of the tools were originally designed 10-15 years ago, before cloud architecture, DevOps, and microservices. A significant amount of time is spent adapting these older tools to work in today's dynamic environments.

Once they had built and tuned their "kit car" monitoring stack, their services started growing, and they needed to spend more time modifying their monitoring system so that it could handle the increasing amount of data. For example, a large online service with over 1000 instances on AWS had a Zabbix server fall over after the MySQL database behind it filled up to 2TB of data. In the end, they just dropped the database and restarted again, rather than try to make Zabbix scale.



2. SPAMMY ALERTS

If there was one consistent complaint from all the companies we spoke to, it was about overenthusiastic alerting. It's clear that none of the tools, even the monitoring tools that claim to have advanced machine learning algorithms, have solved the problem of alert fatigue. The problem is only getting worse as companies scale out on more servers or run microservices on continuously changing cloud environments. It was also clear that, despite the marketing claims of many companies in the space, none of the machine learning algorithms for anomaly detection or predictive alerting really worked well in practice. This indicated to us that there is still a long way to go before these tools help automatically filter out the noise of alerts in monitoring.

In one company's case, they were receiving around **5000 alert emails a day**. With that volume, alerting had just become noise, and most of the team simply filtered the alerts into a folder or automatically deleted them altogether.



3. DATA SILOS

Many companies we talked to were collecting real-time data. These data sources even included business metrics such as number of signups, checkouts, or revenue figures, which teams used to keep an eye on the service. However, most of the monitoring tools they used suffered from poor usability and dated UIs, so the collected data was siloed away for the eyes of the operations team alone. This means that real-time data about the service was less accessible to other stakeholders who would get value out of seeing this data too.

Many services solved the data silo problem by building custom dashboards that could be displayed on TVs around the office or shared via URL. But these dashboards were usually very static and required a developer to make changes when needed.

On the other hand, for companies where monitoring data was easily shared, the monitoring tool became a much more valuable tool for different teams to collaborate around, both to identify areas for improvement and to gain visibility into real-time performance across the business.



4. MICROSERVICES

A key trend in online services is the microservices deployment model, which involves separate cross-functional development teams deploying and supporting their own services in production. This strategy enables a large, complex application to become highly scalable as it grows. However, it dramatically increases the number of servers and services the DevOps/operations team needs

to support, so it only works if the development teams become the first line of support when things go wrong.

In this model, operations staff become a “platforms” team, providing common tools and processes for development teams. This ops-provided platform includes self-service monitoring, whereby developers have the ability to add their own checks and create their own dashboards and alerts.

For companies where monitoring data was easily shared, the monitoring tool became a much more valuable tool.

Keeping up with high-speed deployments and ephemeral instances in a microservices model is a huge challenge for today's monitoring tools. It's also difficult to visualize the complex flow of tasks through various services and to deal with the highly dynamic scale [2]. Unfortunately, it's clear that current monitoring tools have not been designed around this microservice-centric model, and most suffer from poor usability and adoption in teams outside operations. New tools designed specifically to handle microservices are required so that both operations and development can collaborate easily around a single source of performance information, instead of having developers use their own tool (typically New Relic) and operations use theirs (typically Nagios).

CONCLUSION

There are many more monitoring tools available in the four years after “#monitoringsucks” became a DevOps meme, but our research shows that many organizations are still struggling with monitoring. We believe this is mainly because new tools focus on the technical aspects of monitoring, but do not adequately drive adoption in teams outside operations. This is the problem we're focusing on at Dataloop.IO, because we believe that the glue between Dev and Ops is reliable monitoring that everyone in the organization uses to make data-driven decisions that improve their software.

[1] 2015 DZone Performance & Monitoring Survey

[2] <http://www.slideshare.net/adriancockcroft/software-architecture-monitoring-microservices-a-challenge>



DAVID GILDEH is the Founder and CEO of Dataloop.IO, a monitoring service for online services. Designed for Cloud, DevOps & Microservices, Dataloop.IO is the first monitoring solution that allows Operations to provide monitoring as a self-service solution to other teams while still having overall visibility and control.

MOST VALUABLE BLOGGER ARTICLE

Try Optimizing Memory Consumption First

BY PETER LAWREY

When you want your application to run faster you might start off by doing some CPU profiling. However, when I'm looking for quick wins in optimization, it's the memory profiler I target first.

MY TOOLS

When profiling I use [YourKit](#) for ease of use and I confirm these results with [Java Mission Control](#) for great accuracy. I use [IntelliJ IDEA](#) as my IDE.

ALLOCATING MEMORY IS CHEAP

Allocating memory has never been cheaper. You can buy 16 GB for less than \$200. There are affordable machines with hundreds of GBs of memory. The memory allocation operation is also cheaper than it has been in the past, and it's multi-threaded, so it scales reasonably well. However, memory allocation is not free.

Your CPU cache is a precious resource, especially if you are trying to use multiple threads. While you can buy 16 GB of main memory easily, you might only have 2 MB of cache per logical CPU. If you want these CPUs to run independently, you want to spend as much time as possible within the 256 KB L2 cache (see table, top right).

ALLOCATING MEMORY IS NOT LINEAR

Allocating memory on the heap is not linear. The CPU is very good at doing things in parallel. This means that if memory bandwidth is not your main bottleneck, the rate you produce garbage has less impact than whatever is your bottleneck. However, if the allocation rate is

CACHE LEVEL	SIZE	ACCESS TIME (IN CLOCK CYCLES)	CONCURRENCY
L1	32 KB data 32 KB instruction	1	Cores independent
L2	256 KB	3	Cores independent
L3	3 MB - 45 MB	10-20	Sockets independent
Main Memory	4 MB - 4 TB	200+	Each memory region separate

high enough (and in most Java systems it is high), it will be a serious bottleneck. You can tell that the allocation rate is a bottleneck if:

- You are close to the maximum allocation rate of the machine. Write a small test that creates a lot of garbage and measure the allocation rate. If you are close to the max allocation rate, you have a problem.
- When you reduce the garbage produced by say 10%, the 99% latency of your application becomes 10% faster, and yet the allocation rate hardly drops. This means your application sped up so that it reached your bottleneck again.
- You have very long GC pause times (e.g. into the seconds). At this point, your memory consumption is having a very high impact on your performance, so reducing the memory consumption and allocation rate can improve scalability (how many requests you can process concurrently) and reduce the amount of time during which the application freezes.

COMBINING THE CPU AND MEMORY VIEWS

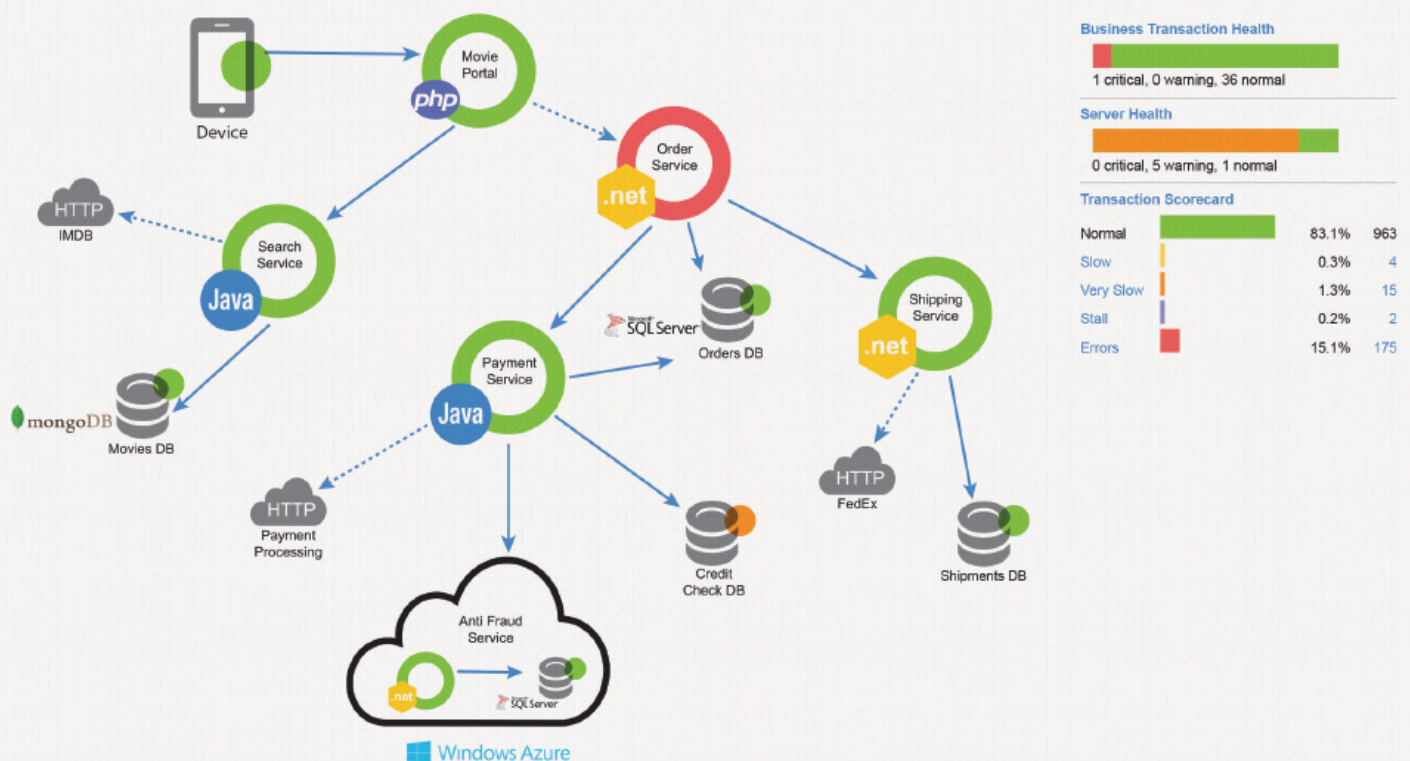
After reducing the memory allocation rate, I look at the CPU consumption with memory tracing turned on. This gives more weight to the memory allocations and will provide an alternative to just looking at the CPU alone. When this CPU and memory view shows you that the application is spending most of its time doing essential work, and there are no more easy performance gains to be made, I then look at CPU profiling alone. Using these techniques as a starting point, my aim typically is to reduce the 99th percentile latency (the worst 1%) by a factor of 10. This approach can also increase the throughput of each thread and allow you to run more threads concurrently in an efficient manner.



PETER LAWREY is the architect of Chronicle Software. He has the most answers on StackOverflow for Java and JVM, and is the founder of the Performance Java User's Group. His blog "Vanilla Java" has over 3 million views.

If your business runs on apps, Application Intelligence is for you.

- **See** real-time application performance, user experiences, and infrastructure capacity
- **Act** fast. Isolate, resolve and automate the resolution of performance bottlenecks — in production
- **Know** the business impact with real-time application analytics



See how your apps are performing.
Start a FREE trial at appdynamics.com

Top 5

Java Performance Metrics to Capture in Enterprise Applications

This article reviews the top five performance metrics to assess the health of your enterprise Java application. They are:

- Business Transactions
- External Dependencies
- Caching Strategy
- Garbage Collection
- Application Topology

1. Business Transactions

If you were to measure only a single aspect of your application, I would encourage you to measure the behavior of your business transactions. While container metrics can provide a wealth of information and can help you determine when to auto-scale your environment, your business transactions determine the performance of your application. Instead of asking for the thread pool usage in your application server you should be asking whether or not your users are able to complete their business transactions and if those business transactions are behaving normally.

A business transaction entry-point can be defined by interactions like a web request, a web service call, or a message on a message queue. Alternatively, you may choose to define multiple entry-points for the same web request based on a URL parameter or for a service call based on the contents of its body. Once a business transaction is identified, then its performance is measured across your entire application ecosystem.

Visit AppDynamics to [read the rest of this article](#).



WRITTEN BY STEVEN HAINES

TECHNICAL ARCHITECT,
PIKSEL

AppDynamics

APPDYNAMICS

AppDynamics is the application intelligence company that helps today's software-defined businesses proactively monitor, manage, and optimize the most complex software environments —from desktop to mobile — all in real time, and all in production. With cloud, on-premises, and hybrid deployment flexibility, AppDynamics works and partners with many of the world's most innovative companies.

CATEGORY

APM and Infrastructure Monitoring

FEATURES

- End-to-end transaction tracing
- Code level visibility
- Dynamic baselining and alerting
- Troubleshooting and control
- Scalability
- Data retention

CUSTOMERS

- Cisco
- NASDAQ
- Expedia
- eHarmony
- Fox News
- Hallmark
- Overstock.com
- Kraft

LANGUAGE SUPPORT

- Java
- .NET
- PHP
- Node.js
- Python
- C++
- Mobile (iOS and Android)
- Database
- Infrastructure
- Cloud
- Extensions

ALERTS

- Threshold alerts
- Limit alerts
- Status alerts
- Conditional alerts
- Alert scheduling

CASE STUDY

NASDAQ software technology powers over 70 marketplaces, regulators, central securities depositories, and clearing houses in over 50 countries. Traditionally, NASDAQ has monitored and managed its applications using a variety of somewhat disparate monitoring, alerting, and log aggregation tools. NASDAQ scrutinized how the solution was architected to gather data, and what kind of usability and traceability it offered right out of the box. After looking at several of the leading APM solutions, AppDynamics quickly rose to the top. The AppDynamics Application Intelligence Platform immediately demonstrated its ability to deliver value right out of the box.

BLOG blog.appdynamics.com

TWITTER [@AppDynamics](https://twitter.com/AppDynamics)

WEBSITE appdynamics.com

Mechanical Sympathy:

UNDERSTANDING THE HARDWARE MAKES YOU A BETTER DEVELOPER

BY DAVE FARLEY

I have spent the last few years of my career working in the field of high-performance, low-latency systems. Two observations struck me when I returned to working on the metal:

- *Many of the assumptions that programmers make outside this esoteric domain are simply wrong.*
- *Lessons learned from high-performance, low-latency systems are applicable to many other problem domains.*

Some of these assumptions are pretty basic. For example, which is faster: storing to disk or writing to a cluster pair on your network? If, like me, you survived programming in the late 20th century, you know that the network is slower than disk, but that's actually wrong! It turns out that it's much more efficient to use clustering as a backup mechanism than to save everything to disk.

Other assumptions are more general and more damaging. A common meme in our industry is “avoid pre-optimization.” I used to preach this myself, and now I'm very sorry that I did. These are just a few supposedly obvious principles that truly high-performance systems call into question. For many developers, these rules of thumb appear reasonably inviolable in everyday practice. But as performance demands grow increasingly strict, it becomes proportionally

important for developers to understand exactly how systems work—at both the abstract, procedural level, and the level of the metal itself.

MOTIVATION: THE REPERCUSSIONS OF INEFFICIENT SOFTWARE

First, let's think about why it's important to transcend crude oversimplifications like “disk I/O is faster than network I/O.”

One motivation is a bit negative. I can think of no other field of human endeavor that tolerates the levels of inefficiency that are normal in our industry. My experience has been that for most systems, any performance specialist can improve its performance ten-fold quite easily. This is because most applications are hundreds, thousands, even tens of thousands of times less efficient than they could be. Modern hardware is phenomenally capable, but we software folks regularly underestimate the strides hardware manufacturers have made. As a result, we often fail to take advantage of new hardware capabilities and miss significant shifts in the locations of common performance bottlenecks.

You may say this doesn't matter—after all, what is all that hardware performance for, if not to make it easier to write software? And then I'll reply: it's for lots of things. Here's one very concrete reason. The energy consumption from

QUICK VIEW

01

The key to high performance: minimize instructions, minimize data. To do this, model the domain and simplify the code.

02

Hardware probably isn't the bottleneck. Check for the physical theoretical max (read/write, process, send/receive) and compare with application performance before blaming the metal.

03

If all software companies challenged themselves to improve their code's efficiency by several factors, it would benefit the whole world by reducing our carbon footprint.

data centers constitutes a significant fraction of the CO₂ being pumped into our atmosphere [1]. If most software is more than 10x less efficient than it could be, then that could mean 10x more CO₂. It also means 10x the capital cost in hardware required to run all that inefficient software.

“You don’t have to be an engineer to be a racing driver, but you do have to have Mechanical Sympathy.”

– Jackie Stewart, racing driver

Perhaps more important is that performance is about more than just efficiency for its own sake. Performance is also an important enabler of innovation. The ability to pack the data representing 1000 songs onto the tiny hard disk in the first generation iPod made it possible for Apple to revolutionize the music industry. For an even more basic example, graphical user interfaces only became feasible when the hardware was fast enough to “waste” all that time drawing pretty pictures. High-performance hardware enabled these innovations; the software simply had to follow.

Then there’s the opportunity cost. What could we be doing with all the spare capacity of our astonishingly fast hardware and enormously vast storage if we weren’t wasting it on inefficient software?

In the problem domains where I’ve worked, the idea of mechanical sympathy has helped enormously. Let’s examine this idea a little closer.

KEY CONCEPT: MECHANICAL SYMPATHY

The term Mechanical Sympathy was coined by racing driver Jackie Stewart and applied to software by Martin Thompson. Jackie Stewart said, “You don’t have to be an engineer to be a racing driver, but you do have to have Mechanical Sympathy.” He meant that understanding how a car works makes you a better driver. This is just as true for writing code. You don’t need to be a hardware engineer, but you do need to understand how the hardware works and take that into consideration when you design software.

Let’s take something simple like writing a file to disk. Disks are random access, right? Well, not really. Disks work by

encoding data in sectors and addressing them in segments of the disk. When you read or write data to disk, you need to wait for the heads to move to the correct physical location. If you do this randomly, then you will incur performance penalties as the heads are physically moved across the surface of the disk and you wait for the spin of the disk to place the sector you want beneath the read/write heads. This averages out to about 3ms per seek for the heads, and about 3ms rotational latency. That makes for an average total of about 6ms per seek!

This seek time is dramatically slow when compared to the electronic. Electrons beat spinning rust every time! And there are even more specific reasons why “random access” is a poor rule of thumb. In fact, modern disks are optimized to stream data so that they can play movies and audio. If you understand that, and treat your file storage as a serial, block device rather than random access, you will get dramatically higher performance. In this example, the difference can be two orders of magnitude (200MB/s is a good working figure for the upper bound).

REACHING HIGH PERFORMANCE: DO EXPERIMENTS, DO ARITHMETIC

Abstraction is important. It is essential that we are, to some degree, insulated from the complexity of the devices and systems that form the platform upon which our software executes. However, many of the abstractions that we take for granted are very poor and leaky. The overall system complexity gets amplified when we build abstraction on top of abstraction to hide the problems caused by the leakiness, resulting in the dramatic performance differences we see between high-performance and “normal” systems.

You don’t need to model or measure your entire system to tackle its complexity. The starting point is to do some experimenting to understand your theoretical maximums. You should have a rough model of the following:

- How much data you can write to disk in one second
- How many messages your code can process in one second
- How much data you can send or receive across your network in one second

A common meme in our industry is, “avoid pre-optimization.” I used to preach this myself. I am now very sorry that I did.

The idea here is to measure your actual performance against the theoretically possible performance of your system within the limits imposed by the underlying hardware.

Let's consider a specific example. I recently talked with a developer who believed he was working on a high-performance system. He told me that his system was working at 10 transactions per second. As of mid-2015, modern processors on commodity hardware can easily perform 2-3 billion instructions per second. So at 2-3 billion instructions per second, the developer was limited to 200-300 million instructions per transaction ($[10 \text{ transactions/sec}] / [2\text{-}3\text{G instructions/sec}] = 200\text{-}300\text{m instructions/transaction}$) to achieve his goals. Of course, his application isn't responsible for all of these—the operating system is chewing some cycles too—but 200 million instructions is an awful lot of work. You can write an effective chess player in 672 bytes, [as David Horne has shown](#).

Okay, so maybe the bottleneck was I/O. Perhaps this developer's application was disk bound. That's unlikely—modern hard disks found in commodity hardware can transfer data at phenomenal rates. A moderate transfer rate from disk to a buffer in memory is about 10MB/s. If this is the limit, he must be pushing more than 10MB per transaction between disk and memory. You can represent a lot of information in 10 million bytes!

Well, perhaps the problem was the network. Probably not—10Gbit/s networks are now on the low end. A 10Gbit/s network can transmit roughly 1GB of data per second. This means that each of our misguided developer's 10 transactions per second is occupying 1/10th of a GB, or approximately 100MB—more than 10 times the throughput of our disks!

The hardware wasn't the problem.

HIGH PERFORMANCE SIMPLICITY I: MODEL THE PROBLEM DOMAIN

One of the great myths about high-performance programming is that high-performance solutions are more complex than “normal” solutions. This is just not true. By definition, a high-performance solution must do the most amount of work in the fewest instructions. Complex solutions precisely don't last long in high-performance systems because complexity is where performance bottlenecks hide.

The best programmers I know achieve the simplicity demanded by high-performance systems by modeling the problem domain. A software simulation of the problem domain is the best way to come up with an elegant solution.

HIGH PERFORMANCE SIMPLICITY II: SIMPLIFY THE CODE

If you follow the lead of the problem domain, you tend to end up with smaller, simpler classes with clearer

relationships. If you follow strong separation of concerns as a guiding principle, then your design will push you in the direction of cleaner, simpler code. The cardinal rule of object-oriented simplicity is “one class, one thing; one method, one thing.”

Modern compilers are extremely effective at optimizing code, but they are best at optimizing simple code. If you write 300-line methods with multiple for-loops, each containing several nested if-conditions throwing exceptions all over the place and returning from multiple points, then the optimizer will simply give up. If you write small, simple, easy to read methods, they are easier to test and easier for the optimizer to understand, which results in significant improvements to performance.

One of the great myths about high-performance programming is that high-performance solutions are more complex than “normal” solutions. This is just not true.

THE KEY TO HIGH PERFORMANCE: MINIMIZE INSTRUCTIONS, MINIMIZE DATA

Fundamentally, developing high-performance systems is simple: minimize the number of instructions being processed and the amount of data being shunted around.

You achieve this by modeling the problem domain and eliminating nonessential complexity. For software developers, “Mechanically Sympathizing” with modern high-performance hardware will help you understand where you're doing things wrong. It can also point you in the direction of better measurement and profiling, which will help you understand why your code is not performing to its theoretical maximum. So why not try to save your company some money, work in a simplified codebase, and maybe help reduce the carbon footprint of our data centers—all at the same time?

[1] <https://energy.stanford.edu/news/data-centers-can-slash-co2-emissions-88-or-more>



DAVE FARLEY is an independent consultant and the co-author of [Continuous Delivery](#), a seminal work on software development and delivery. Dave is one of the authors of [The Reactive Manifesto](#). Dave is an independent consultant and the founder and director of Continuous Delivery Ltd. He has significant experience working on high-performance systems from his time at LMAX Ltd.

diving deeper

INTO PERFORMANCE & MONITORING

top 10 #performance twitter feeds

TO FOLLOW RIGHT AWAY



@SOUDERS



@BRENDANGREGG



@ALLSPAWE



@PERFNIGHTMARES



@MDAOUDI



@LUSIS



@TAMEVERTS



@BBREWER



@BBINTO



@CHRISLOVE

performance zones

LEARN MORE & ENGAGE YOUR PEERS IN OUR PERFORMANCE-RELATED TOPIC PORTALS

Performance Zone

dzone.com/mz/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/mz/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.

Javalobby

java.dzone.com

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 & monitoring refcardz

Java Performance Optimization

bit.ly/DZ-JavaPerformance

Java Profiling with VisualVM

bit.ly/DZ-JavaProfiling

Refactoring Patterns

bit.ly/DZ-Refactoring

Scalability & High Availability

bit.ly/DZ-Scalability

top performance & monitoring websites

High Performance Websites

stevesouders.com

Brendan Gregg's Blog

brendangregg.com/blog

Planet Performance

perfplanet.com

top speed test tools

WebPage Test

webpagetest.org

Pingdom Website Speed Test

tools.pingdom.com/fpt

Google PageSpeed Insights

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



The real disruptors? Your customers.

Welcome to the application economy. Where customers don't walk into a store. They pull out their mobile phones. Where users don't just expect a flawless experience, but one tailored to their tastes. Where the difference between success and failure come down to quality of your applications—and billions in potential revenue hangs in the balance. It's an enormous opportunity for those nimble enough to seize it. And CA Technologies helps you do just that. From planning to development to management to security, only CA offers an end-to-end portfolio of software solutions to give your business a competitive advantage in the application economy.

See how leading businesses are driving double-digit growth.

Read CA's new Application Economy Market Study at [**rewrite.ca.com/appecon**](http://rewrite.ca.com/appecon)



Business, rewritten by software™

Want to win in the App Economy?

Begin at the end user.

In a time when businesses are literally being rewritten by software, applications have now become the face of your business. And in the age of rapid adoption and rapid rejection, you have mere seconds to impress your users. This is the reality of the App Economy. Despite the enormous complexity of today's application delivery chain, your end users expect a flawless app experience, regardless of how, when, or where they access your app. This means app issues are not IT issues; they are customer satisfaction and retention issues.

What may seem like a simple app to an end user may consist of a complicated set of underlying services,

APIs, and middleware, both cloud and on-premises, spanning mobile to mainframe. *The truth is, your end user just wants an app that performs. They don't care how complicated it is under the cover.* To provide a winning app experience for your end users, a modern approach to Application Performance Management must be Easy, Proactive, Intelligent, and Collaborative.

Easy: Making APM easy to adopt, fast time-to-value, simple-to-use, and simple to manage and configure

Proactive: Giving you automatic visibility into each native mobile or web-based transaction to enable your organization to stay ahead of emerging performance issues

Intelligent: Collecting and delivering not just more information, but actionable information

Collaborative: Improving communication between Dev and Ops specialists to resolve problems faster



WRITTEN BY DAVID HARDMAN

DIRECTOR, PRODUCT MARKETING
CA TECHNOLOGIES

CA Application Performance Management by CA Technologies



CA APM provides broad, modern support, and is able to follow transactions from mobile to mainframe, providing transaction-level visibility.

CATEGORY

APM

FEATURES

- Synthetic monitoring
- Real-user monitoring
- Browser testing

CUSTOMERS

- Rackspace
- BCBS Tennessee
- Dansk
- Tesco
- Lexmark
- Telefonica
- HCL Technologies
- Innovapost

LANGUAGE SUPPORT

- C#
- Java
- Python
- PHP
- JavaScript
- SQL

ALERTS

- Threshold alerts
- Limit alerts
- Status alerts
- Conditional alerts
- Alert scheduling

CASE STUDY

Innovapost is the IS and IT shared services provider for the Canada Post Group of Companies. With up to CAN\$14 million worth of transactions being processed by Canada Post's e-shipping application every day, availability is a major concern for Innovapost. It uses CA Application Performance Management (APM) to monitor its e-shipping application round-the-clock and alert Innovapost if performance thresholds are breached. It also provides crucial data for capacity management. Innovapost is able to maximize the availability of the e-shipping tool, which helps to maintain its revenues and reputation. It is also able to provide a more cost-effective and scalable service.

BLOG communities.ca.com/community/ca-apm/blog

TWITTER @CAinc

WEBSITE ca.com/apm

The Web Performance APIs Reference

BY BARBARA BERMES

The W3C [Web Performance Working Group](#) and browser vendors have acknowledged that performance is an important piece of web development, proposing new standards and implementations manifested in [performance APIs](#). The purpose of a (web) API is to provide better access to users' browser and device. Performance APIs can ease the process of accurately measuring, controlling, and enhancing the users' performance.

Each of the following performance APIs are in different stages of the W3C's [specification maturity](#) process. You can see each spec's stage next to their title. Visit [this article](#) for a concise graphic of all the performance APIs' maturity levels.

NAVIGATION TIMING - W3C RECOMMENDATION

The Navigation Timing API helps measure real user data (RUM) such as bandwidth, latency, or the overall page load time for the main page. Here is how developers check the page's performance via JavaScript through the PerformanceTiming interface:

```
var page = performance.timing,
    plt = page.loadEventStart - page.navigationStart,

console.log(plt); // Page load time (PTL) output for
specific browser/user in ms;
```

Navigation timing covers the metrics of the entire page. To gather metrics about individual resources, you'll use the Resource Timing API explained further down the list.

HIGH RESOLUTION TIMING - W3C RECOMMENDATION

The High Resolution Timing API supports floating point timestamps and provides measurements to a microsecond

level of detail so that it is not subject to system clock skew or adjustments.

```
var perf = performance.now();
// console output 439985.4570000316
```

PAGE VISIBILITY - W3C RECOMMENDATION

The `visibilitychange` event created by this spec is fired on document whenever the page gains or loses focus. This event is very helpful in programmatically determining the current visibility state of the page. For example, the API can be applied if your user has several browser tabs open and you don't want specific content to execute (e.g playing a video, or rotating images in a carousel).

```
document.addEventListener('visibilitychange',
function(event) {
  if (document.hidden) {
    // Page currently hidden.
  } else {
    // Page currently visible.
  }
});
```

RESOURCE TIMING - CANDIDATE RECOMMENDATION

The Resource Timing API lets you dig deeper into understanding the behavior of *each individual resource* in a page. As an example, let's pick the DZone logo, and retrieve the total loading time in ms:

```
var img = window.performance.getEntriesByName("http://
cdn.dzone.com/static/images/TOP_NAV_LOGO.png")[0];
var total = parseInt(img.responseEnd - img.startTime);
console.log(total); // output e.g. 281 (in ms)
```

QUICK VIEW

01

There are web standard APIs you can use right now that measure RUM data, give more accurate timestamps, save browser resources for non-viewed pages, and provide smoother animations.

02

Several APIs are coming soon that will provide a unified view of performance metrics and allow developers to track battery levels and measure individual resources, blocks, and functions.

03

APIs further in the future will provide frame performance tracking, navigation error logging, and optimizations based on predicted browsing.

Beyond the main page's performance (via the Navigation Timing API), you can also track real user experiences on a more granular basis (i.e. resource basis).

PERFORMANCE TIMELINE - CANDIDATE RECOMMENDATION

The Performance Timeline specification defines a unifying interface to retrieve the performance data collected via Navigation Timing, Resource Timing, and User Timing.

```
// gets all entries in an array
var perf = performance.getEntries();
for (var i = 0; i < perf.length; i++) {
  console.log("Asset Type: " + perf[i].name + "
Duration: " + perf[i].duration + "\n"); }
```

BATTERY STATUS - CANDIDATE RECOMMENDATION

This API provides access to the battery status of your user's battery-driven device. The specific metrics you can access are charging, chargingTime, dischargingTime and level. Events can also be fired based on these statuses:

```
var battery = navigator.battery || navigator.
webkitBattery || navigator.mozBattery || navigator.
msBattery;
if (battery) {
  console.log("Battery charging? " + battery.charging ?
"Yes" : "No");
  console.log("Battery level: " + battery.level * 100 +
"%");
  console.log("Battery charging time: " + battery.
chargingTime + " seconds");
  console.log("Battery discharging time: " + battery.
dischargingTime + " seconds");
};
```

USER TIMING - CANDIDATE RECOMMENDATION

With the User Timing API, you can set markers to measure specific blocks or functions of your application. The calculated elapsed time can be an indicator of good or bad performance.

```
performance.mark("start");
loadSomething();
performance.mark("end");
performance.measure("measureIt", "start", "end");
var markers = performance.getEntriesByType("mark");
var measurements = performance.
getEntriesByName("measureIt");
console.log("Markers: ", markers);
console.log("Measurements: ", measurements);
function loadSomething() {
  // some crazy cool stuff here :)
  console.log(1+1);
}
```

BEACON - WORKING DRAFT

With the beacon API, you can send analytics or diagnostic code from the user agent to the server. By sending this asynchronously, you won't block the rendering of the page.

```
navigator.sendBeacon("http://mywebserver.com", 'any
information you want to send');
```

ANIMATION TIMING - EDITOR'S DRAFT

Instead of using setTimeout or setInterval to create animations, this spec will introduce the requestAnimationFrame. This method grants the browser control over how many frames it can render; aiming to match the screen's refresh rate (usually 60fps) will result in a jank-free experience. It can also throttle animations if the page loses visibility (e.g., the user switches tabs), dramatically decreasing power consumption and CPU usage.

Check out Microsoft's [demo page](#) comparing setTimeout with requestAnimationFrame.

RESOURCE HINTS - EDITOR'S DRAFT

Predictive browsing is a great way to serve your users with exactly what they want to see or retrieve next. "Pre-browsing" refers to an attempt to predict the users' activity with your page—i.e. is there anything we can load prior to the user requesting it? This spec will allow developers to give the User Agent hints on the download priority of a resource.

The following [pre-browsing attributes](#) are meant to help you with pre-loading assets on your page.

```
<link rel="dns-prefetch" href="//host_to_resolve.com">
<link rel="subresource" href="/javascript/mydata.
json">
<link rel="prefetch" href="/images/labrador.jpg">
<link rel="prerender" href="//example.org/page_2.
html">
```

FRAME TIMING (NOT SUPPORTED YET) - EDITOR'S DRAFT

This specification defines an interface to help web developers measure the frame performance of their applications by giving them access to measurements like frames per second and time to frame. – [W3C](#)

NAVIGATION ERROR LOGGING (NOT SUPPORTED YET) - EDITOR'S DRAFT

This specification defines an interface to store and retrieve error data related to the previous navigations of a document. – [W3C](#)

BROWSER SUPPORT

For a chart of the browser support for all of these standards, visit this [blog post](#) and scroll down to *Browser support overview*. Your best method for keeping up with upcoming standards is to subscribe to the Web Performance Working Group [mailing list](#) for any performance-related updates.



BARBARA BERNES oversees the development of [OANDA's](#) next-generation APIs. She acts as the internal and external evangelist for the API product suite as well as OANDA's developer community. She is the author of [Lean Websites](#).


The Scale of Computing Latencies

To computers, we humans work on a completely different time scale, practically geologic time. Which is completely mind-bending. The faster computers get, the bigger this time disparity grows.

- *Jeff Atwood*


For all scale analogies, consider 1 CPU cycle = 1 second
(In reality, 1 CPU cycle = 0.3 nanoseconds)

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
(THE WORLD RECORD IS ABOUT 29 SECONDS)

MUTEX LOCK/UNLOCK
= 17 NS, WHICH = 56 SEC,
OR IS EQUAL TO




Washing
your dishes

MAIN MEMORY ACCESS
= 100 NS, WHICH = 6 MIN,
OR IS EQUAL TO




LISTENING TO QUEEN'S
"Bohemian
Rhapsody"

COMPRESS 1KB WITH ZIPPY
= 2μS, WHICH = 2 HOURS,
OR IS EQUAL TO




Watching
a movie

READ 1M BYTES
SEQUENTIALLY FROM MEMORY
= 9μS, WHICH = 9 HOURS,
OR IS EQUAL TO




COMPLETING A STANDARD
US workday

SSD RANDOM READ
= 16 μS, WHICH = 14 HOURS,
OR IS EQUAL TO




TAKING A FLIGHT FROM
New York
to Beijing

SOLID-STATE DISK I/O
= 50-150 μS, WHICH = 2-6 DAYS,
OR IS EQUAL TO



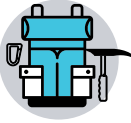
WAITING FOR A STANDARD GROUND-SHIPED
US domestic
package

READ 1M BYTES
SEQUENTIALLY FROM SSD
= 200 μS, WHICH = 8 DAYS,
OR IS EQUAL TO




IF THERE WERE 8 DAYS IN A WEEK,
IT WOULD NOT BE ENOUGH TIME FOR
The Beatles
TO SHOW THEY CARE

ROUND TRIP IN THE
SAME DATACENTER
= 500 μS, WHICH = 19 DAYS,
OR IS EQUAL TO




Free climbing
EL CAPITAN'S DAWN WALL
IN YOSEMITE NATIONAL PARK

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




PLANTING + HARVESTING A
zucchini

DISK SEEK
= 4MS, WHICH = 5 MONTHS,
OR IS EQUAL TO




TRAINING FOR YOUR
first marathon
IF YOU'VE NEVER DONE ONE + YOU'RE
AT AN AVERAGE FITNESS LEVEL

ROTATIONAL DISK I/O
= 1-10MS, WHICH = 1-12 MONTHS,
OR IS EQUAL TO




WAITING UNTIL THE NEXT SEASON OF
Game of Thrones

INTERNET: SF TO NYC
= 71MS, WHICH = 7 YEARS,
OR IS EQUAL TO




ATTENDING + GRADUATING
Hogwarts
IF YOU'RE A WITCH OR WIZARD

OS VIRTUALIZATION REBOOT
= 4 S, WHICH = 423 YEARS,
OR IS EQUAL TO




423 YEARS AGO,
Shakespeare
WROTE RICHARD III

SCSI COMMAND TIME-OUT
= 30 S, WHICH = 3,000 YEARS,
OR IS EQUAL TO




3,000 YEARS AGO, PEOPLE STARTED
wearing pants

HARDWARE VIRTUALIZATION
REBOOT
= 40 S, WHICH = 4,000 YEARS,
OR IS EQUAL TO



4,000 YEARS AGO, THE PHAROHS STILL
ruled Egypt

PHYSICAL SYSTEM REBOOT
= 5 MINUTES, WHICH = 32,000 YEARS,
OR IS EQUAL TO



32,000 YEARS AGO, THE AREA THAT IS THE
Sahara desert was
well-watered

ORIGINAL CONCEPT, PETER NORVIG: NORVIG.COM/21-DAYS.HTML#ANSWERS
COLIN SCOTT: EECS.BERKELEY.EDU/~RCS/RESEARCH/INTERACTIVE_LATENCY.HTML

SOURCES

SYSTEMS PERFORMANCE: ENTERPRISE + CLOUD BY BRENDAN GREGG: AMAZON.COM/DP/0133390098/
AT&T US NETWORK LATENCIES: IPNETWORK.BGTMO.IP.ATT.NET/PWS/NETWORK_DELAY.HTML

See your code like you've never seen it before.



Build, deploy, and manage great web software with New Relic APM™.

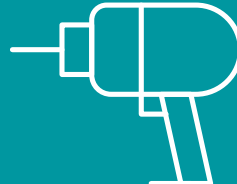
New Relic constantly monitors your app performance so you don't have to.



Monitor any app, in any language, on any hosting environment



See every aspect of your app's environment in real time



Drill down to the exact line of code and SQL statements



Identify bottlenecks, streamline issues, and reduce resolution times

3 APM Tips Every Enterprise Developer Should Know

Today's modern architectures are constantly evolving. Distributed applications are being rapidly built and shipped with the ability to run anywhere. But for developers, the end goal shouldn't just be about shipping software faster; it should be about delivering high-quality software faster. Here are three recommended ways to get there.

1. Set up business-critical transactions. In web applications, some transactions are more important to your business than others. You may want to closely monitor these transactions to make sure that you're not causing frustration for your users—and negatively impacting the business. We recommend that you review end user and app response times, call counts, and error rates, and that you set alert notifications for when these key transactions are performing poorly.

2. Track deployment history. When dev teams are pushing new code out the door as frequently as possible, it can be hard to measure the impact each deployment is having on performance. One way to stay in tune with how these changes are affecting your application is via deployment reports. Deployment reports can be used to see all recent deployments and their impact on end users and app servers, along with response times, throughput, and errors.

When dev teams are pushing new code out the door as frequently as possible, it can be hard to measure the impact each deployment is having on performance.

3. Follow alerting patterns. Good alerting is often about notifying individuals or teams about changes in the systems for which they are responsible. Following an alert from the initial warning to its final resolution and examining patterns can be invaluable in helping avoid alertable situations in the future.



WRITTEN BY CHRISTINE SOTELO

SENIOR PRODUCT MARKETING MANAGER
NEW RELIC

New Relic APM by New Relic



New Relic APM surfaces issues you cannot see coming, helping your team reduce resolution times so they can focus on writing new code, not troubleshooting it.

CATEGORY

APM

LANGUAGE SUPPORT

- Java
- Ruby
- Python
- Node.js
- .NET
- SQL

ALERTS

- Threshold alerts
- Limit alerts
- Status alerts
- Conditional alerts
- Alert scheduling

CASE STUDY

Code.org is a non-profit organization dedicated to expanding participation in computer science by making it available in more schools and increasing participation by women and underrepresented students of color. When Code.org announced a new learning event called the Hour of Code in 2013, it set the bar even higher for its engineering team. The idea was to reach tens of millions of students during a one-week period with videos and tutorials designed to get them interested in learning more about computer science. With New Relic installed, the Code.org engineering team began tuning database queries and optimizing the website to handle the expected spike in traffic to the site. By the time the event kicked off, the team felt ready. Today Code.org uses New Relic Browser, New Relic APM, New Relic Server, and New Relic Platform plug-ins to monitor its website, the web-based tutorials on the site, and backend processes related to the website architecture.

FEATURES

- Real-user monitoring
- Provides SQL analysis
- Transaction tracing and breakdowns

CUSTOMERS

- Strava
- Kickstarter
- rdio
- Zendesk
- Trulia
- Groupon
- Nike
- ShopTo

BLOG blog.newrelic.com

TWITTER [@newrelic](https://twitter.com/newrelic)

WEBSITE newrelic.com

Chasing Performance Phantoms:

Looking Beyond “Why the Slowest Thing is Slow”

BY JON C. HODGSON

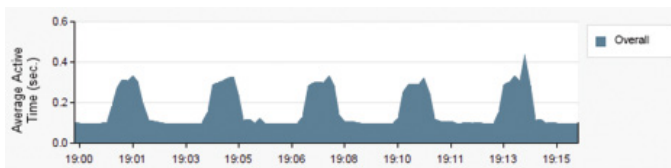
Troubleshooting application performance issues in virtualized environments can be extremely challenging. If you don't use appropriate techniques, you can be easily misled and waste countless hours trying to solve a symptom instead of the true root cause. I've spent over a decade helping organizations solve complex issues in mission critical applications, and I repeatedly encounter cases where troubleshooters are misdirected by virtualization.

Rather than discussing this topic anecdotally or theoretically as many other articles have already done, I will take you through a practical example and teach you some tricks that you can immediately employ in your own environment.

PRELUDE: A COMMON SCENARIO

Hypervisor resource contention can cause VMs running mission critical applications to pause for fractions of each second, resulting in increased response time. Traditional monitoring software and methodologies will often incorrectly categorize this as “slow code,” leading troubleshooters to conclusions that never result in a resolution of the underlying issue.

As an example of what we'll discuss in this article, consider the following application, which has spikes in response time every couple of minutes:



With traditional APM workflows, you'd probably determine that a particular piece of code is causing the spikes and then work with development to optimize that code.

However, if you take the right approach to troubleshoot this issue, you'll find there are actually two different behaviors at play:

QUICK VIEW

01

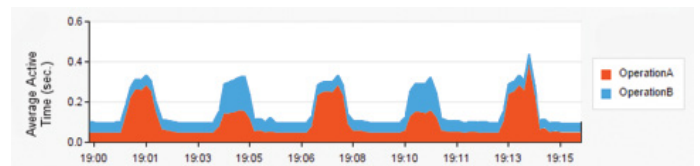
Targeting just the slowest transactions for optimization may not improve application performance as a whole. You need to step back and analyze the top delays for all transactions collectively.

02

Some code delays may be side effects of environmental contention and can't be solved with code optimization.

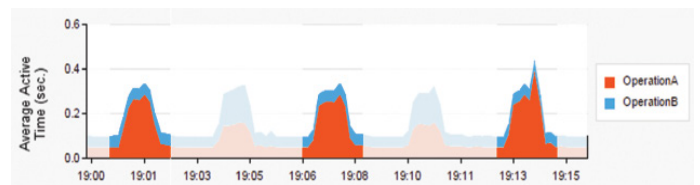
03

Extra diligence is required in virtualized environments where hypervisor overcommitment can cause application slowness disguised as code delays.



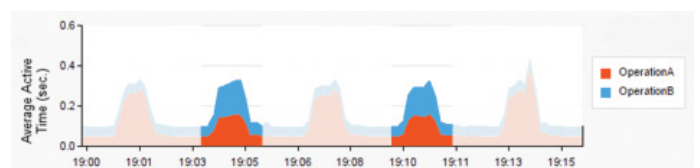
Notice that alternating spikes have a different ratio between sub-operations A & B.

The first behavior shows an increase in operation A's response time:



This is probably a code issue since it's consistently related to operation A.

However the second behavior shows a proportional increase in both operations A & B:



This is probably due to some overarching environmental issue rather than the code itself.

If you incorrectly assume that all the spikes are due to the same issue, then whichever spike you investigate and resolve will only solve part

of the problem. This article will walk you through how to effectively identify the kind of problem(s) you're dealing with, and how to determine if virtualization is playing any part in the slowness.

TEST APPLICATION

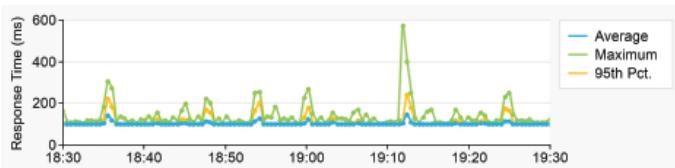
Complicated real-world applications can make it difficult to initially understand the core concepts at play. If you don't know precisely what should be happening in your application, then it's hard to discern if what you're seeing in your APM software makes sense. For this article I wrote a simple Java web page to act as a stand-in for a mission critical application function:

Class	Method	Category	Response Time (s)	0s	1s
com.rvbd.test.servlet.ImportantPage	service	Servlet	0.1	<div></div>	
com.rvbd.test.Utils	!DoOperationA	OperationA	0.05	<div></div>	
com.rvbd.test.Utils	!DoOperationB	OperationB	0.05	<div></div>	

The page executes two sequential sub-operations (Java methods). Each one normally takes about 50ms to complete, so the page should typically take around 100ms to load.

LOAD TEST

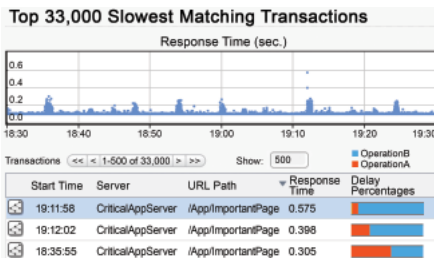
I ran a load test against this page for one hour, generating over 30,000 requests. I monitored the details of every transaction with my APM software, and trended the response time:



Since "The Flaw of Averages" can mask intermittent issues, I plot the 95th percentile and maximum response times as well to ensure I get an accurate perspective. Although the average is typically around 100ms, there are occasional periods where some of the executions are significantly slower.

DRILLING DOWN

I will now switch gears from a high-level summarization to a detailed analysis of individual transactions. Since my APM software captures all transactions and stores them in a modern data warehouse, I am able to see a precise scatter chart on how each one of the 33,000 executions behaved. I can see at a glance the top categories of delay for each transaction (left):



Immediately the orange/blue ratio reveals that the slowest transactions no longer have a 50/50 split between the two sub-operations. Drilling down into the slowest transaction confirms this:

Class	Method	Category	Response Time (s)	0s	58s
com.rvbd.test.servlet.ImportantPage	service	Servlet	0.575	<div></div>	
com.rvbd.test.Utils	!DoOperationA	OperationA	0.05	<div></div>	
com.rvbd.test.Utils	!DoOperationB	OperationB	0.523	<div></div>	

OperationA still took exactly 50ms, while OperationB is 10x slower. The next slowest transaction tells a similar story:

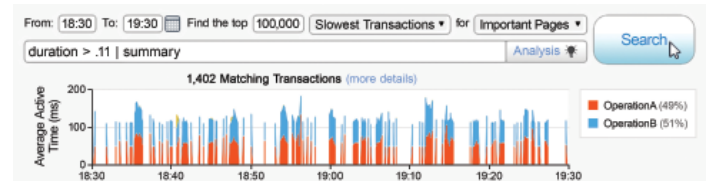
Class	Method	Category	Response Time (s)	0s	4s
com.rvbd.test.servlet.ImportantPage	service	Servlet	0.398	<div></div>	
com.rvbd.test.Utils	!DoOperationA	OperationA	0.101	<div></div>	
com.rvbd.test.Utils	!DoOperationB	OperationB	0.297	<div></div>	

At this point, many of you are probably thinking, "this is basic, tell me something new... I already do this!"

This is indeed the common workflow most people use for APM. When there's slowness, you drill down to determine "why is the slowest thing slow?" so you can then tell the developer to optimize the offending code. The problem with this approach is that the reason why the slowest things are slow may not be the same reason the application is slow overall. You need to step back and analyze the top delays of all the slow transactions collectively.

HOLISTIC ANALYSIS

Since my transaction should typically complete in ~100ms, I scope my analysis to the transactions slower than 110ms:



This reveals something unexpected: as a whole, the slowest transactions still have a 50/50 split between operations A & B, and neither is predominantly slow. The initial "Why is the slowest thing slow?" conclusion doesn't apply to the application as a whole.

TAKING A STEP BACK

Since APM software is so powerful at diving into the code and identifying slow methods and SQL queries, we sometimes forget to take a step back and consider the environment in which our code runs.

CRITICAL APP SERVER



The page we've been analyzing runs inside a JVM and is dependent on the JVM's resources such as heap, threads, etc. If a garbage collection occurs, then the entire JVM pauses and every transaction that was running at that time gets a little bit slower.

That JVM is a process in the operating system, and is dependent on the OS' resources such as CPU, RAM, Disk, etc. If

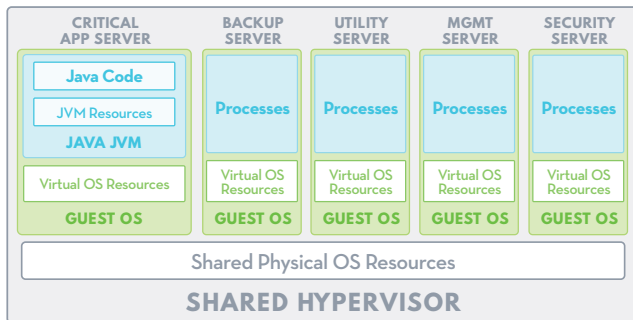
one of these resources becomes saturated, then the JVM will have to wait, and any transaction that needs that resource will get a little bit slower.

STEPPING BACK EVEN FURTHER

In virtualized environments, we have to remain cognizant that our servers live in a complex ecosystem with other systems that share the physical resources of the host. The critical application we've been analyzing runs in a VM on the same hypervisor as

four other IT operations servers, which are comparatively less mission critical (below).

Just as the JVM is dependent on the resources of the OS, the OS is dependent on the resources of the hypervisor. If the physical resources become saturated or unavailable, that will result in delays in the VMs, JVMs, and applications that run on top of them.



TRUST BUT VERIFY

There are many ways virtual environments can be configured. By default, all VMs on a host share all physical resources and the hypervisor figures out who gets what resources and when. VMs can be given dedicated resources, or sets of VMs can share a dedicated portion of resources.

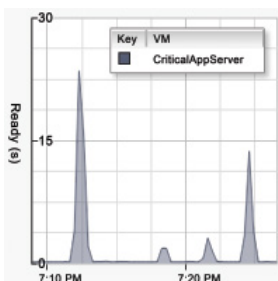
One common misconception is that if your guest OS shows four cores, it must have four cores dedicated to it, but that's often not the case. I repeatedly encounter situations with my customers where they swear that the VM administrators gave them "dedicated resources" for their mission critical, production application, only to find out that their "four core server" was capped at 2GHz. It's imperative to verify that your VMs are consistently getting the resources they need, or else your applications will incur delays due to contention or limits.

VERIFYING YOUR RESOURCES

My APM software can monitor the VM from inside the guest OS or remotely via the hypervisor, but since this is so critical, I want to show you other ways of investigating this in case your APM software doesn't provide this visibility. For this example I'll focus solely on CPU, but other resources, such as memory and storage, should also be verified in the real world.

HYPERSVISOR MONITORING TOOLS

If you have access to your hypervisor's monitoring software then it's easy. The application we've been analyzing runs on VMware vSphere, which has a great client that shows the utilization and performance of the hypervisor and all VMs running on it. The first metric I look at is **Ready**, which shows how much time a VM was unable to run because it couldn't get access to the physical CPU. This is the percentage of time the VM was paused.

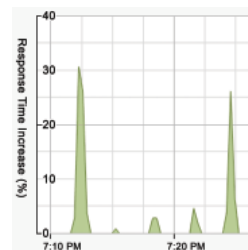


Here is **Ready** for the CriticalAppServer during the load test (left):

The way you interpret the value is to compare it to the number of cores in the VM multiplied by the sampling interval of the chart. In this case it's $4 \text{ cores} * 20 \text{ seconds}$ which means the VM can run a maximum of 80sec in between each sample.

At 7:12PM **Ready** spiked to 24.5 seconds, which is 29% of the maximum. Everything that depends on CPU—the OS, the JVM, and the application code, were completely paused for 29% of those 20 seconds. On average, all the code that ran during that period would be roughly about 29% slower, but individual executions could be much higher.

Let's compare the average response time increase observed by the APM software for the same time range, rolled up at the same 20 second interval:

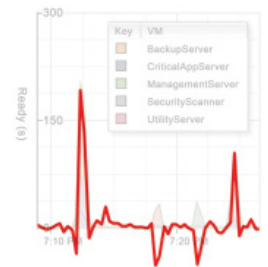


The spikes line up perfectly and the response time at 7:12 PM increased by 31%, closely matching the **Ready** value at that time. This confirms that the increase in response time for both operations A & B is actually a symptom of hypervisor pauses, rather than the root cause. Had we not looked beyond "why is the slowest thing slow?"

we would have chased a phantom and never solved the problem.

VM GUEST METRICS

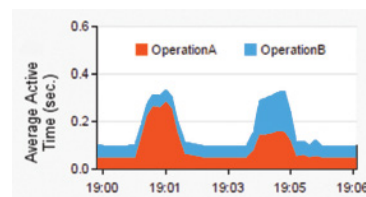
The previous method is best, but quite often you won't have access to the hypervisor monitoring tools. Some virtualization platforms have APIs that expose metrics about the guest from within the guest itself. For example, vSphere has the VMware Tools package, which is typically installed in every VM. On Windows, it publishes a handful of counters to WMI that you can view and collect with Performance Monitor (Perfmon). The counter that most closely matches **Ready** is **VM Processor \ CPU stolen time**. Here it is overlaid on **Ready** (right):



This counter is a good indicator that resource contention is occurring, but you may not be able to precisely quantify the severity.

RECOGNIZING THE SIGNS

If you don't have access to any VM statistics, with experience you can identify that contention may be occurring just from the APM performance data. Now that we understand what hypervisor contention looks like, let's review the example at the beginning of the article:



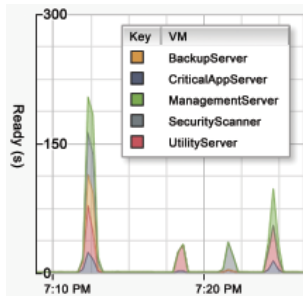
The spikes with a disproportional increase in operation A are most likely due to a code issue, but the spikes with a proportional increase in both operations A &

B are probably due to some sort of overarching contention that's affecting all code. Next you rule out the usual suspects: if it's not garbage collection, and the OS doesn't look saturated from the inside, then the VM is likely encountering contention beyond your field of view. You then have justification to ask the virtualization team to investigate this further.

IDENTIFYING THE ROOT CAUSE

While we now know that hypervisor contention was the reason for the slow pages, that's still just a symptom of a larger issue. To determine what's causing the contention, you'll need to look at the

utilization of the hypervisor and the other VMs running on it. Here is a stacked chart of **Ready** for all running VMs:



The issue is not due to any single VM, but rather when multiple VMs demand a lot of CPU concurrently. The 7:12 PM spike was due to all five VMs demanding CPU simultaneously, far exceeding the physical cycles available.

This type of synchronization issue is a common cause for many of the problems I investigate. When virtualization admins do the math on how many VMs they can squeeze into a server, they often base that information on long-term utilization reports, which have a tendency to “deflate” intermittent spikes. On average, each of these VMs uses very little CPU, and their individual peaks are quite low, so those are the workload numbers that get factored into the provisioning equation. Occasionally, bad luck inadvertently strikes when multiple VM spikes stack up and exceed the expected demand.

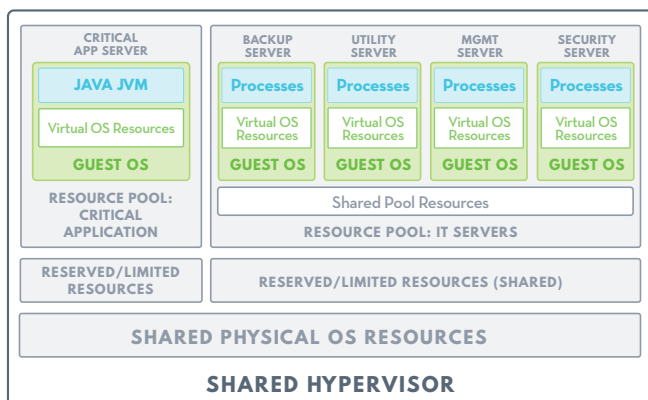
INFINITE MONKEYS

If you consider how complex modern applications are, with tens of thousands of unique pieces of code, running in infinitely varying combinations of thousands of simultaneous executions every second, with 86,400 seconds each day, and then compound that with dozens or even hundreds of different applications sharing physical resources, it's the equivalent of the *infinite monkey theorem*; it's extremely likely that some unexpected combination of events will create a demand for resources orders of magnitude higher than anyone anticipated, if only for a fraction of a second. That's enough to make a transaction that normally completes in less than a second take 5x or 10x longer. Quite often these random events are the reason behind lingering complaints of slowness that never get resolved.

RESOLVING THE ROOT CAUSE

We identified that four IT Operations VMs are periodically stealing resources from our mission critical application, so how do we prevent that from happening again in the future? You must isolate your mission critical VMs so that less critical VMs can't step on their toes.

I reconfigured my hypervisor and created two independent resource pools, one dedicated to my Critical App Server and another for the IT Operations VMs:

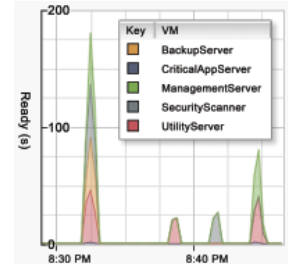


Both pools were configured with *reserved* resources to ensure that there would always be a certain amount available, as well as resource *limits* to ensure that an unexpected series of events in one VM will have limited impact on the rest of its peers. We had a phrase in the Army: “locks keep honest people honest.” Setting resource allocation limits is an effective way to compel your VMs to behave, and limit the impact if they don't.

VALIDATE THE FIX

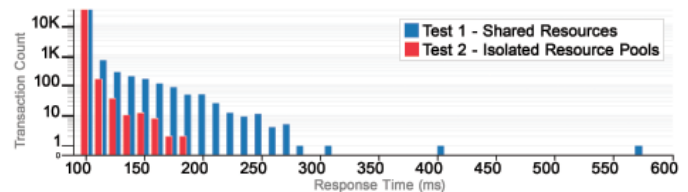
After reconfiguring the hypervisor, I repeated the load test and reviewed all the key stats again.

The synchronized demand spikes for the IT Operations VMs are still occurring, but now they have almost no impact on the Critical App Server.



While the hypervisor stats look good, success is truly achieved only when the pages are no longer slow. To verify this, I use my APM software's histogram analysis to compare the response time distribution for 33,000 executions in each test.

Test 1, with shared resources, had many executions between 200-300ms, and some outliers taking almost 600ms.



In **Test 2**, with isolated resource pools, the slowest transaction was less than 200ms, with a much tighter distribution around the expected 100ms best case.

CONCLUSION

Virtualization is a wonderful technology that allows us to efficiently utilize physical resources and dynamically adapt to changes in demand and workloads. However, like any powerful weapon, it must be wielded thoughtfully and carefully. Those of us responsible for monitoring and improving the performance of applications in virtualized environments must ensure that we have the proper visibility into the physical realm to verify that our applications are getting the resources they need to perform their functions in a consistently timely manner.

If I could have you take away one thing from this article, it would be the idea that you shouldn't shortsightedly use your APM software to simply determine “*why is the slowest thing slow?*” You need to broaden your methodologies to holistically analyze the root cause of overall performance issues rather than a tiny subset of bad examples skimmed off the top. Leveraging modern Big Data approaches to APM will allow you to quickly identify meaningful fixes that will yield the largest overall benefit for your application and its user community.



JON C. HODGSON is an APM subject matter expert for Riverbed Technology. For over a decade, Jon has helped hundreds of organizations around the world optimize the reliability and performance of their mission critical applications.

THE SECRET HEART OF THE MODERN WEB

**Building a great application is just the beginning.
NGINX Plus delivers your application fast and securely.**

NGINX POWERS 1 IN 3 OF THE WORLD'S BUSIEST WEBSITES FROM AIRBNB TO UBER.



LOAD BALANCING

Optimize the availability of apps and services with HTTP & TCP load balancing.



WEB SERVER

Deliver assets with unparalleled speed and efficiency.



CONTENT CACHING

Accelerate local origin servers and create edge servers.



STREAMING MEDIA

Stream high-quality video on demand to any device.



MONITORING

Monitor instances with detailed activity monitoring and health checks.

**DISCOVER THE SECRET TO MAKING YOUR SITE AND APPS
PERFORM BETTER. LEARN MORE AT [NGINX.COM](https://nginx.com).**

(C) 2015 NGINX, Inc. All rights reserved.

NGINX

Building a Great App Is Just the Beginning;

How Do You Deliver It at Scale?

Today's businesses face disruption and opportunity on a scale we have never seen before. As developers and technology professionals, we find ourselves at the center of this storm. No matter how sophisticated, functional, and beautiful a great application may be, if it does not perform in the real world, for real users, it has not succeeded.

What sets top development teams apart from the rest is the ability to innovate and adapt, to build and deliver extraordinary new products and experiences faster than the competition.

What can you do to harden and prepare your application for the onslaught of traffic it will encounter in production?

There are many best practices and steps that you can take to make an application production-ready. Caching, load balancing, HTTP optimizations, request scrubbing, and offloading operations like compression and encryption—these all contribute to improved performance, resilience, and control:

1. **Caching** reduces application load and delivers content faster to end users.
2. **Load balancing** distributes traffic across application instances to optimize performance and avoid failed or underperforming servers.
3. **HTTP optimizations** transform slow client traffic into fast local transactions so that applications operate at peak efficiency.
4. **Request scrubbing** cleans up traffic and drops bad requests before they can impact the application.
5. **Offloading compression and encryption** frees up application servers to focus on what they do best—running your application!

[Read more](#) of our [blog posts](#) and learn about modern application delivery at scale.



WRITTEN BY OWEN GARRETT
HEAD OF PRODUCTS
NGINX

NGINX Plus by NGINX



NGINX Plus is a software-based application delivery framework that is highly scaleable and designed for cloud and microservices architectures.

CATEGORY

Application Delivery

LOAD BALANCING

- HTTP load balancing
- TCP load balancing
- Session persistence
- Live activity monitoring

CUSTOMERS

- Airbnb
- Netflix
- Zendesk
- Groupon
- Discovery Education
- Uber
- Blue Jeans
- Gogo

FEATURES

- Load balancing
- Session Persistence
- Media streaming
- High-performance HTTP server
- Application gateway
- Web accelerator
- Advanced monitoring and management tools

HEALTH CHECK USE CASES

- Check for a response code
- Check for type of content from a specified URL or IP address

CASE STUDY

Warpwire has been working with a prominent southern university that has embraced the use of video in education in a big way. The university generates between 10 and 20 terabytes of video content each semester. Warpwire already had been using NGINX, the world's most popular open source web server for high-traffic websites, as a reverse proxy and for core app delivery. Their positive experience with NGINX made the Warpwire team look to NGINX Plus as the solution to the streaming media project. NGINX Plus provides streaming media delivery capability, along with other enterprise-ready capabilities that give developers freedom to innovate without infrastructure constraints. Warpwire now had what it needed to stream all common video formats, including Apple HTTP Live Streaming (HLS), Adobe HTTP Dynamic Streaming (HDS), and Real Time Messaging Protocol (RTMP), quickly and reliably, no matter how high the volume of content being streamed.

BLOG NGINX.com/blog/

TWITTER [@NGINX](https://twitter.com/NGINX)

WEBSITE NGINX.com

MOST VALUABLE BLOGGER ARTICLE

How to Define Performance Requirements

BY NIKITA SALNIKOV-TARNOVSKI

"The app is slow; can you make sure it is fast?"

- A business owner

The quote above should send shivers down your spine if you're an experienced engineer. I have always stressed the idea of "measuring not guessing" when dealing with performance tuning. This means that you need a clear definition of what your manager means by "fast." Without that definition, you could be spinning your wheels forever in the optimization cycle since every non-trivial application can always be made *faster* in some regard.

In the real world, performance is not the only task on our plate; in order to provide the most value, we should know when to stop the performance optimization. More importantly, we should know exactly what our performance goals are and have a clear path to meeting them.

BADLY DEFINED PERFORMANCE REQUIREMENTS

Business owners have become much better at expressing the functional requirements for software. But when we look outside of the functional requirements—be it usability, compatibility, or performance—the mind of a business owner often draws a blank. This blank spot can take the form of "make sure it is fast" or in a better case, you might be given requirements resembling these:

- *95% of the operations carried out in the system must respond within 5 seconds*
- *The system has to support 100 concurrent users*

QUICK VIEW

01

Different categories of operations need to have different latency distributions in your overall performance requirements.

02

There must be agreement as to where the latency is measured. For each type of operation, you need to decide if it is measured at the end user environment, at the server side, or somewhere else.

03

Throughput and load-based performance requirements should focus on specific user behaviors in a given timespan (e.g. 1K invoices/min).

04

Infrastructure constraints should be determined before specifying performance requirements.

At first glance you might think, not so bad. Instead of just saying "make it fast," you now have a clear target, don't you? As it turns out, the targets above are even worse than just saying "make it *fast*." Now that it contains some numbers, it looks like it is something you could use as the ultimate goal; but in reality the two requirements above are, at best, just a foundation from which you can start asking better questions. Let me unpack what is wrong with these requirements.

"95% OF THE OPERATIONS MUST RESPOND WITHIN 5 SECONDS."

What is supposed to happen with the remaining five percent? Is the goal to fit the response times to 10 seconds or is it okay to timeout these connections? Instead of setting one time as the only goal, you should set an acceptable latency distribution.

Another issue with this requirement is that it treats all operations as equivalent. Is it okay to have 95% of the operations respond in 4.9 seconds, even though many could go faster? Take these operations for example:

1. **Show my current account balance:** Executed millions of times a day, this operation is the first question each and every retail client has while interacting with their bank.
2. **Show all transactions in 2015:** This operation is only done by a few users each day.

Obviously you would need to treat the first operation differently and have more demanding targets, while the second operation could have more relaxed requirements. So instead of treating all operations equally, you should **set acceptable latency distributions for each operation type**.

You should also **link your latency requirements to load/throughput requirements**. So find out what the load in the system is while the measurements are being made, and find out how many other operations could be taking place simultaneously. Then use that information to build your latency requirements

Be precise about the layer in which the latency is measured.

Is the response time measured in the end-user environment (such as the browser rendering the response or an Android app updating the results) or is it measured when the last byte is sent out from the server side?

MOST SOFTWARE CAN ALWAYS BE TWEAKED TO GO FASTER. THE QUESTION IS WHETHER IT IS ECONOMICALLY VIABLE.

Lastly, you should **identify which operations don't need to worry about latency**. Batch jobs and asynchronous processes are good examples. A monthly batch job running for two hours calculating the final credit card balances shouldn't be considered a violation of the five second threshold. The full account CSV statements sent via email 10 minutes later, which are also compiled asynchronously, shouldn't be violations either.

"THE SYSTEM HAS TO SUPPORT 100 CONCURRENT USERS."

If you imagine a site with 100 users on it, each clicking on static images every 10 seconds served via CDN—I bet you can build a system like that with your eyes closed. But 100 users simultaneously encoding 4K video files on your site is a whole other ballgame.

Things turn from ambiguous to meaningless when thinking in terms of real concurrency, such as translating the "100 concurrent users" to "100 operations being processed concurrently by 100 threads." Assuming each such operation takes ten seconds to process, then the throughput of your system is ten operations per second. If you now reduce the operation duration tenfold, with each operation taking just one second, you have improved the throughput to 100 ops/sec. However, you are not fulfilling your "100 real concurrent users" requirement and are only processing ten operations concurrently, which means you failed to meet the requirement.

Instead of "concurrent users" or any similar terms, the **requirements should more clearly express the behavior of certain users**, with the potential of turning these descriptions into load tests allowing you to emulate the necessary load.

Note that I am not recommending you measure throughput. That wouldn't be helpful because real-world applications are often multi-functional and are being used in very dynamic situations. This makes it hard to express the performance goals in throughput (operations per hour). But if a particular application is designed to do just one thing, for example

invoice payments, then **having goals measuring throughput similar to "1000 invoices/minute" is an excellent way to have measurable and specific goals**.

CAPACITY PLANNING

Capacity planning is another major performance requirement you should precisely specify. Is your team expected to achieve goals like the ones set earlier in the article with ten thousand accounts and ten million transactions in the database, or is the system expected to fulfill those criteria with one million accounts and one billion transactions? **Be clear about the volume of data present in the system.**

Identify the financial constraints regarding the infrastructure as well. Are you expected to achieve your performance requirements with \$500/mo worth of AWS, or can you go berserk and deploy the solution on the high-end stuff with 32 cores and several TBs of memory? Knowing the answer helps you to understand what performance goals your infrastructure will allow. So, before you draft your performance requirements, you should **specify the constraints from your infrastructure**.

The network bandwidth of your customers will also be a constraint to consider in your performance requirements. Will the network bandwidth be acceptable to send several MBs back and forth during each operation? With the widespread adoption of mobile apps, you cannot count on the almighty 4G being present for every customer, so you should probably build a set of offline operations that can be handled locally and squeeze the traffic into kilobytes instead of megabytes.

CONCLUSIONS

The list of aspects described in this post is by no means complete. There are numerous performance considerations around scalability and availability that need to guide your requirements as well. Even so, you should be better prepared to scrutinize vague performance requirements when they arise and list off a series of questions that will start drilling into the actual needs. Work with the business owner to discover measurable and specific goals. Without such goals, you have no real target to meet and measure your results against.

Walking through the process allows you to explain the related costs as well. Business owners are always eager to get better insight into those! If you recall, most software can always be tweaked to go faster. The question is whether it is economically viable. From the business owner perspective, it is only natural to want all the operations to be ultra fast. It's only when we understand the costs of achieving this that more realistic expectations can be set.



NIKITA SALNIKOV-TARNOVSKI is the co-founder of [Plumbr](#), a Java performance monitoring solution, where he now contributes as a core developer. Besides his daily technical tasks, he is an active blogger, JavaOne RockStar, and frequent conference speaker.

Is your site performing for mobile users?

Even in Mumbai?

Get real user data with mPulse.™

In an online world where milliseconds add up to millions, performance is everything. mPulse delivers detailed views of every user experience on every page so you always know what's working, what isn't and why. A unified dashboard lets you measure performance by geography, OS, browser or network, then isolate and correct issues on an object-level, instantly. So instead of guessing at outcomes, you're optimizing in real time.

Get started for free at www.soasta.com/mPulse



Synthetic Versus Real-User Monitoring?

Here's Why You Need Both.

One of the most common questions that people have when they look at their synthetic and RUM numbers side by side is, "why are they so different?" That's because an average is only a calculation on a page. There's no such thing as an average user. Every user experience is different, and every site delivers both great and terrible user experiences to users every day.

Your RUM numbers illustrate the broad spectrum of user experiences. Your synthetic numbers give you snapshots of user experiences within very specific parameters.

Synthetic tests can help you answer questions such as: how does the design of your page affect performance? Are you sticking to your performance budget? How does the newest version of your site compare to earlier versions? How do you compare to your competitors?

RUM gathers data from every user, anywhere in the world. Today, the best RUM tools have powerful analytics engines that allow you to slice and dice your data in endless ways. RUM can teach you a lot about how people use your site, uncovering insights that would otherwise be impossible to obtain.

Your RUM data can answer questions such as: what are your users' environments? Where are they coming from? What are their most common user paths? Where are your site's performance pains? What impact does performance have on your bottom line?

There's no one-size-fits-all monitoring tool. Understand what each type of measurement tool can help you with, and use each one's strengths to your advantage.



WRITTEN BY TAMMY EVERTS

SENIOR RESEARCHER & EVANGELIST
SOASTA

mPulse by SOASTA

SOASTA

mPulse bridges web performance with business KPIs so you can optimize and fix performance issues before they hit users.

CATEGORY

FEO

LANGUAGE SUPPORT

• JavaScript

ALERTS

- Threshold alerts
- Limit alerts
- Status alerts
- Conditional alerts
- Alert scheduling

CASE STUDY

A Fortune 50 company was losing revenue on its eCommerce site because of downtime. The new SVP of eCommerce turned to SOASTA to see if we could help. An assessment of the current situation confirmed his suspicion that performance testing was an afterthought even though the company has 45-47 releases a year. They started using the mPulse results as a "fix list." It pinpointed performance problem areas and built a tangible list of what to fix. Another key change was moving the performance engineering process to the initial development stage of the SLC. The results were impressive. A reduction in downtime by 80%, a reduction in defect backlog by 89%, and a ROI of 2424% due to increased revenue from an increase in uptime.

FEATURES

- Synthetic monitoring
- Real-user monitoring
- Load testing

CUSTOMERS

- SuccessFactors
- TurboTax
- SquareSpace
- NASA

- SAP
- Netflix

BLOG performancebeacon.com

TWITTER [@CloudTest](https://twitter.com/CloudTest)

WEBSITE SOASTA.com/products/mpulse/

Front-End Optimization Checklist

HTML/CSS/JS

- ☐ Use gzip compression on all HTML, CSS, and JS files
- ☐ Create a combined JavaScript file and combined CSS file
- ☐ Minify HTML, CSS, and JavaScript
- ☐ Load CSS before JavaScript

HTML

- ☐ Pre-fetch assets
- ☐ Specify a character set

CSS

- ☐ Remove inline style blocks and use `<link>` CSS files in the `<head>`
- ☐ Remove unused CSS
- ☐ Use efficient CSS selectors
- ☐ Avoid CSS `@import`
- ☐ Avoid CSS expressions

JavaScript

- ☐ Load 3rd-Party assets asynchronously
- ☐ Defer loading JavaScript not executed onload
- ☐ Defer parsing of JavaScript until absolutely necessary
- ☐ Use intelligent script loaders for parallel async processing
- ☐ Avoid using `document.write()`

Images

- ☐ Combine images using CSS sprites
- ☐ Compress images
- ☐ Specify image dimensions
- ☐ Serve scaled images
- ☐ Use data URIs for smaller images

SVG

- ☐ Use vector images where possible (icon fonts or SVG)
- ☐ Optimize SVGs with an SVG cleaning tool

URLs

- ☐ Remove broken links, missing images, and any other bad requests
- ☐ Serve resources from multiple hostnames for parallel processing
- ☐ Serve assets from a single URL
- ☐ Serve static content from a cookieless domain
- ☐ Reduce the number of unique hostnames that serve assets
- ☐ Minimize redirects

Caching

- ☐ Use a Content Delivery Network (CDN)
- ☐ Use browser caching
- ☐ Make redirects cacheable
- ☐ Enable public caching in the HTTP headers of static assets
- ☐ Use Google Library APIs when possible

Goals

- ☐ Start render time under 2 seconds
- ☐ Single pages smaller than 500KB

For more information on how to accomplish many of these checklist items, visit [Google's PageSpeed documentation](#) and read Yottaa's [optimization checklist](#).

SOURCES

developers.google.com/speed/docs/insights/rules
yottaa.com/blog/application-optimization/bid/296075/Infographic-The-Ultimate-Checklist-for-Optimizing-Web-Perfo
websiteperformancechecklist.com
frontendtest.com/checklist

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.

For a deeper look into a small selection of these products, see [page 42](#).

PRODUCT/COMPANY	CATEGORIES	FREE TRIAL	HOSTING	WEBSITE
AccelOps	ITOA	30 days	SaaS	accelops.com
Akamai Ion	Network Monitoring, FEO, CDN	Upon request	SaaS	akamai.com
Amazon CloudFront	CDN	Free tier available	SaaS	aws.amazon.com
Amazon CloudWatch	Infrastructure Monitoring, Network Monitoring	Free tier available	SaaS	aws.amazon.com
Apica Systems	APM, Infrastructure Monitoring	Limited by usage	SaaS	apicasystems.com
AppDynamics	APM, Database Monitoring	15 days	SaaS	appdynamics.com
AppFirst	APM, Infrastructure Monitoring, ITOA	30 days	SaaS	appfirst.com
AppNeta TraceView	APM, ITOA	Free tier available	SaaS	appneta.com
Appnomic AppsOne	ITOA	Upon request	On-premise or SaaS	appnomic.com
Apptio	ITOA	Upon request	SaaS	apptio.com
Aternity	APM, ITOA	Upon request	On-premise	aternity.com
Bay Dynamics	ITOA	5 days	On-premise	baydynamics.com
BigPanda	ITOA, Alert Software	21 days	SaaS	bigpanda.io

PRODUCT PROFILES CONTD.

PRODUCT/COMPANY	CATEGORIES	FREE TRIAL	HOSTING	WEBSITE
BlueStripe FactFinder	APM, Infrastructure Monitoring	Upon request	On-premise or SaaS	bluestripe.com
BMC TrueSight	APM, Network Monitoring, ITOA	30 days	SaaS	bmc.com
Boundary	APM, Infrastructure Monitoring	Free tier available	SaaS	boundary.com
BrowserStack	FEO	Limited by usage	SaaS	browserstack.com
CA APM	APM	30 days	On-premise	ca.com
Catchpoint Suite	APM, Infrastructure Monitoring, FEO	14 days	SaaS	catchpoint.com
Cedexis	Infrastructure Monitoring, Network Monitoring	Upon request	SaaS	cedexis.com
Circonus	Infrastructure Monitoring, ITOA	Free tier available	SaaS	circonus.com
CloudFlare	FEO, ITOA, CDN	Free tier available	SaaS	cloudflare.com
CorrelSense SharePath	APM, Network Monitoring, Middleware Monitoring	Upon request	On-premise	correlsense.com
CoScale	ITOA	30 days	SaaS	coscale.com
Datadog	Infrastructure Monitoring	14 days	SaaS	datadoghq.com
Dotcom Monitor	APM, Infrastructure Monitoring	30 days	SaaS	dotcommonitor.com
Dyn	Infrastructure Monitoring, Network Monitoring, ITOA, Managed DNS	7 days	SaaS	dyn.com
Dynatrace Application Monitoring	APM	Upon request	SaaS	dynatrace.com
eg Innovations Monitors	APM, Infrastructure Monitoring	14 days	SaaS	eginnovations.com
Evolgen	ITOA	Upon request	On-premise	evolgen.com
Extrahop Networks	ITOA	Free tier available	SaaS	extrahop.com
F5 Big IQ	APM, Network Monitoring	30 days	On-premise or SaaS	f5.com
Fastly	CDN, APM	Upon request	SaaS	fastly.com

PRODUCT PROFILES CONTD.

PRODUCT/COMPANY	CATEGORIES	FREE TRIAL	HOSTING	WEBSITE
HP APM	APM, ITOA	30 days	On-premise	hp.com
IBM Service Engage	APM, Infrastructure Monitoring	Upon request	On-premise or SaaS	ibm.com
Icinga	Infrastructure Monitoring	Open source	On-premise	icinga.org
Idera Copperegg	Infrastructure Monitoring, Network Monitoring, APM	14 days	SaaS	idera.com
Idera UpTime Software	APM, Infrastructure Monitoring, Network Monitoring	30 days	On-premise	idera.com
Inetco Insight	APM, Middleware Monitoring	Upon request	On-premise	inetco.com
Infovista 5view	APM, Network Monitoring	Upon request	On-premise	infovista.com
jClarity	APM (JVM)	7-14 days	On-premise	jclarity.com
JenniferSoft	APM	14 days	On-premise	jennifersoft.com
Keynote Platform	APM, ITOA	Upon request	SaaS	keynote.com
Librato	Infrastructure Monitoring, ITOA	30 days	SaaS	librato.com
Logentries	Log Management, ITOA	Free tier available	SaaS	logentries.com
Loggly	Log Management, ITOA	Free tier available	SaaS	loggly.com
LogMatrix NerveCenter	APM, Infrastructure Monitoring, Network Monitoring, Database Monitoring	Upon request	On-premise	logmatrix.com
LogStash	Log Management	Open source	On-premise	logstash.net
Metafor Software	APM, ITOA	Upon request	SaaS	metaforsoftware.com
Moogsoft	ITOA	None	SaaS	moogsoft.com
Nagios XI	Infrastructure Monitoring, Network Monitoring	Open source	On-premise	nagios.com
Nastel Autopilot	APM, Infrastructure Monitoring, Middleware Monitoring	Upon request	SaaS	nastel.com
NetScout nGeniusOne	APM, Network Monitoring, ITOA	Upon request	On-premise	netscout.com

PRODUCT PROFILES CONTD.

PRODUCT/COMPANY	CATEGORIES	FREE TRIAL	HOSTING	WEBSITE
Netuitive	APM, Infrastructure Monitoring, ITOA	21 days	SaaS	netuitive.com
Neustar Website Monitoring	APM	30 days	SaaS	neustar.biz
New Relic APM	APM, Infrastructure Monitoring, ITOA	Free tier available	SaaS	newrelic.com
NGINX Plus	Application delivery	Open source version available	On-premise or SaaS	nginx.com
NSONE	Managed DNS	Free tier available	SaaS	nsone.net
op5 Monitor	APM, Infrastructure Monitoring, Network Monitoring, 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
PA Server Monitor	Infrastructure Monitoring, Network Monitoring	30 days	On-premise	poweradmin.com
PagerDuty	ITOA, Alert Software	14 days	SaaS	pagerduty.com
Performance Co-Pilot	Infrastructure Monitoring, APM	Open source	On-premise	pcp.io
Pingdom	APM, FEO	30 days	SaaS	pingdom.com
Plumbr	APM (JVM)	14 days	On-premise or SaaS	plumbr.eu
Prometheus	Infrastructure Monitoring	Open source	On-premise	github.com/prometheus
Rackspace	Infrastructure Monitoring, Database Monitoring	Limited by usage	SaaS	rackspace.com
Radware Fastview	FEO	Upon request	On-premise	radware.com
Rigor	APM	Upon request	SaaS	rigor.com
RISC Networks CloudScape	Infrastructure Monitoring, Network Monitoring, ITOA	Upon request	SaaS	riscnetworks.com
Riverbed Steelcentral	APM, Infrastructure Monitoring, Network Monitoring, ITOA	30 days	On-premise	riverbed.com
SauceLabs	FEO	14 days	SaaS	saucelabs.com

PRODUCT PROFILES CONTD.

PRODUCT/COMPANY	CATEGORIES	FREE TRIAL	HOSTING	WEBSITE
ScienceLogic EM7	APM, Infrastructure Monitoring, Network Monitoring	Upon request	SaaS	sciencelogic.com
Sematext SPM	APM	Free tier available	On-premise or SaaS	sematext.com
SevOne	Infrastructure Monitoring, Network Monitoring	Upon request	SaaS	sevone.com
Site24x7	APM, Infrastructure Monitoring, Network Monitoring	Limited by usage	SaaS	site24x7.com
SmartBear AlertSite UXM	APM, FEO, API monitoring	Free tier available	SaaS	smartbear.com
Soasta mPulse	FEO	Up to 100 users	SaaS	soasta.com
Solarwinds Network Performance Monitor	Network Monitoring, ITOA	30 days	On-premise	solarwinds.com
SpeedCurve	FEO, ITOA	None	SaaS	speedcurve.com
Spiceworks	Network Monitoring, ITOA	Free	On-premise	spiceworks.com
Splunk Enterprise	ITOA, Log Management	Free tier available	On-premise	splunk.com
Stackify	APM, Network Monitoring, Database Monitoring, ITOA	Upon request	SaaS	stackify.com
Sumo Logic	Log Management, ITOA	Free tier available	SaaS	sumologic.com
TeamQuest	ITOA	Upon request	On-premise	teamquest.com
Telerik	FEO	Free	On-premise	telerik.com
ThousandEyes	Network Monitoring, ITOA	15 days	SaaS	thousandeyes.com
VictorOps	Alert Software	14 days	On-premise	victorops.com
VividCortex	Database Monitoring	Free tier available	On-premise or SaaS	vividcortex.com
Zabbix	Infrastructure Monitoring	Open source	On-premise	zabbix.com
Zenoss Service Dynamics	Infrastructure Monitoring, Network Monitoring	Open source version available	On-premise or SaaS	zenoss.com
Zoho ManageEngine	APM, Infrastructure Monitoring	Free tier available	On-premise	manageengine.com

diving deeper

INTO FEATURED PERFORMANCE & MONITORING PRODUCTS

Looking for more information on individual APM providers? Eight of our partners have shared additional details about their offerings, and we've summarized this data below.

If you'd like to share data about these or other related solutions, please email us at research@dzone.com.

AlertSite UXM by SmartBear

DOCKER	✗	DATABASE INTEGRATIONS <ul style="list-style-type: none">• MySQL• Oracle• SQL Server• IBM• DB2• MongoDB• Cassandra• Redis• And four others
PREDICTIVE ALGORITHMS	✗	
AGENT-BASED	✗	
HISTORICAL REPORTS	✓	
CLOUD MONITORING	✓	

AppDynamics by AppDynamics

DOCKER	✗	DATABASE INTEGRATIONS <ul style="list-style-type: none">• Oracle• SQL Server• IBM DB2• MongoDB• Sybase
PREDICTIVE ALGORITHMS	✓	
AGENT-BASED	✓	
HISTORICAL REPORTS	✓	
CLOUD MONITORING	✓	

CA Application Performance Management by CA Technologies

DOCKER	✗	DATABASE INTEGRATIONS <ul style="list-style-type: none">• Oracle• SQL Server• IBM DB2• MongoDB• Sybase
PREDICTIVE ALGORITHMS	✓	
AGENT-BASED	✓	
HISTORICAL REPORTS	✓	
CLOUD MONITORING	✓	

Datadog by Datadog

DOCKER	✓	DATABASE INTEGRATIONS <ul style="list-style-type: none">• MySQL• SQL Server• MongoDB• Cassandra• Redis• PostgreSQL• Memcached• And four others
PREDICTIVE ALGORITHMS	✗	
AGENT-BASED	✓	
HISTORICAL REPORTS	✓	
CLOUD MONITORING	✓	

Dynatrace Application Monitoring by Dynatrace

DOCKER	✓	DATABASE INTEGRATIONS <ul style="list-style-type: none">• MySQL• Oracle• SQL Server• IBM DB2• MongoDB• Cassandra• Redis• And four others
PREDICTIVE ALGORITHMS	✓	
AGENT-BASED	✓	
HISTORICAL REPORTS	✓	
CLOUD MONITORING	✓	

Librato by Librato

DOCKER	✓	DATABASE INTEGRATIONS <ul style="list-style-type: none">• AWS RDS• Heroku Postgres• MySQL• PostgreSQL• Cassandra• MongoDB• Redis
PREDICTIVE ALGORITHMS	✗	
AGENT-BASED	✗	
HISTORICAL REPORTS	✓	
CLOUD MONITORING	✓	

New Relic APM by New Relic

DOCKER	✓	DATABASE INTEGRATIONS <ul style="list-style-type: none">• Mongo• MySQL• PostgreSQL• Oracle• DB2• SQL Server• SQLite• And twelve others
PREDICTIVE ALGORITHMS	✗	
AGENT-BASED	✓	
HISTORICAL REPORTS	✓	
CLOUD MONITORING	✓	

NGINX by NGINX

DOCKER	✗	DATABASE INTEGRATIONS n/a
PREDICTIVE ALGORITHMS	✗	
AGENT-BASED	✗	
HISTORICAL REPORTS	✓	
CLOUD MONITORING	✗	

SOASTA by SOASTA

DOCKER	✗	DATABASE INTEGRATIONS n/a
PREDICTIVE ALGORITHMS	✓	
AGENT-BASED	✓	
HISTORICAL REPORTS	✓	
CLOUD MONITORING	✓	

glossary

APPLICATION PERFORMANCE

MONITORING (APM) The practice of monitoring the performance and availability of software in many areas of the stack and then being able to take action on those monitored areas to improve performance.

BANDWIDTH The maximum I/O throughput of a system.

BENCHMARK A program built for any piece of software or hardware to measure its performance under specific operating conditions.

BIG-O NOTATION A notation for describing the change in efficiency of a software algorithm as you increase the number of iterations/cycles in a task. $O(n)$ will scale linearly, $O(n^2)$ will become quadratically slower, and $O(1)$ will not lose any efficiency over time.

BOTTLENECK Occurs when demand for a particular resource is beyond the capacity of that resource, which causes other system resources to be adversely affected.

BUFFER An area where data is temporarily stored while it is being moved from one place to another.

BUFFER CACHE A cache that provides random access memory (RAM) to store copies of frequently used disk sectors so they can be read without accessing the slower disk.

CACHE MEMORY A type of memory placed between a CPU and main memory that has high-speed and a low-access time. It's used for caching, a technique to save frequently used sections of code that don't change often so applications don't always have to reload those pieces.

CPU BOUND A system in which there is insufficient CPU power to run software processes on time. This results in poor interactive response times in applications.

DYNAMIC SITE ACCELERATION (DSA) Also known as whole site acceleration, this is a group of techniques and tools—

specifically, application delivery controllers and content delivery networks (CDNs)—that aim to decrease page loading and response times.

FRONT-END OPTIMIZATION A group of tools and techniques that modify web page content to reduce the amount of resources required to download a given page, thereby helping the browser process the page faster. These techniques include device-aware content serving, minification, resource consolidation, and inlining.

GARBAGE COLLECTION Also known as automatic memory management, this is the process of reclaiming the memory of objects that are no longer in use.

HIGH-PERFORMANCE COMPUTING (HPC) A field of computing devoted to maximizing the throughput and minimizing the latency involved in solving high-volume, time-sensitive problems. Practitioners work to maximize the efficiency of their code on commodity hardware.

I/O BOUND A system in which the peripheral devices cannot transfer data as fast as requested.

IT OPERATIONAL ANALYTICS (ITOA) An approach to performance management that collects, correlates, and reports on all areas of operational metrics, applying large scale data science techniques to various types of performance data.

JOB One or more processes that are grouped together but issued as a single command.

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

LOAD TESTING A method for determining how a server, network, or application performs under a specific (often at the high end of the range of expected) load/number of users. It is often performed with a tool that generates the desired number of tasks or virtual users simultaneously to see how the system responds.

LOG MANAGEMENT An approach to collecting, aggregating, and correlating large volumes of computer-generated log messages (also known as audit records, audit trails, event-logs, etc.) to better understand the causes of errors and performance problems.

MEMORY BOUND A system where physical memory is a significant limiting factor.

MEMORY LEAK Occurs when a computer program incorrectly manages memory allocations and continues to allocate memory that's no longer being used.

MICROSERVICES An application deployment model consisting of small, loosely coupled services that each perform a single function according to your domain's [bounded contexts](#).

RACE CONDITION An issue that occurs when an operation O_n whose functionality presupposes that another operation O_{n+1} is complete begins before O_{n+1} is complete.

REAL-USER MONITORING (RUM) A type of end-user monitoring that uses client-side code to record real user interactions with an application.

RESPONSE TIME The time elapsed between issuing a command and receiving some feedback from the system.

SERVICE LEVEL AGREEMENT (SLA) A contract that specifies the consumer's IT requirements and the provider's commitment to them.

STRESS TESTING A similar method to load testing that is used to test a system under extreme load conditions well over the expected maximum.

SYNTHETIC MONITORING A type of end-user monitoring that uses virtual users with pre-determined behaviors to test the performance of a system from the user experience perspective.

THREAD The smallest sequence of programmed instructions that can be managed independently by a scheduler.

THROUGHPUT The amount of work (number of jobs, disk requests, etc.) that a system processes in a specified time.

TRANSACTION MONITORING A technique for auditing the performance of individual transactions (e.g. business transactions like purchases) across all sections of the infrastructure.

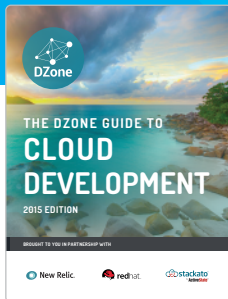
WEB CONTENT OPTIMIZATION (WCO) Optimizing HTML for fast page rendering.

WEB PERFORMANCE OPTIMIZATION A catchall term that refers to all aspects of server-side and client-side web application performance.

DZone Research Guides

Better, more concrete, and closer to reality than reports from Gartner and Forrester...

ROBERT ZAKRZEWSKI,
DZONE MEMBER



DZONE GUIDE TO Cloud Development

Get a full analysis of developer program trends and learn how you can benefit from joining the right one.

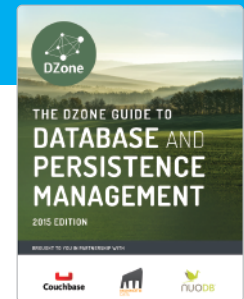
DOWNLOAD



DZONE GUIDE TO Continuous Delivery

Understand the role of DevOps, automation, testing, and other best practices that allow Continuous Delivery adoption.

DOWNLOAD



DZONE GUIDE TO Databases

Explore effective database performance strategies and resolve common database-related obstacles.

DOWNLOAD

UPCOMING IN 2015

Java Ecosystem
Software Quality

Internet of Things
Mobile Development

Big Data
Security

Get them all for free at dzone.com/research