DZone

THE DZONE GUIDE TO

Performance

Optimization & Monitoring

VOLUME III

RESEARCH PARTNER SPOTLIGHT

ThousandEyes

Executive Summary

BY MATT WERNER
CONTENT AND COMMUNITY MANAGER, DZONE

Back in the 1990s, it could take minutes to load a web page, while today it typically takes seconds. However, as Google discovered in 2012, even 400 milliseconds can be considered to be "too slow" for users, which may cause them to bounce or complain to the owners of the site they're trying to visit. Imagine how they might react when something in the application code stops them from instantly accessing the information they need by a factor of seconds or even minutes. As developers are being expected to optimize their applications for security and rapid changes through DevOps methodologies, they are also starting to become more involved in application performance optimization. To monitor the state of the industry regarding performance optimization, DZone surveyed 471 tech professionals to discover how they prepared for performance issues and how they dealt with them.

LOCKING EVERYTHING DOWN

DATA 46% of respondents build performance optimization into the development process. Of those, 30% were likely to find frequent code issues compared to 38% of developers who build functionality first, then optimize for performance. Those who bake performance into the SDLC solve performance issues 35 hours faster on average than those who do not.

IMPLICATIONS Incorporating performance optimization from the start of a project can drastically reduce headaches in the long term, and can make it easier to get users back on track as soon as possible. In addition to being faster, there are likely to be fewer failures in the first place, leading to saved cost in development time, so more resources can be allocated to different projects rather than stuck doing maintenance.

RECOMMENDATIONS Developers need to start learning about the best ways to optimize their applications from day one. The extra time it may take to do this right from the start will be worth it in saved time and costs from fixing problems down the road. Leadership teams should also be educated by performance optimization experts and project managers on the long-term benefits of incorporating performance optimization early. For an interesting case study on how Oren Eini and his team optimized the RavenDB database, check page 16.

DESIGNING FOR PARALLEL EXECUTION HASN'T CAUGHT ON YET

DATA The number of DZone users who design programs for parallel execution increased 1% over last year's survey to 44%. Of the parallel execution design techniques, load balancing was the most used at 68%. Multithreading is the most popular parallel programming model at 72%.

IMPLICATIONS Load balancing continues to be an important part of running applications in production. Parallel execution was also seen as crucial for embedded apps and high-risk software by 54% of survey respondents, since failure of these applications can lead to the loss of life. The small amount of growth between last year and this year seems to indicate that redesigning existing applications or spending time to design new applications for parallel execution seems to be less of a priority for apps where it is not seen as a crucial feature.

RECOMMENDATIONS Development and operations teams need to invest more into load balancing in order to reduce the strain on your servers as traffic comes in. Fewer constraints on your resources means you should encounter fewer performance problems related to load and traffic. For high-risk or business-critical software, consider adopting parallel execution to improve speed when it absolutely counts. Developers working on applications outside of these fields should consider these techniques and models. Not only will it be useful experience, but it also potentially pay off in application speeds and happy users.

TOOLING MATTERS

DATA 64% of respondents reported that they use between 1 and 4 performance monitoring tools. The three most popular tools are Nagios (33%), LogStash (27%), and AWS CloudWatch (21%). Those who use monitoring tools are 7% less likely to discover problems through communication with users and support tickets than those who do not use them, and were 12% less likely to accidentally encounter performance issues.

IMPLICATIONS LogStash and CloudWatch have both made large jumps in popularity since 2016 (5% and 6%, respectively), suggesting that more developers and organizations are adopting monitoring tools. These tools have proven their usefulness by helping to pinpoint performance issues before anyone notices or encounters it while using the application.

RECOMMENDATIONS The only thing better than quickly fixing a performance issue for a user is to fix it before the user can find it. Monitoring tools are becoming critically important for maintaining applications. In addition to monitoring tools, we found that those who use multiple methods or tools like application logs to find the root cause of performance problems will find the root cause of an issue faster than those who don't use monitoring tools, or only use one tool or method. For more detail, consult the Key Research Findings on the following page or Denis Goodwin's article on API monitoring on page 8.

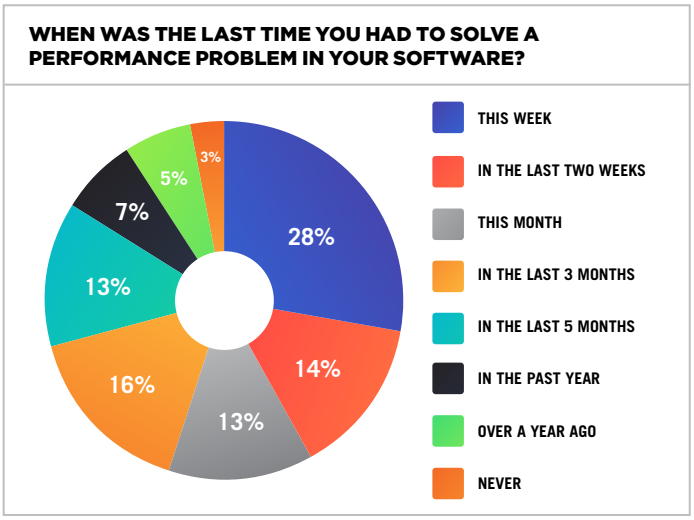
Key Research Findings

BY G. RYAN SPAIN
PRODUCTION COORDINATOR, DZONE

471 respondents completed our 2017 Performance and Monitoring Survey. The demographics of the survey respondents include:

- 24% of respondents work at organizations with at least 10,000 employees; 18% work at organizations between 1,000 and 10,000; and 19% work at organizations between 100 and 1,000.
- 37% of respondents work at organizations in Europe, and 29% work at organizations in the US.
- Respondents had 15 years of experience as an IT professional on average; 29% had 20 years or more of experience.
- 30% of respondents identify as developers or engineers; 21% as developer team leads; and 20% as software architects.
- 83% of respondents work at organizations that use Java, and 79% work at organizations using JavaScript (45% only using client-side, 3% only using server-side, and 31% using both).

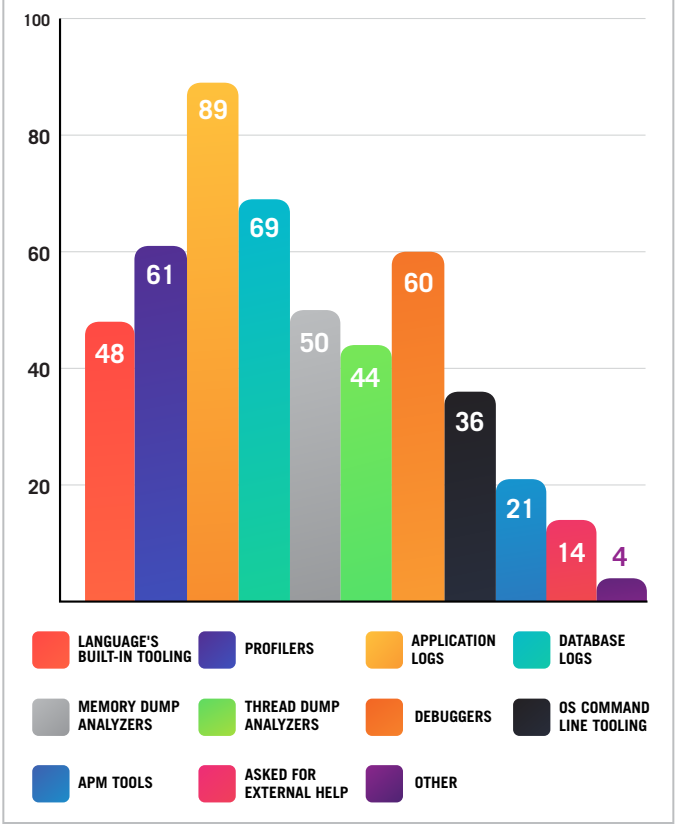
WHEN WAS THE LAST TIME YOU HAD TO SOLVE A PERFORMANCE PROBLEM IN YOUR SOFTWARE?



STARTING OFF

54% of this year's survey respondents said they worry about application performance only after they have built application functionality, a response similar to the results of DZone's 2016 Performance and Monitoring survey. However, the frequency with which respondents claimed to experience certain application performance issues was positively impacted by building performance into the application first. For example, the most frequent area for performance issues in this year's survey was application code, with 35% of respondents saying they have frequent issues with this part of their technology stack. On average, respondents who said they build performance in from the beginning of their application were 30% likely to find frequent performance issues in their application code, as opposed to 38% of respondents who worry about performance after functionality. Likewise, those who said they generally considered application performance from the beginning were able to solve performance issues 35 hours faster, on average, than those who did not (187 hours compared to 222). Of course, focusing too much on performance from the outset of a project can lead to unnecessarily lengthy design and development times, but having an idea of how performance will fit into an application from the start can save headaches later on in the SDLC.

WHAT TOOLS DOES YOUR TEAM COMMONLY USE TO FIND ROOT CAUSE FOR APPLICATION PERFORMANCE PROBLEMS?



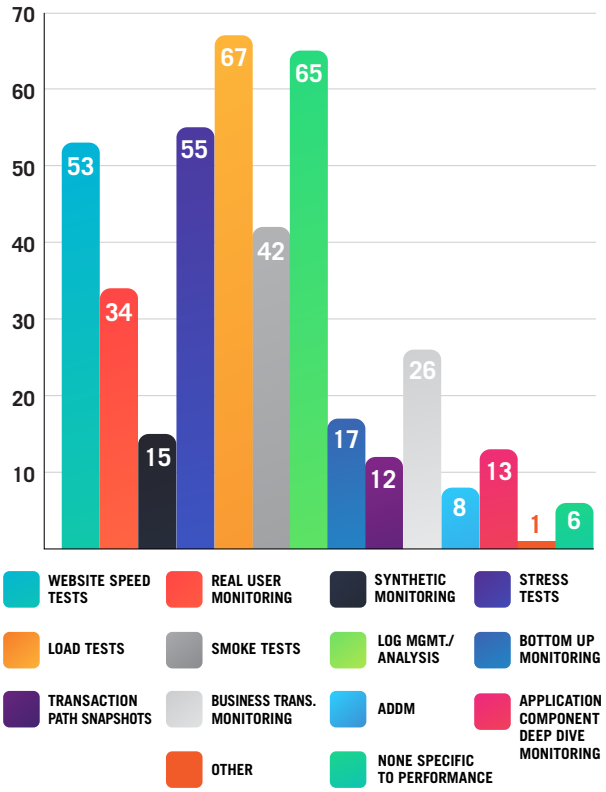
KEEPING AN EYE OUT

The majority of respondents (64%) said they use between 1 and 4 performance monitoring tools. The most popular monitoring tools were Nagios, used by 33% of respondents' organizations, and LogStash, used by 27%. Both LogStash and Amazon's CloudWatch saw significant growth from last year's results, with LogStash growing 5% and CloudWatch growing 6% to 21%, making it this year's third most popular performance monitoring tool. Increased usage of monitoring tools decreased the average estimated amount of discovering performance issues through user support emails/social media or through "dumb luck;" respondents whose organizations use 3 or 4 monitoring tools were 7% less likely to find out about performance problems from users than those who used none (17% vs. 24%), and were 12% less likely to accidentally stumble upon performance issues through dumb luck (11% vs. 23%). The most popular types of monitoring were real user monitoring (34%) and business transaction monitoring (26%).

WHAT'S THE PROBLEM?

Much like last year, finding the root cause of an issue was found to be the most time-consuming part of fixing performance-related problems. 52% of respondents ranked this as the most time consuming, followed by 25% of respondents who said collecting and interpreting various metrics took the most time. Respondents said they use a number of different tools in order to search for the root cause of performance issues. The most popular of these methods included application logs (89%), database logs (69%), profilers (61%), and debuggers (60%). Individually, none of these tools had an impact on how time-consuming respondents found root cause discovery; however, respondents using more of these tools together were increasingly less likely to find root cause discovery time consuming until peaking at 6 tools (because of a sample size of less than 1%, responses showing 0 tools used were not considered in this analysis).

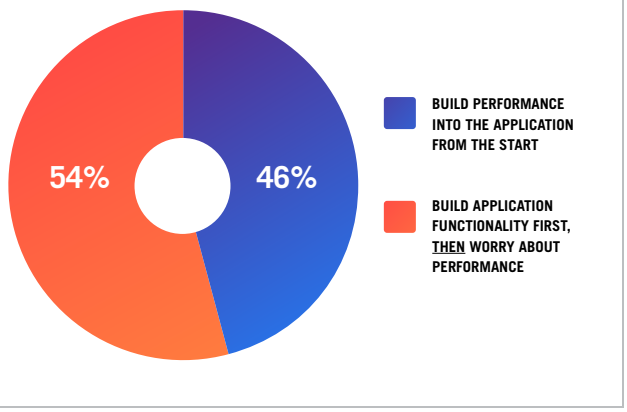
WHICH OF THE FOLLOWING PERFORMANCE TESTS AND/OR MONITORING TYPES DOES YOUR ORGANIZATION USE?



SPLITTING THE LOAD

The usage of parallel execution in application design has not taken off much since last year. 44% of respondents this year said they regularly design programs for parallel execution, only 1% higher than last year. The tools and methods for parallel design hasn't changed much either; like last year, the ExecutorService framework in Java is the most frequently used framework/API among respondents, with 50% of those who design for parallel execution regularly using this framework often. Also, load balancing is again the most popularly used parallel algorithm design technique used, with 68% of parallel execution designers using this often. And multithreading is at the top of the list for parallel programming models, with 72% of this subset of respondents using multithreading often. The choice to design for parallel execution in an application can be affected by multiple factors. For instance, the type of application being designed may increase the need for parallel execution; respondents who said they build embedded services or high-risk software (i.e. software in which failure could lead to significant financial loss or loss of life) were much more likely to regularly design for parallel execution, with over half of these respondents (54% each) answering this question positively.

HOW DO YOU PRIORITIZE PERFORMANCE IN YOUR APPLICATION DEVELOPMENT PROCESS?



Metric or Log Analytics Checklist

BY STELA UDOVICIC
SR. DIRECTOR, PRODUCT MARKETING, **WAVEFRONT**

To meet critical SLAs and maintain reliability, modern digital enterprises running applications in the cloud must measure the performance of their revenue generated essential services, distributed applications, and infrastructures. For developers, DevOps and TechOps engineers, it can be confusing to know when to use metrics or log monitoring to isolate code performance anomalies, proactively monitor and baseline their scaled out, dynamic and distributed applications.

Metrics describe numeric measurements in time. The metric format includes the measured metric name, the metric data value, the timestamp, the metric source, and an optional tag. Metrics convey small information bits, much lighter than logs. Logs, unlike metrics, contain textual information about an event that occurred. Logs are meant to convey detailed information about the application, user, or system activity. The primary purpose of logs is troubleshooting a specific issue after the fact, e.g., code error, exception, security issue, or other. This checklist will help you select the right approach for your environment.

Use metric analytics if you:

- ☐ Need to continuously measure and get split-second insights from your cloud application code performance, business KPIs, and infrastructure metrics at high scale. The almost instant insights are essential for digital businesses generating revenue from customer-facing applications.
- ☐ Are concerned with CPU, memory, or storage consumption, in particular, when you are developing and monitoring complex distributed applications requiring benchmarking and storing large code performance data sets. As numeric measurments, metrics can be highly compressed.
- ☐ Run many microservices and containers.
- ☐ Use messaging pipelines for your application monitoring data including Kafka or others.
- ☐ Work for an organization that has many developers that need to collaborate and share metrics analysis and dashboards (such as self-service analytics for engineering teams).
- ☐ Need to apply complex processing on your code performance measurments or business KPI data such as using aggregates, histograms (distributions), and other mathematical transformations.

Use metric and log analytics if you:

- ☐ Need to process both continuous metric data events and logs. Metrics analytics helps you get the first-pane of glass across the entire application stack. Then use log monitoring to deep-dive into a specific issue to investigate the root-cause after an issue happened.
- ☐ Need proactive query-driven smart alerting.
- ☐ Implementing DevOps principles and continuous delivery of your code.
- ☐ Need to troubleshoot and deep dive into a particular system such as storage or network, after an issue occurred that generated a log.

Use log analytics if you:

- ☐ Need to analyze only unstructured text-based data from your applications and infrastructure.
- ☐ Can afford application performance data under-sampling and coarser monitoring.
- ☐ Don't need to develop and don't need to run highly distributed applications that require high scalability.
- ☐ Are developing monolithic applications that typically do not require frequent code updates requiring continuous monitoring.
- ☐ Are not concerned with slower processing of your application performance data, such as in batch-like processing.

How Should We Learn from Large-Scale Outages?

OUTAGES EXPOSE CRITICAL VULNERABILITIES

It's a time for reflection in the tech community, after huge numbers of popular and critical applications were rocked by the recent AWS S3 and Dyn DNS outages.

What can we learn from them? It's true that widely impactful outages target specific vulnerabilities—like the lack of redundancy and overdependence on AWS S3 and Dyn—but these outages also expose and publicize those same vulnerabilities.

Learn from each major outage (even if you weren't affected) and adjust your network architecture and monitoring strategies accordingly. Dealing with outages then becomes a process of incremental fortification. Your networks will be strengthened by each outage, and history won't repeat itself.

GET THE VISIBILITY TO LOCATE FAILURE POINTS

Having a full-stack network and application monitoring system in place for both internal and external services is key to learning the right lessons. To detect outages, diagnose root cause and quickly resolve issues, find a monitoring solution with:

- End-to-end visibility, from source all the way to destination
- Visibility across the network and application stacks, including web, network, routing and device layers, so you can correlate data and understand root cause
- The ability to share data with providers, team members and affected users

With the right solutions in place, you'll have a bird's-eye view of critical applications and the networks that deliver them. You'll be able to rapidly deduce the root cause of issues, keep your providers accountable with actionable data, and be equipped with the knowledge to reinforce your environment against future events.

WRITTEN BY YOUNG XU

PRODUCT MARKETING ANALYST, **THOUSANDEYES**

PARTNER SPOTLIGHT

Network & Application Monitoring By ThousandEyes

ThousandEyes

ThousandEyes delivers powerful insights for the Internet-centric enterprise by correlating application performance to network behavior.

CATEGORY
Network & Application Performance

NEW RELEASES
Bi-weekly

OPEN SOURCE
No

CASE STUDY

Zendesk is a customer service platform that around 60,000 enterprises rely on to foster better customer relationships. As a SaaS service delivered over the Internet, the perceived performance of Zendesk is heavily dependent on application performance and network quality. "We would encounter situations where our application was working well but would still hear customers report slow performance," says Steve Loyd, Vice President of Engineering Operations at Zendesk.

Zendesk uses ThousandEyes to get deep insight into application delivery that equips the operations team to react quickly to problems. Zendesk now uses ThousandEyes metrics as the ground truth to measure and share SLA metrics with their customers.

STRENGTHS

- SaaS-based solution that provides an unified view of performance from user to application
- Smart, lightweight, active monitoring probes deployed across the Internet and your network
- Pinpoint network dependencies and perform root cause analysis with intuitive visualizations
- Customizable alerts, integrations and API transform insights into actions
- Interactive snapshots shared across internal and external teams to promote collaborative problem solving

NOTABLE CUSTOMERS

- Evernote
- PayPal
- Craigslist
- Twitter
- RichRelevance
- Avera Health
- Shutterfly
- Wayfair
- Lyft

WEBSITE www.thousandeyes.com | **TWITTER** @thousandeyes | **BLOG** blog.thousandeyes.com

Understanding and Monitoring Dependencies in Cloud Applications

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

QUICK VIEW

01 Multiple regions can be used to comprise a single offering, multiple services are combined to provide another cloud product, or both. There are complex dependencies built into almost every cloud service on the market today.

02 Cloud-based services can be a cost-effective alternative to building your own, but each provider builds them differently. Have a monitoring strategy to understand the dependencies and foundational structures of these services.

03 APIs tie together cloud-based services and applications, but when APIs fail, so can everything else that relies on them. Continually test the performance of APIs and their connections for operational awareness.

It's been a nerve-wracking few months for teams managing cloud applications. In October 2016, a DDoS impaired Dyn's DNS services for hours, rendering unavailable myriad sites and services across the Internet. And in an unrelated, but similarly impactful event, the outage of AWS S3 at the end of February 2017 caused widespread and unpredictable collateral damage. With more applications leveraging more services hosted in just a few infrastructure environments, how can we make sense of application dependencies? How can we adopt a monitoring strategy that clearly accounts for the risks of improbable but hugely catastrophic service disruptions?

We'll dig into how you can identify and manage cloud dependencies by:

- Understanding underlying cloud architectures and failure scenarios
- Getting a handle on the API connections in your app and in customer interactions
- Developing a comprehensive monitoring strategy based on these requirements

MAKING SENSE OF IAAS ARCHITECTURES

Public cloud environments are a popular and powerful way to gain access to advanced services that would be costly to build or maintain on your own. But these services, from firewalls to DDoS mitigation to globe-spanning databases to data streaming platforms, are themselves composed of many other services. Like a digital matryoshka doll, it can be hard to know just how many layers and dependencies are bound up inside. In the case of the AWS S3 outage, many operations teams were surprised at how many different AWS offerings failed. They had not appreciated, and AWS had not communicated, just how interdependent various services were.

In your own data center, the failure of your entire file storage system would have a dramatic impact. In the cloud, it is the same story. The oldest services are building blocks from which other services are built (as represented in the AWS logo), and are foundational and critical to almost all other services. Basic compute (AWS EC2), storage (AWS S3) and networking (underpinning it all) are critical services that you should be monitoring and evaluating for failure scenarios. The same goes for Microsoft Azure (VMs, Blob Storage) and Google Cloud (Compute Engine, Cloud Storage). If you use cloud services that depend on these foundational elements, make sure they are part of your monitoring strategy.

Developing for the cloud also requires an understanding of failure isolation. AWS is built around the concept of regions, with previous outages typically corresponding to a single region. Unfortunately, many developers don't invest (sometimes wisely, sometimes naively) in cross-region failover strategies. So when US-East-1, the first and largest of the AWS regions has an issue, the impact is unmistakable. Some services, like Google Spanner, have different isolation mechanisms that need to be evaluated.

When it comes to architecture planning, performance monitoring and optimization, you'll want to monitor each potential failure domain. So if you are using cloud services in 4 different regions, make sure that you are collecting metrics on each.

IDENTIFYING API USAGE IN APPLICATIONS

Your applications depend on the specialized functionality of third-party applications, typically accessed via APIs. Don't think you rely on APIs for critical capabilities? Think again. APIs are very common in modern applications, hiding in plain view a complex set of dependencies. Some of these external services are important for just small portions of functionality. But many impact customer experience and revenue generation in fundamental ways.

What kind of APIs should you be monitoring? The specific APIs will be unique to your application, but some examples include:

- User authentication is accomplished with single sign-on APIs and services to detect fraud or abuse.
- Pricing and merchandising require the complex integration of many back-end applications to show an accurate price to a customer.
- Supply chain and logistics APIs ensure shipping is fulfilled.
- Payment gateways and billing systems are necessary to transact with your customers.
- Advertising is the lifeblood of many media sites and relies on APIs to display targeted products, images, descriptions, and reviews in real time.
- Customer chat, phone and CRM systems use APIs to seamlessly integrate with sites, and typically are the difference between successfully communicating with your users and being dead in the water.

There are myriad APIs that make up your overall customer experience. Getting a handle on performance dependencies requires a clear appreciation for the APIs used by your application. You enumerate your APIs in various ways: observing domains of objects on web pages, looking at connection logs from your application servers and using documentation (well hopefully it exists!) of embedded services.

Monitoring External Services, Infrastructure and APIs
Once you've figured out what to monitor, the next step to operational awareness is collecting data. There are several key elements you'll want as part of your monitoring toolkit:

1. **Log errors** of failed API connections and requests. Track trends over time to understand services that fail under your application load.
2. **Actively monitor** API servers and infrastructure services. Regularly test the reachability, response time and response codes of these services with preconfigured tests. Don't know what targets to test? Your cloud provider typically has canary servers or endpoints (here is the list for AWS) they can point you to.

Taken together, these two approaches will give you an understanding of baseline performance and specific issues as they occur. As a bonus, tying both of these methods together with a correlation engine such as Splunk can be an effective way to make sense of seemingly disparate events that are actually all related.

FOUR STEPS TO TACKLING DEPENDENCIES

Cloud-based applications, and the business models that they support, rely on an increasingly diverse set of underlying services, tied together through APIs. The availability and efficacy of APIs and infrastructure services has, therefore, become a key element in monitoring and optimizing cloud applications.

As you are building out your next cloud application or rethinking ways to meet your SLAs, follow these four steps:

1. **Map** key cloud infrastructure and application APIs. It can be a monster task but you can't optimize or mitigate services you don't know about.
2. **Test** each of the critical service dependencies with a combination of logging and active monitoring. Logs will give you forensic evidence while active monitoring will provide a heads up to impending trouble.
3. **Validate** functionality and performance with event correlation, alerting and baselining. With interdependent services, you may not know likely failure scenarios until you correlate your data.
4. **Optimize** performance over the long run to influence vendor or architectural decisions. From choosing vendors to investing in redundancy, it all starts with having clear insights from your monitoring data.

The next time a major cloud outage or service disruption hits, you'll be well aware of what is wrong, and with the proper planning, well positioned to ride out the storm.

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

what ails your APPLICATION?

Application performance issues are a lot like illnesses. There are thousands of possible culprits, and they can range from a mild annoyance to you and those around you to potentially fatal. However, like sicknesses, you can take preventative measures to minimize the occurrence and seriousness of performance problems. We surveyed 476 members of the DZone audience to learn about the most common bugs that bring them down, how difficult they are to overcome, and how proactive they are in preventing them.

HOSPITAL

APPLICATION CODE

THE FLU

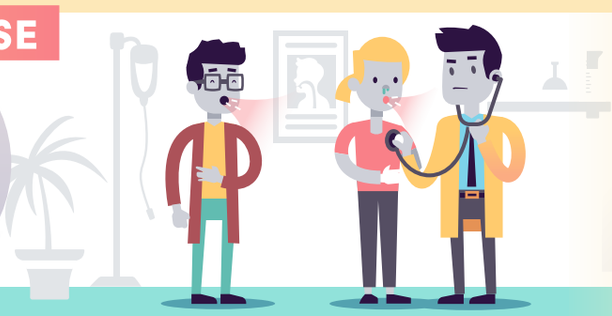
notes

36% of respondents encountered frequent performance issues with their code, while 47% had some issues.

Those using their language's built-in tooling and thread dump analyzers were 4% less likely to find such issues to be challenging than those who did not, while developers using debuggers were 5% less likely.

DATABASE

STREP THROAT



notes

Frequent database issues plagued 24% of survey respondents.

Those who build performance into their applications throughout the SDLC are 5% more likely to have no database issues than those who build application functionality first and worry about performance later.

WORKLOAD

BROKEN BONES

notes

16% of respondents encountered frequent workload issues, with 12% finding such issues to be challenging.

Developers using APM tools were 5% more likely to solve workload issues easily compared to those who do not.

MEMORY

MIGRAINES

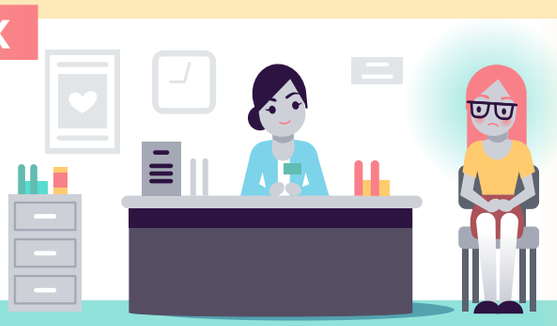
notes

15% of respondents were having problems with application memory.

Those who build performance into their applications from the start see an 8% decrease in frequent memory issues compared to those who do not.

NETWORK

COMMON COLD



notes

Frequent network issues affected 14% of survey respondents, with 10% finding such problems to be challenging.

Those who build performance into their applications from the start of development see a 5% decrease in frequent network issues compared to those who worry about performance later.

Reinventing Performance Testing

BY **ALEX PODELKO**
CONSULTING MEMBER OF TECHNICAL STAFF, **ORACLE**

QUICK VIEW

01 Cloud practically eliminated the lack of appropriate hardware as a reason for not doing load testing while also significantly decreasing the cost of large-scale tests.

02 With Agile development we've had a major "shift left" allowing us to start testing early.

03 In Agile development, performance testing should be interwoven throughout the SDLC, not an independent step.

04 Dynamic architectures provide new challenges for performance testing—more sophisticated tools may be needed.

As the industry is changing with many modern trends, performance testing should change too. A stereotypical, last-moment performance validation in a test lab using a record-playback load testing tool is no longer enough.

CLOUD

Cloud practically eliminated the lack of appropriate hardware as a reason for not doing load testing while also significantly decreasing the cost of large-scale tests. Cloud and cloud services significantly increased a number of options to configure the system under test and load generators. There are some advantages and disadvantage of each option. Depending on the specific goals and the systems to test, one deployment model may be preferred over another.

For example, to see the effect of a performance improvement (performance optimization), using an isolated lab environment may be a better option for detecting even small variations introduced by a change. For load testing the whole production environment end-to-end to make ensure the system will handle the load without any major issue, testing from the cloud or a service may be more appropriate. To create a production-like test environment without going bankrupt, moving everything to the cloud for periodical performance testing may be your best solution.

When conducting comprehensive performance testing, you'll probably need to combine several approaches. For example, you might use lab testing for performance optimization to get reproducible results and distributed, realistic outside testing to check real-life issues you can't simulate in the lab.

AGILE

Agile development eliminates the primary problem with traditional development: you need to have a working system before you may test it. Now, with agile development, we've had a major "shift left", allowing us to start testing early.

Theoretically, it should be rather straightforward—every iteration you have a working system and know exactly where you stand with the system's performance. From the agile development side, the problem is that, unfortunately, it doesn't always work this way in practice. So, such notions as "hardening iterations" and "technical debt" get introduced. From the performance testing side, the problem is that if we need to test the product each iteration or build, the volume of work skyrockets.

Recommended remedies usually involve automation and making performance everyone's job. Automation here means not only using tools (in performance testing, we almost always use tools), but automating the whole process including setting up the environment, running tests, and reporting/analyzing results. Historically, performance test automation was almost non-existent as it's much more difficult than functional testing automation, for example. Setups are more complicated, results are complex (not just pass/fail) and not easily comparable, and changing interfaces is a major challenge—especially when recording is used to create scripts.

While automation will take a significant role in the future, it only addresses one side of the challenge. Another side of the agile challenge is usually left unmentioned. The blessing of agile development, early testing, requires another mindset and another set of skills and tools. Performance testing of new systems is agile and exploratory in itself. Automation, together with further involvement of development, offloads performance engineers from routine tasks. But, testing early—the biggest

benefit being that it identifies problems early when the cost of fixing them is low—does require research and analysis; it is not a routine activity and can't be easily formalized.

CONTINUOUS INTEGRATION

Performance testing shouldn't just be an independent step of the software development life-cycle where testers get the system shortly before release. In agile development/DevOps environments, it should be interwoven with the whole development process. There are no easy answers here to fit every situation. While agile development/DevOps is becoming more and more mainstream, their integration with performance testing is just making its first steps.

What makes agile projects really different is the need to run a large number of tests repeatedly, resulting in the need for tools to support performance testing automation. The situation started to change recently as agile support became the main theme in load testing tools. Several tools recently announced integration with Continuous Integration Servers (such as Jenkins and Hudson). While initial integration may be minimal, it is definitely an important step toward real automation support.

It doesn't look like we'll have standard solutions here, as agile and DevOps approaches differ significantly and proper integration of performance testing can't be done without considering such factors as development and deployment processes, system, workload, and the ability to automate gathering and the analysis of results.

NEW ARCHITECTURES

Cloud seriously impacts system architectures, having a lot of performance-related consequences.

First, we have a shift to centrally managed systems. 'Software as a Service' (SaaS) are basically centrally managed systems with multiple tenants/instances.

Second, to get the full advantage of cloud, such cloud-specific features as auto-scaling should be implemented. Auto-scaling is often presented as a panacea for performance problems, but, even if it is properly implemented, it just assigns a price tag for performance. It will allocate resources automatically, but you need to pay for them. Any performance improvement results in immediate savings.

Another major trend involves using multiple third-party components and services, which may be not easy to properly incorporate into testing. The answer to this challenge is service virtualization, which allows one to simulate real services during testing without actual access.

Cloud and virtualization triggered the appearance of dynamic, auto-scaling architectures, which significantly impact collecting and analyzing feedback. With dynamic architectures, we have a great challenge ahead of us: to discover configuration automatically, collect all necessary information, and then properly map the collected information and results to a changing configuration in a way that highlights existing and potential issues—and potentially, to make automatic adjustments to avoid them. This would require very sophisticated algorithms and sophisticated Application Performance Management systems.

NEW TECHNOLOGIES

New technologies may require other ways to generate load. Quite often, the whole area of load testing is reduced to pre-production testing using protocol-level recording/playback. Sometimes, it even leads to conclusions like "performance testing hitting the wall" just because load generation may be a challenge. While protocol-level recording/playback was (and still is) the mainstream approach to testing applications, it is definitely just one type of load testing using only one type of load generation; such equivalency is a serious conceptual mistake, dwarfing load testing and undermining performance engineering in general.

Protocol-level recording/playback is the mainstream approach to load testing: recording communication between two tiers of the system and playing back the automatically created script (usually, of course, after proper correlation and parameterization). As far as no client-side activities are involved, it allows the simulation of a large number of users. But, such a tool can only be used if it supports the specific protocol used for communication between two tiers of the system. If it doesn't or it is too complicated, other approaches can be used.

UI-level recording/playback has been available for a long time, but it is much more viable now. New UI-level tools for browsers, such as Selenium, have extended the possibilities of the UI-level approach, allowing the running of multiple browsers per machine (limiting scalability only to the resources available to run browsers). Moreover, UI-less browsers, such as HtmlUnit or PhantomJS, require significantly fewer resources than real browsers.

Programming is another option when recording can't be used at all, or when it can, but with great difficulty. In such cases, API calls from the script may be an option. Often, this is the only option for component performance testing. Other variations of this approach are web services scripting or the use of unit testing scripts for load testing. And, of course, there is a need to sequence and parameterize your API calls to represent a meaningful workload. The script is created in whatever way is appropriate and then either a test harness is created or a load testing tool is used to execute scripts, coordinate their executions, and report and analyze results.

SUMMARY

Performance testing should reinvent itself to become a flexible, context-, and business-driven discipline. It is not that we just need to find a new recipe; now, we need to be able to adjust on the fly to every specific situation in order to remain relevant.

ALEX PODELKO has specialized in performance since 1997, working as a performance engineer and architect for several companies. Currently he is Consulting Member of Technical Staff at Oracle, responsible for performance testing and optimization of Enterprise Performance Management and Business Intelligence (a.k.a. Hyperion) products. Alex periodically talks and writes about performance-related topics, advocating tearing down silo walls between different groups of performance professionals. His collection of performance-related links and documents (including his recent papers and presentations) can be found at alexanderpodelko.com. He blogs at alexanderpodelko.com/blog and can be found on Twitter as @apodelko. Alex currently serves as a director for the Computer Measurement Group (CMG, cmg.org), an organization of performance and capacity planning professionals.

Executive Insights Performance Optimization and Monitoring

BY **TOM SMITH**
RESEARCH ANALYST, **DZONE**

QUICK VIEW

01 Use real-time user monitoring to provide visibility into the entire pipeline to ensure an optimal user experience with video, applications, and web pages.

02 While there's a proliferation of tools providing visibility across networks, architectures, and devices, no one has developed a single, holistic solution.

03 In the future, there will be a single, holistic solution that uses machine learning to solve problems before they even occur for an optimal user experience.

To gather insights on the state of performance optimization and monitoring today, we spoke to 12 executives from 11 companies that provide performance optimization and monitoring solutions for their clients Here's who we spoke to:

JOSH GRAY, Chief Architect, Cedexis

JEFF BISHOP, General Manager, ConnectWise Control

BRYAN JENKS, CEO and Co-Founder, DropLit.io

DORU PARASCHIV, Co-Founder, IRON Sheep TECH

YOAV LANDMAN, Co-Founder and CTO, JFrog

JIM FREY, V.P. Strategic Alliances, Kentik

ERIC SIGLER, Head of DevOps, PagerDuty

NICK KEPHART, Senior Director Product Marketing, ThousandEyes

KUNAL AGARWAL, CEO, Unravel Data

LEN ROSENTHAL, CMO, Virtual Instruments

ALEX RYSENKO, Lead Software Engineer, Waverly Software

EUGENE ABRAMCHUK, Sr. Performance Engineer, Waverly Software

Here are the key findings from the subjects we covered:

01 The keys to performance optimization and monitoring are **the design infrastructure and real-time user monitoring (RUM)** to ensure an optimal end-user experience (UX) whether it's videos, web pages, or applications. The proliferation of new services, requirements, and devices in diverse geographic locations has made visibility into the entire network critical. You need to be able to see where all of your data is residing to understand how performance is, or is not, being optimized.

02 There's a greater **need for visibility, and there's a proliferation of tools** coming online to provide that visibility. However, no one has developed a single solution to provide a complete view across a diverse collection of infrastructures and application architectures. Response times and page-load times have continued to decrease with the adoption of virtualization and microservices. We're evolving from performance monitoring to performance intelligence with the addition of easy-to-understand, contextually relevant, algorithmically-driven performance analytics. However, it's important to identify and focus on key business metrics, or else you run the risk of being overwhelmed with data.

03 The most frequently mentioned performance and monitoring tools used are **AppDynamics, New Relic, and DataDog**. However, these were just three of more than 30 mentioned, with a trend towards more granular and specialized offerings, and respondents mentioning just a few solutions that came to mind besides their own.

04 Real-world problems that are being solved with performance optimization and monitoring are **time to market, optimization of UX, and reduction in time to**

resolve issues through greater collaboration among teams. While more tools are coming online, some providers are enabling disparate tools to provide an integrated view to the client, which results in greater visibility into the entire pipeline and faster time to problem resolution. This visibility is also enabling clients to ensure service level agreements (SLAs) are being met by third-party providers.

05 Nonetheless, the most common issues continue to be the need to **improve visibility, ease of use, performance, and knowledge of the impact that code has on the UX**. Incomplete visibility throughout the pipeline prevents organizations from accurately finding the source of latency in the network, the application, or the endpoint. There continues to be a lack of knowledgeable professionals that know distributed computing and parallel processing. As such, technical complexity of these tools must be reduced for companies to get the most value from them. Vendors should also improve the ease of use through analytics so IT operations do less data interpretation and can focus more on remediation. Understanding the product, load, load tests, and performance graphs is critical. Several developers do not understand the performance impact of their code and they are not pre-optimizing their code, which can lead to less readable code with more complex bugs. Ensure that you talk to end users in order to understand what they are experiencing and what's important to them. Do not assume you know what they want.

06 The biggest opportunities for improvement are the **automatic reaction to, and correction of, issues and having more elegant, thoughtful design, and testing** resulting in an optimal UX. In the future, performance and monitoring tools will automatically react to issues and know the difference between mitigating and fixing problems. They'll be able to do this by collecting more data and identifying a dynamic system to determine what the problem may be before it affects the customer. Data will be more manageable with automated analysis. Application design will feature higher level programming, better tools, and graceful degradation. Just as data is used to solve problems, it can also be used to change the way performance testing is done and measured. All monitoring products will monitor across the hybrid data center, including on-premise and public cloud-deployed applications.

07 The biggest concerns about performance and monitoring today are the **lack of collaboration, identification of KPIs and how to measure them, and expertise**. Companies are not moving quickly enough to share and integrate different viewpoints. Smaller teams can implement more iterative solutions more quickly, which allows them learn faster and observe how small optimization differences can have massive hardware implications. It's important to identify and agree upon KPIs for each business unit, and how they will be measured. Premature optimization is a common pitfall in software development. It's common to see software being developed without concern for consistency or use cases, which dramatically affect the quality and speed of the software.

08 The skills needed by developers to optimize application performance and monitoring are: **1) understanding of the fundamentals; 2) understanding the concept of benchmarking and improving; and 3) staying creative**. Have authoritative understanding of the underlying IT infrastructure and the expertise to keep it running in the face of constant change, independent of vendors or location. Understand the architecture of the system, how services talk to each other, how the database is accessed, and how messages are read by concurrent consumers. Keep a broad perspective, an open mind, and an understanding of the needs and wants of the end user. Don't assume the model you have in your mind is correct and know you're going to get it wrong. Get used to designing in a way that makes it easy to make a few small changes than having to rebuild the entire application. Set a reliable benchmark for the performance goals that are relevant to your business application and work to improve on those goals as you get more information.

09 An additional consideration made by a few of our participants is the question of **where performance monitoring begins and ends versus testing and validation**. Once a problem is identified and remediation proposed, there is a need to test and validate that the change has completely fixed the problem. What effect will advancements in technologies such as AI, bots, BI, data analytics, ElasticSearch, natural language search, and new open source frameworks with standardized APIs have on performance and monitoring?

Let us know if you agree with their perspective or have answers to the questions they raised. We'd love to get your feedback.

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

Real World Performance and the Future of JavaScript Benchmarking

BY **BENEDIKT MEURER**

TECH LEAD OF THE JAVASCRIPT EXECUTION OPTIMIZATION TEAM, **GOOGLE**

QUICK VIEW

01 Web workloads are changing, performance metrics and tooling need to be adapted appropriately.

02 JavaScript engines are focusing on broadening the fast path beyond just peak scripting performance.

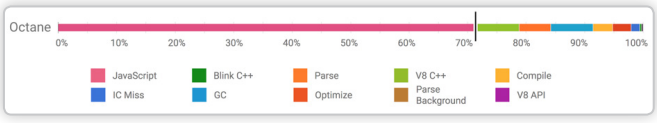
03 Whenever possible, modern JavaScript should be shipped to the browser to avoid the transpiler overhead.

04 Limiting the amount of JavaScript proportionally to what's visible on the screen is a good strategy.

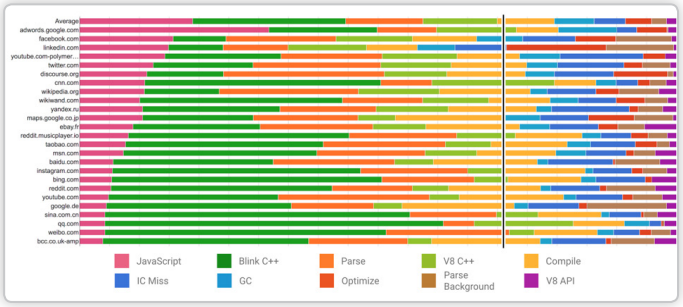
In the last 10 years, an incredible amount of resources went into speeding up peak performance of JavaScript engines. This was mostly driven by peak performance benchmarks like SunSpider and Octane, and shifted a lot of focus toward the sophisticated optimizing compilers found in modern JavaScript engines like Crankshaft in Chrome.

This drove JavaScript peak performance to incredible heights in the last two years, but at the same time, we neglected other aspects of performance like page load time, and we noticed that it became ever more difficult for developers to stay on the fine line of great performance. In addition to that, despite all of these resources dedicated to performance, the user experience on the web seemed to get worse over time—especially page load time on low-end devices.

This was a strong indicator that our benchmarks were no longer a reasonable proxy for the modern web, but rather turned into a caricature of reality. Looking at Google's Octane benchmark we see that it spends over 70% of the overall execution time running JavaScript code.

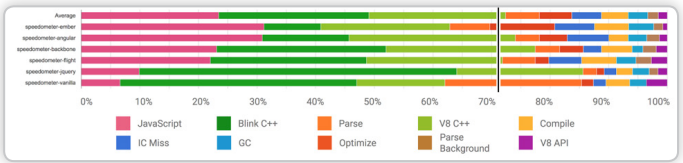


Comparing this to profiles we see during startup of some 25 top web pages, we see that those are nowhere near the 70% JavaScript execution of Octane. They obviously spend a lot of time in Blink doing layouting and rendering, but also spend a significant amount of time in parsing and compiling JavaScript.



On average, the time spent in executing JavaScript is roughly 20%, but more than 40% of the time is spent in just parsing, IC (inline cache) Miss and V8 C++ (the latter of which represent the subsystems necessary to support the actual JavaScript execution, and the slow paths for certain operations that are not optimized in V8). Optimizing for Octane might not provide a lot of benefit for the web. In fact, parsing and compiling large chunks of JavaScript is one of the main problems for startup of many web pages nowadays, and Octane is a really bad proxy for that.

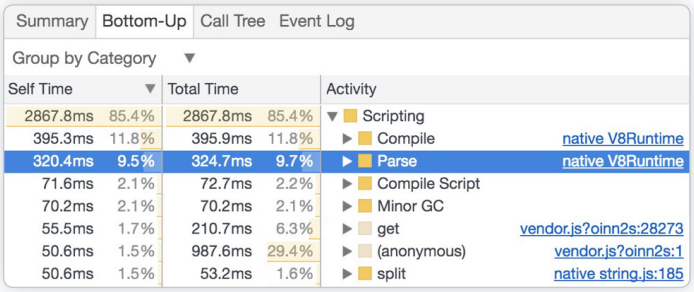
There's another benchmark suite named Speedometer, that was created by Apple in 2014, which shows a profile that is closer to what actual web pages look like. The benchmark consists of the popular TodoMVC application implemented in various web frameworks (i.e. React, Ember, and AngularJS).



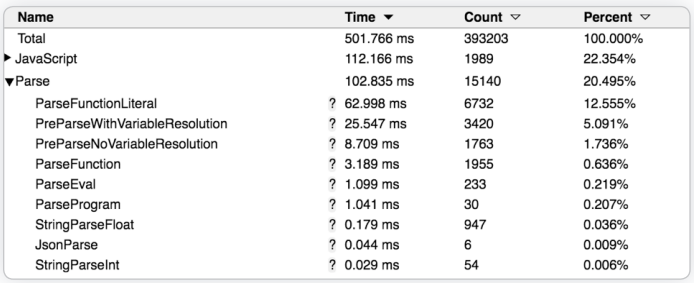
As shown in the profile, the Speedometer benchmark is already a lot closer to what actual web page profiles look

like, yet it's still not perfect - it doesn't take into account parse time for the score, and it creates 100 todos within a few milliseconds, which is not how a user interacts with a web page usually. V8's strategy for measuring performance improvements and identifying bottlenecks thus changed from using mostly traditional JavaScript benchmark methods toward using browser benchmarks like Speedometer and also tracking real-world performance of web pages.

What's interesting to developers in light of these findings is that the traditional way of deciding whether to use a certain language feature by putting it into some kind of benchmark and running it locally or via some system like jsperf.com might not be ideal for measuring real-world performance. When following this route, it's possible for the developer to fall into the microbenchmark trap and observe mostly the raw JavaScript execution speedup, without seeing the real overhead cumulated by the other subsystems of the JavaScript engine (i.e. parsing, inline caches, slow paths triggered by other parts of the application, etc.) that negatively affect a web page's performance. At Chrome, we have been making a lot of the tooling that supported our findings available to developers via the Chrome Developer Tools.



You can now see parsing and compile buckets in the profiler. And, over the last few years, we've introduced another mechanism - called chrome://tracing - which allows you to record traces that collect all kinds of events. For example, you can analyze in detail how much time V8 spends in the different parsing steps, and thereby understand whether it might make sense to consider using a tool like optimize-js to mitigate the overhead of pre-parsing when it's not beneficial, for example the function is executed immediately anyway.



Chrome Tracing provides you with a pretty detailed understanding of what's going on performance-wise by offering a view into the less obvious places. V8 has a step-by-step guide on how to use this. For most use cases though, I'd recommend sticking to the Developer Tools, because they offer a more familiar interface and don't expose an overwhelming amount of the Chrome / V8 internals. But for advanced developers, chrome://tracing might be the swiss army knife that they were looking for.

Looking at the web today, we've discovered that it is important to significantly reduce the amount of JavaScript that is shipped to the browser, as we live in a world where more and more users consume the web via mobile devices that are a lot less powerful than a desktop computer and might not even have 3G connectivity.

One key observation is that most web developers use ECMAScript 2015 or later for their daily coding already, but for backwards compatibility compile all their programs to traditional ECMAScript 5 with so-called transpilers, like Babel, for example. This can have unexpected impact on the performance of your application because often the transpilers are not tuned to generate high performance code. Thus the final code that is shipped might be less efficient than the original code. But there's also the increase in code size due to transpilers: The generated code is usually **200-1000%** the size of the original code, which means the JavaScript engine has up to 10 times the work, in parsing, compiling, and executing your code.

Since not all browsers support all new language features, there's a certain period of time where new features require transpilation. But if you are building a web application today, consider shipping as much of the original code as possible. An Intranet application with dedicated clients inside the company, all using some recent browser version, could as well be written and shipped as ES2015 code.

A good rule of thumb currently, is to ship amounts of JavaScript proportionally to what's on the screen. Think about code splitting from the beginning and design your web application with progressive enhancement in mind whenever possible. And independent of what kind of application you are developing, try to be as declarative as possible, using appropriate algorithms and data structures; i.e. if you need a map, use a Map. If it turns out to be slow in a certain browser, file a bug report. Focus optimization work on bottlenecks identified via profiling.

BENEDIKT MEURER joined Google in 2013 to work on the V8 JavaScript VM that powers both Node.js and Chrome. He is the tech lead of the JavaScript Execution Optimization team, focusing on the compiler architecture and performance of new language features. He contributed to various open source projects in the past, including OCaml, Xfce, and NetBSD. In his spare time, he's a father of two, enjoys hiking and biking.