Leveraging the Crowd: How 48,000 Users Helped Improve Lync Performance

Robert Musson and Jacqueline Richards, Microsoft

Danyel Fisher and Christian Bird, Microsoft Research

Brian Bussone and Sandipan Ganguly, Microsoft

// A new analysis approach produces visualizations to help development teams identify and prioritize performance issues by focusing on performance early in the development cycle, evaluating progress, identifying defects, and estimating timelines. //

REAL-WORLD PERFORMANCE IS an aspect of software quality that historically has been difficult to measure. Software developers have devoted enormous amounts of time and effort to effectively predict how well a piece of software will perform under various real-world conditions. It's especially difficult to evaluate performance for applications that rely on human communication and network operations for the majority of their functionality.

As mobile devices become more prevalent and Web services and applications grow in market share, information flow across networks and the Internet is becoming an increasingly important piece of most applications. However, network environments are often heterogeneous, and their latency and bandwidth can vary wildly depending on factors such as the physical link used (wired versus Wi-Fi), routing hardware, protocols employed, distance between endpoints, firewall rules, and network saturation. In each of these conditions, different use cases—for example, large group meetings or a two-person video chat—can have radically different performance characteristics. Despite this, users expect applications to perform well regardless of environment. With the primary goal of performance monitoring and improvement being high levels of customer satisfaction, how can software project stakeholders evaluate the performance of network-reliant applications in a way that reflects diverse, real-world use?

Rather than improve methods of simulating various operations, scenarios, and environments as testing has traditionally done,[1] we can deploy software in a controlled way to groups of users during development and collect performance data. We then dynamically instrument the code to inspect scenarios of interest. Compared to laboratory testing or simulation, the resulting data is both more diverse and more representative because it comes from real use and therefore represents customers' actual experiences. Once this data has been collected, we can break it down into constituent dimensions—by usage scenario, location, and machine configuration—and present the results in ways that can help project stakeholders make decisions. Finally, an analysis dashboard, Engineering Intelligence Analytics (EI Analytics) lets developers investigate performance data.

We've carried out this type of work with several teams, and in the case study presented here, we describe our work with the team responsible for Lync, Microsoft's enterprise

0740-7459/13/$31.00 © 2013 IEEE

communication tool. (For more information, see the "Lync" sidebar.) Specifically, we cover our techniques and experiences in using live data, combined with interactive surveys, to analyze performance. Our performance-monitoring approach has been successfully deployed, has improved development decisions, and is continuously in use with a large-scale enterprise-level software service.

LYNC

Lync is a large-scale enterprise communication system. In its current version, Lync supports a vast number of features from awareness of user presence to large-scale meetings. It also supports IP telephony, video-conference calling, and whiteboard sharing, and integrates with address books, calendars, email, and even word processors. As such, the Lync development team is large and contains many different members, with different subsets of the team responsible for different features and aspects of the tool.

Background

We start with a brief description of Lync itself, then discuss real-world performance and provide examples of the types of questions that project stakeholders often ask.

The Lync Application

One major issue for an application such as Lync is maintaining acceptable levels of responsiveness across its many features—for instance, even if a text messaging feature is quick, if looking up a name is slow, then users will feel dissatisfied. Thus, as new features are developed and tested, the development team might need to modify what data collection occurs at a low cost.

Lync has recently moved toward a rapid release cycle. To accommodate this change, versions must be compatible with each other, so there's a tendency to mutate previous versions rather than start from scratch. In this new model, the available time for the stabilization phase of development decreases. Rather than reaching feature completion and then focusing on performance, Lync examines the performance of each scenario during the entire development cycle.

Finding Performance in the Real World

The ultimate measure of application performance is whether users are happy with the application's responsiveness. This is subjective and difficult to directly measure (see the "Related Work in Performance Testing" sidebar). Past versions of Lync periodically solicited performance testers' judgment—they would ask users if a particular scenario "feels okay" or "isn't slow." However, because this information wasn't connected to logged outcomes, it remained difficult to assess performance over an application's development.

The Lync user-experience team has long set target specification values as goals for performance—for example, "A video conference should connect within 500 milliseconds." However, these would often be measured only in laboratory testing.

In this article, *performance* refers only to the wall-clock time of a given network-bound operation. We divide the user experience into a series of *scenarios*—each is a discrete operation or set of operations that doesn't require user intervention.

Testing performance in a laboratory is both costly and inaccurate relative to real-world use. On one hand, test matrices must be written to cover each possible combination of external factors and scenarios, and testers must walk through multiple scenarios to measure them. It can be difficult for a laboratory to simulate the possible external conditions that users routinely experience. For example, it's hard to imagine a test matrix that would cover one colleague working in Africa calling another in Europe, initiating screen-sharing during the conversation, and adding in a third colleague one office over.

Some of these externalities can be simulated in vitro by randomly dropping packets, introducing delay, or misrouting traffic, but the sheer amount of variation in real-world environments is nearly impossible to replicate. Our solution is to embrace and operate directly in these environments rather than try to reproduce them.

Analytics Questions

With the many different features in Lync, project stakeholders are beginning to ask nuanced questions about performance to understand how aspects of application usage affect it:

- What is the relative impact on performance if everyone uses Lync for four-person meetings, eight-person meetings, and all-hands meetings?
- What is the performance difference of four people talking versus one person presenting while three are watching?
- Are there differences in performance by geographic region, distance to servers, or time of day?
- Does server type affect performance?

Answers to these questions can help project members make decisions

RELATED WORK IN PERFORMANCE TESTING

Performance analysis is a complex subject with a long history (for example, see Henry Lucas Jr.'s survey on performance monitoring and evaluation from 1971[1]). *Performance* can take on multiple meanings, from disk speed to graphics rendering. With the rise of networked systems, performance analysis becomes more urgent,[2] particularly in modern client-server scenarios. In these clients, performance degradation can come from client-side issues, such as network connections, and server-side issues, such as server load and the time to service requests. Common approaches to discover performance issues include modeling[3] and creating synthetic workloads,[4] sometimes based on past user data.[2] Our system is different because we are able to deploy incremental versions of the system to a broad set of users.

Dieter Haban and Dieter Wybranietz's work is more similar to ours.[5] It comprises an event-driven system for monitoring distributed applications in situ that collects performance and behavioral data, as well as Simple, a tool environment for performance evaluation and modeling that includes multiple visualizations.

The Paradyn suite of tools is also similar to our approach in functionality because it doesn't require manual code modification when the target areas for performance analysis change. It automatically instruments code at runtime by modifying the binary to report function calls and memory accesses.[6] Our approach leverages event hooks as part of the core functionality of the application—that is, it doesn't require additional code or code modifications.

References

1. H. Lucas Jr., "Performance Evaluation and Monitoring," *ACM Computing Surveys,* vol. 3, no. 3, 1971, pp. 79–91.
2. M.F. Arlitt and C.L. Williamson, "Internet Web Servers: Workload Characterization and Performance Implications," *IEEE/ACM Trans. Networking*, vol. 5, no. 5, 1997, pp. 631–645.
3. S. Balsamo et al., "Model-Based Performance Prediction in Software Development: A Survey," *IEEE Trans. Software Eng.*, vol. 30, no. 5, 2004, pp. 295–310.
4. A. Avritzer and E. Weyuker, "The Automatic Generation of Load Test Suites and the Assessment of the Resulting Software," *IEEE Trans. Software Eng.*, vol. 21, no. 9, 1995, pp. 705–716.
5. D. Haban and D. Wybranietz, "A Hybrid Monitor for Behavior and Performance Analysis of Distributed Systems," *IEEE Trans. Software Eng.*, vol. 16, no. 2, 1990, pp. 197–211.
6. B.P. Miller et al., "The Paradyn Parallel Performance Measurement Tool," *Computer,* vol. 28, no. 11, 1995, pp. 37–46.

about parts of the system to work on, where to focus future development, and what support teams might expect after release.

Approach

The goal of our approach is to obtain and analyze data that comes from actual use of Lync and to obtain this as early in the development process as possible. We describe here the relevant details that let us achieve these dual goals.

Early Deployment

Our approach requires that ordinary users operate in-development versions of Lync during the normal course of their work. An in-house program lets users from across Microsoft subscribe to prerelease versions of the software (known internally as "dogfood"); these opportunities are advertised by email notifications to mailing lists, promotional material on the corporate intranet, and physical media such as posters in the workplace. Any user that would like to help—or would like early access to advanced features of upcoming releases—can subscribe to development versions. These versions of Lync help us with performance reporting and also have voting buttons that users can press to indicate satisfaction (or annoyance) with a given feature. The dogfood versions have passed through basic testing rounds but aren't considered to be release-quality code; new builds are released as often as weekly.

Data Collection

Lync contains a subsystem for collecting and transmitting performance data. Over the course of the development cycle, the team often needs to adjust the data it collects with little impact to the user and minimal work for the development team.

Prior Microsoft systems relied on teams adding instrumentation code to their applications. Because instrumentation was a low priority compared to shipping features, the instrumentation code would often be low priority and low quality.

One alternative to this approach is to build a system ready to be fully instrumented. As in many modern network-based systems, Lync is built around an event-driven API. This API creates Windows events when any operating system–level operation occurs, from user interaction (UI) to socket communication. When the development team wishes to collect data regarding a specific scenario, they first identify the events that begin and end that scenario (similar to defining pointcuts in aspect-oriented programming[2]). The scenario

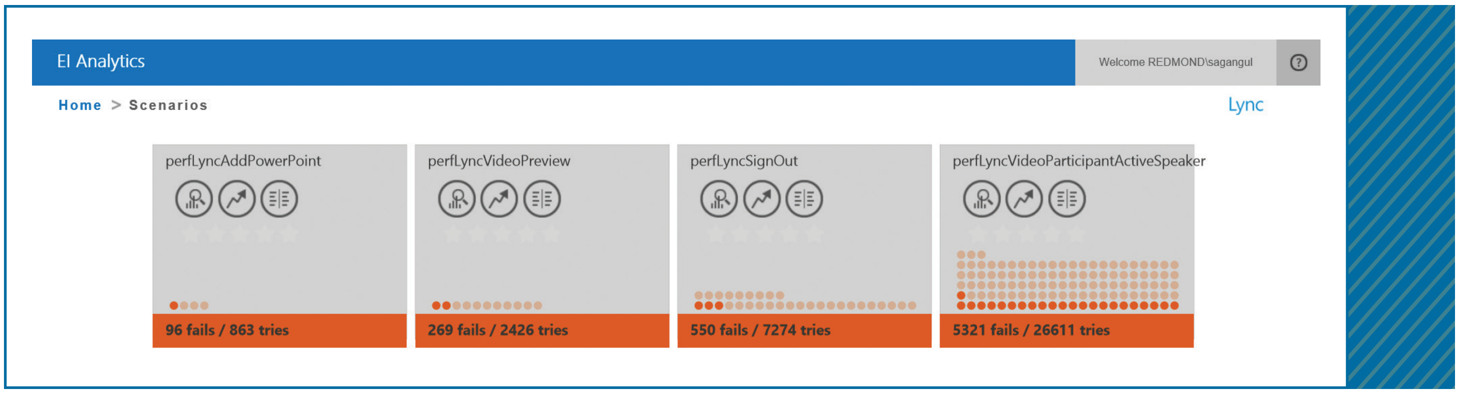

FIGURE 1. A sample of high-level view results for four scenarios in Lync. In each cell, the color reflects the overall prognosis (gray is untested; red and green show performance levels). The fails summary provides the number of attempts that didn't succeed; the other buttons lead to various visualizations and detailed reports on the scenario.

might begin when the user clicks a UI element and conclude when a particular I/O request completes. The team specifies the scenario and the names of its constituent events and adds this scenario to a large scenario table stored on a server. A performance-monitoring subsystem on the client periodically checks for any changes to the scenario table, records the timestamps of these events, and transmits them to the performance database server within Microsoft. Because the act of transmitting collected data can affect performance, the development team provides rules for when and how often the data should be transmitted.

Additional information about the usage context that could help testers is also collected, such as

- build numbers and versions,
- machine architecture (x86, x64, ARM),
- main and video memory available,
- network protocols, and
- connections to the server's geographic region.

Although we primarily gathered data from employees within Microsoft, privacy is still an important concern—that is, although some companies don't treat their employees' work activities as private within the company, Microsoft does. We collect identifying data that we believe could affect performance such as geographical region, but we don't transmit or record any personally identifiable information such as username or content of messages to our database.

For Lync, we have defined approximately 350 scenarios over the course of development. More than 48,000 users have participated in the dogfood program, and have thus used at least one such prerelease version of Lync.

Visualization and Analytics

The Lync development team can explore the data results through a reporting website, EI Analytics. The application lets users examine high-level scenario performance results and, when desired, explore the data at a finer granularity in more detail

Figure 1 depicts a snippet of Lync's entry page, which contains a high-level view of three scenarios' statuses (the full page contains many more scenarios). Each scenario is depicted by an informational box showing its name, how its performance compares to expected levels, and the frequency of success. Figure 2 shows how EI Analytics presents performance data for one service scenario. The histogram is generated from the timings (or durations) for all users running that scenario. This particular case identifies three distinct user experience groups, represented by three Gaussian curves. The Gaussians are generated by decomposing the durations' probability distribution functions. The dotted line shows the Gaussians' actual fitted linear combination. The rightmost Gaussian denotes the worst user experience group in terms of performance and warrants engineer attention.

EI Analytics also lets users compare datasets for differences between conditions. Users can select different filters from a menu. For example, a developer might be concerned that the 64-bit version of an application is behaving differently than the 32-bit version. In Figure 3, one scenario is split across uses in these two versions of the Lync client. The left side of the figure shows event timing distributions on top of each other, and the right side displays a more compact view of the data in boxplot form. The target specification value for the scenario is depicted with a bold vertical black line.

One of our goals with this project is to monitor performance over time during development. Typically, builds go through a rhythm—features are added,

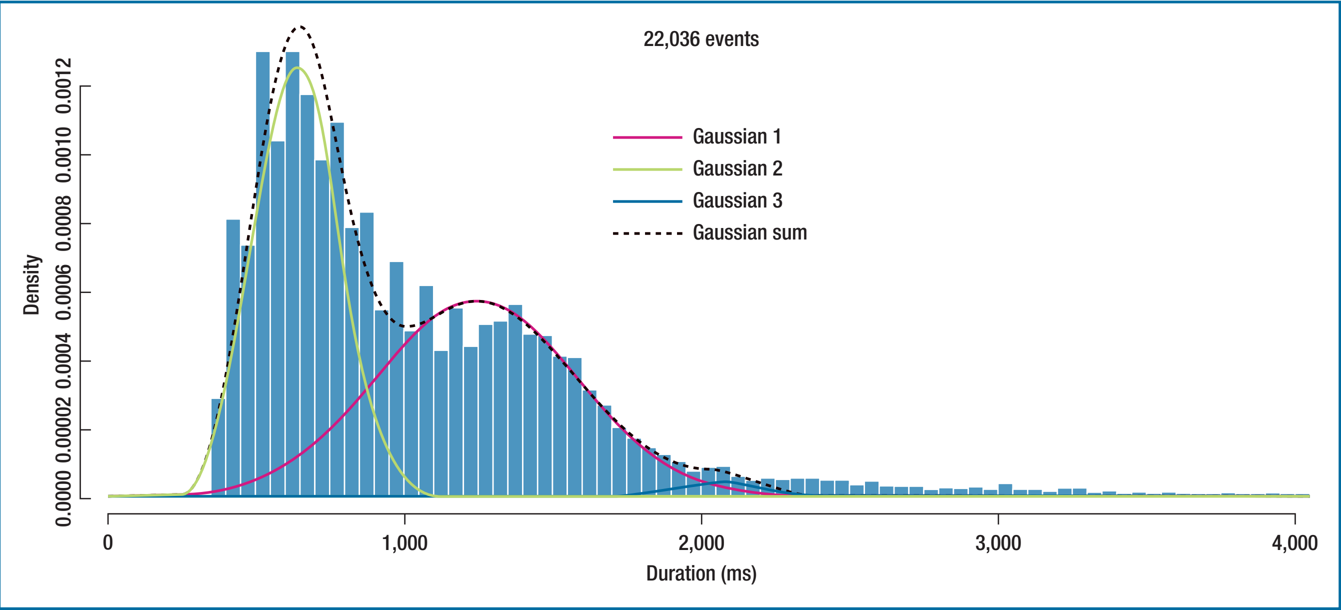

FIGURE 2. A histogram of the durations for a single scenario for a given build. The durations are fit to the sum of Gaussians, which each represent a different user experience.

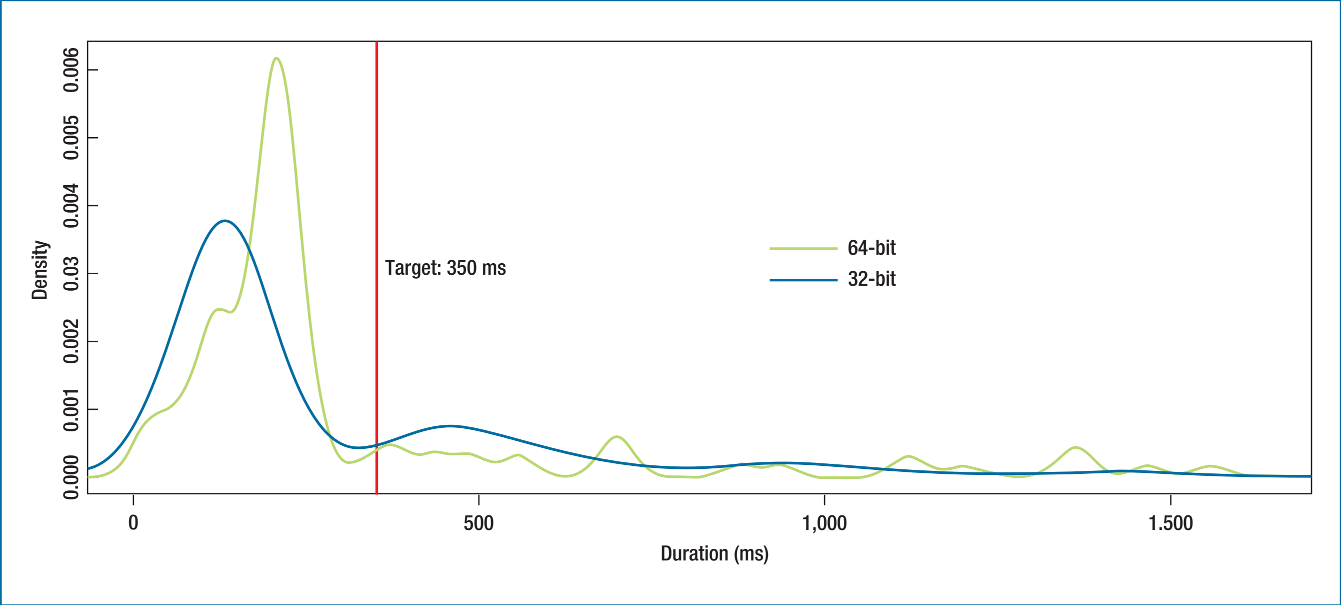

FIGURE 3. Empirical distributions for a scenario for a given build drawn from users using 32-bit (blue) and 64-bit (green) versions of Lync. The black line indicates the target specification. More compact boxplots of the data are shown on the right.

and then developers focus on performance. We therefore provide a time trend analysis for each scenario, depicting a variable width notched boxplot showing the performance distribution for a scenario for each build along a calendar time line. Figure 4 shows one such build-over-build comparison.

The Lync team is also interested in international performance. User experience quality can be affected by distance from servers, local network conditions, and international firewalls and barriers. The world performance map (see Figure 5) indicates whether performance is suffering internationally and supports decisions about investments in network and server infrastructure.

Impact

These techniques are only of value if their use positively impacts software development. Ultimately, this impact would result in improved customer satisfaction, but we're too early in release to see results on that outcome. Nonetheless, we do have evidence that EI Analytics is having a positive impact on the development process and decision making.

Organizational Change and Adoption

The availability of EI Analytics data is driving an organizational change within the Lync team. Historically, the team reserved performance testing for late in the deployment process because it was difficult to get performance data in an easily consumable fashion for the entire development process. Thus, most of the focus on performance occurred in the final weeks before release. This is changing as a result of our techniques.

We're using a phased approach as we roll out EI Analytics to the Lync development team. In the initial phase, which is where we are right now, it's being used in Lync "ship room," a weekly meeting the project managers use to make decisions about shipping dates and feature cuts. Scenario performance provides an indication of whether a given feature needs additional work, should be considered for removal, or is ready to ship.

Some teams have begun to incorporate EI Analytics into their regular use outside ship room meetings. We're working on the next phase of roll out, which will comprise incorporating the approach's use in feature lead developers' daily routines. By continuously monitoring performance during development, teams can catch issues before they reach the ship room.

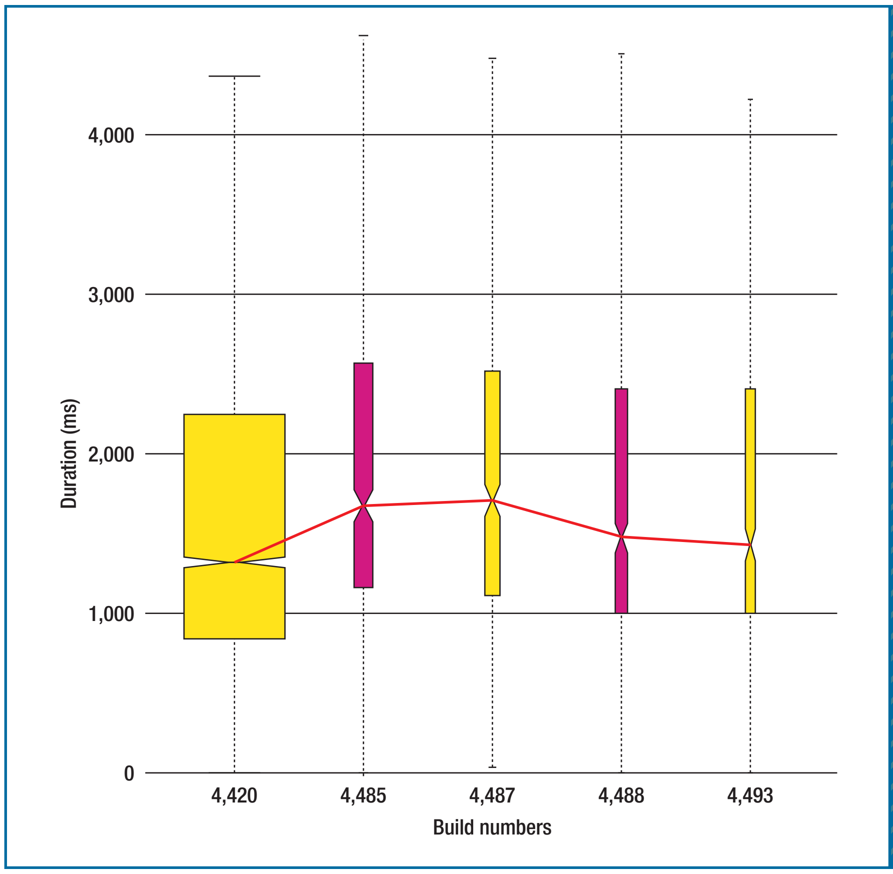

FIGURE 4. A build-over-build analysis. Red indicates a statistically significant difference from the previous build. Width indicates the number of users who used this scenario for this build. Height shows the interquartile range.

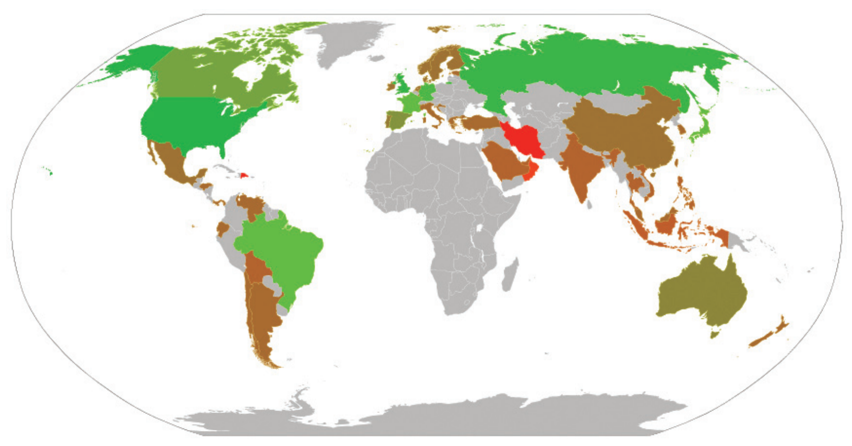

FIGURE 5. An example (not real, for confidentiality reasons) of a world performance map for a single scenario and build. This map indicates how the global user population experiences the application.

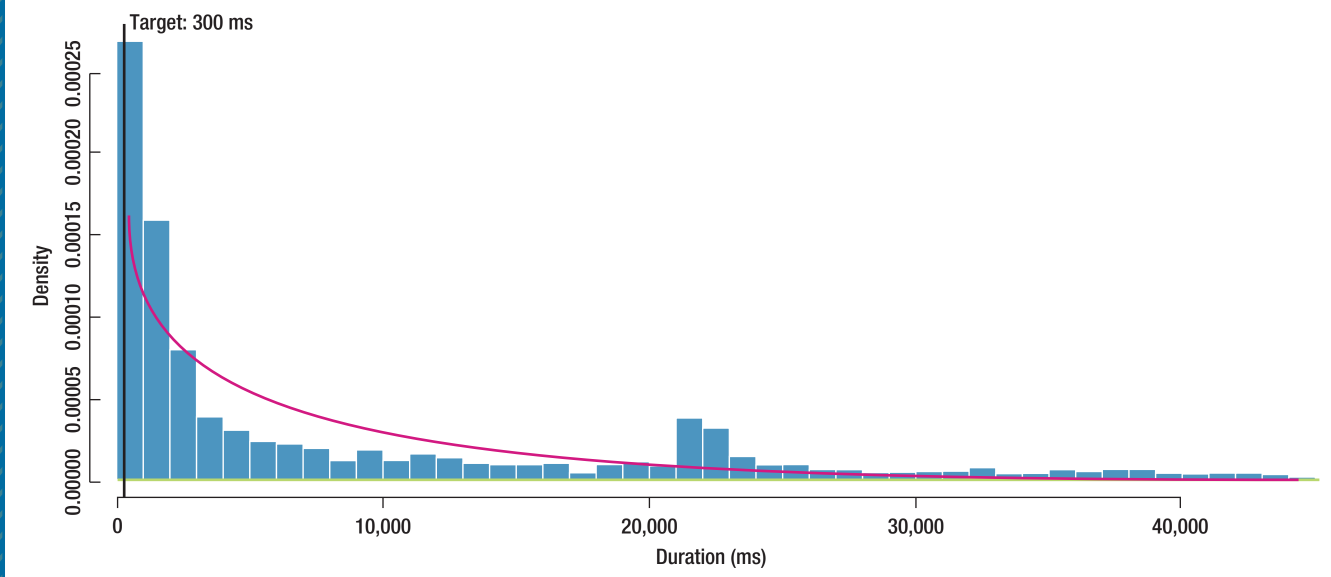

FIGURE 6. A histogram of durations for a scenario for a given build. Note the peak around 21,000 milliseconds, which alerted testers to an IPv6 timeout defect that didn't manifest in standard testing.

Detected Performance Issues

To demonstrate the way that EI Analytics has improved performance evaluation, we share two performance defects that it uncovered that would not have been identified using prior, conventional methods. These defects are representative of many issues that our approach has found.

Three ways to go. Soon after deploying our system and examining the data being returned by it, we saw anomalies in one broadly used scenario. We found that this scenario was being invoked in three different ways. Much to our surprise, the scenario's performance closely correlated with how it was invoked—one method of invocation took more than 10 times longer than the others, which led to further investigation and more fine-grained data gathering. The method of invocation that took the longest was a result of passive use of the scenario, so users and testers didn't explicitly notice it. We were unaware of the impact on performance because it had gone unnoticed until we began using EI Analytics.

IPv6. Figure 6 depicts data for a frequent scenario from an early build of Lync. Although the desired time was around 300 milliseconds, a second peak was visible around 21,000 milliseconds. Although members of the ship room recognized that the behavior was aberrant, they couldn't immediately determine the cause. However, domain experts who were familiar with recent system changes were able to quickly determine that this was being caused by timeouts in the IPv6 stack. Performance results from the test lab on the same build of the application didn't show such a problem: the test machines that used IPv6 weren't running the code that triggered the timeouts.

After developers deployed a fix, the data showed that the next build still had the bump, albeit smaller, around 21,000 milliseconds. Although performance had improved for many users, some were still encountering the issue. Neither the initial problem, nor the fact that the fix didn't completely resolve it, would have been determined without EI Analytics.

This experience also taught us that although our approach is strong at showing if there are problems and which scenarios they relate to, these are simply signposts rather than ways to identify the cause of the problem. That task still requires domain expertise. The value of our approach is that it maximizes the effectiveness of experts' time by pointing them to issues quickly.

These successes with the Lync team have encouraged three other network-based products and services to adopt our tool and monitoring technique.

Currently, testers determine specification thresholds. Ideally, we would like to know precisely what's acceptable to the majority of users and set that value as the goal. This might be derived from usage and performance data or from satisfaction surveys and will enable us to know where to place our emphasis in improving performance.

Our technique currently displays the results of the analysis to project stakeholders, but it doesn't analyze the data to provide recommendations. Because the data can be sliced in many ways, it would be useful to have analysis that automatically looks for problems. For example, one analysis might look through the data to see if there are specific geographical regions experiencing poor performance and alert a team member who could then use fine-grained data to investigate possible causes.

Our technique has been used with beta testers, but we plan to continue using it after release. Lync users will be able to opt in to the data collection program, at which point we can collect timing data to track performance and identify problems as they occur after release. We plan to continue improving performance monitoring and analysis as we strive for data-driven decision making in development.

References

1. A. Avritzer and E. Weyuker, "The Automatic Generation of Load Test Suites and the Assessment of the Resulting Software," *IEEE Trans. Software Eng.*, vol. 21, no. 9, 1995, pp. 705–716.
2. G. Kiczales et al., "Aspect-Oriented Programming," *Proc. European Conf. Object Oriented Programming* (ECOOP 97), Springer, 1997, pp. 327–353.

ABOUT THE AUTHORS

ROBERT MUSSON is a principle data scientist on the Lync/Skype team at Microsoft. His research interests include process data and trends to improve the quality of the Lync product. Contact him at rmusson@microsoft.com.

JACQUELINE RICHARDS is a program manager lead at Microsoft. Her research interests include increasing the engineering capabilities of teams by delivering shared, Microsoft-wide engineering services and systems. Richards received a BA from the University of South Florida. Contact her at jacqrich@microsoft.com.

DANYEL FISHER is a researcher in information visualization and human-computer interaction at Microsoft Research. His research interests include ways to help people work together around big data by providing easy-to-use data visualizations. Fisher received a PhD in information and computer science from the University of California, Irvine. Contact him at danyelf@microsoft.com.

CHRISTIAN BIRD is a researcher in empirical software engineering at Microsoft Research. His research interests include how large teams develop software, both in industrial and open source contexts. Bird received a PhD in computer science from the University of California, Davis. Contact him at cbird@microsoft.com.

BRIAN BUSSONE is a principal development lead on the Services Engineering Team at Microsoft. His research interests include providing insights from data. Bussone received a BS in electrical engineering from Michigan Technological University. Contact him at bbussone@microsoft.com.

SANDIPAN GANGULY is a senior data scientist at Microsoft. His research interests include developing machine learning algorithms using MapReduce for big data environments. Ganguly received a PhD in industrial engineering from Arizona State University. Contact him at sagangul@microsoft.com.