Performance: See a Bigger Picture

There are many discussions about performance, and they often concentrate on only one specific facet of performance. But, the main problem is that performance is the result of every design and implementation detail, so one can't ensure performance approaching it from a single angle only.

There are different approaches and techniques to alleviate performance risks, such as:

- ***Single-User Performance Engineering***. Profiling, tracking and optimization of single-user performance, Web Performance Optimization (WPO), etc. Everything that helps to ensure that single-user response times, the critical performance path, match our expectations.
- ***Software Performance Engineering (SPE)***. Performance patterns and anti-patterns, scalable architectures, modeling, etc. Everything that helps in selecting appropriate architecture and design and proving that it will scale according to our needs.
- ***Instrumentation / Application Performance Management / Monitoring***. Everything that provides insights in what is going on inside the working system and tracks down performance issues and trends.
- ***Capacity Planning / Management.*** Everything that ensures that we will have enough resources for the system.
- ***Load Testing***. Everything used for testing the system under any multi-user load (including all other variations of multi-user testing, such as performance, concurrency, stress, endurance, longevity, scalability, etc.).
- ***Continuous Integration / Deployment.*** Everything allowing quick deployment and removal of changes, decreasing the impact of performance issues.

And, of course, all the above does not exist in a vacuum, but on top of high-priority functional requirements and resource constraints (including time, money, skills, etc.).

Every approach or technique mentioned above somewhat mitigates performance risks and improves chances that the system will perform up to expectations; However, none of them guarantees that. And, moreover, none completely replaces the others, as each one addresses different facets of performance.

Let's look, for example, at load testing. Recent trends of agile development, DevOps, lean startup, and web operations somewhat question the importance of load testing. Some (not many) are openly saying that they don't need load testing while others are still paying lip service to it – but just never get to it. In more traditional corporate worlds we still see performance testing groups and important systems getting load tested before deployment.

Yes, the ways to mitigate performance risks mentioned above definitely can decrease performance risk compared to situations where nothing is done about performance at all until the

last moment before rolling out the system in production without any instrumentation, but they still leave risks of crashing and performance degradation under multi-user load. And if its cost is high, you should do load testing (how exactly is another large topic as there is much more to it than the stereotypical waterfall-like last-moment record-and-replay approach).

There are always risks of crashing a system or experiencing performance issues under heavy load – but the only way to mitigate them is to actually test it. Even stellar performance in production and a highly scalable architecture don't guarantee that it won't crash with a slightly higher load. Even load testing doesn't completely guarantee it (for example, real-life workload may be different from what you have tested), but it significantly decreases the risk.

Another important value of load testing is making sure that changes don't degrade multi-user performance. Unfortunately, better single-user performance doesn't guarantee better multi-user performance. In many cases it improves multi-user performance too, but not always. And the more complex the system is, the more likely exotic multi-user performance issues can be. Load testing is the way to ensure that you don't have such issues.

And when you do performance optimization, you need a reproducible way to evaluate the impact of changes on multi-user performance. The impact of the changes on multi-user performance won't probably be proportional to what you see with single-user performance (even if it still would be somewhat correlated). The actual effect is difficult to quantify without multi-user testing. The same with the issues happening only in specific cases that are difficult to troubleshoot and verify in production – using load testing can significantly simplify the process.

It may be possible to survive without load testing by using other ways to mitigate performance risks and if the cost of performance issues and downtime is low. However, it actually means that you use customers and/or users to test your system, addressing only those issues that pop up; this approach become risky once performance and downtime start to matter.

Moreover, with existing trends of system self-regulation (such as auto-scaling or changing the level of services depending on load), load testing is needed to verify that functionality. You need to apply heavy load to see how auto-scaling will work. So load testing becomes a way to test functionality of the system, blurring the traditional division between functional and non-functional testing.

Summarizing, I don't see that the need for load testing is going away. Even in case of web operations, we would probably see load testing coming back as soon as the systems become more complex and performance issues start to hurt business. There perhaps would be less need for "performance testers" as it was at the heyday due to better instrumenting, APM tools, continuous integration, resource availability, etc. – but I'd expect more need for performance experts that would be able to see the whole picture using all available tools and techniques.