CASE STUDY

Is scaling always the right answer? Insights from Performance Testing with JMeter

Rakibul Hasan 09 Dec 2024 0 78 0

Share

When an application faces heavy traffic, the first thought could be: **"Let's add more servers or resources!"** While this can sometimes help, it's not always the best solution.

My recent performance testing experiment with **JMeter** discloses that while scaling can improve some issues. In fact, it may hide the real problems into the system..

Let's look at why simply adding resources (scaling) is not enough what you can do to fix performance issues properly.

What is Performance Testing?

Performance testing is like giving your app a performance/load/stress test to see how it performs when many users using it at the same time.

- **Response Time:** How rapidly the app responds to a user.
- **Error Rate:** How frequently does the app fail to respond correctly.
- **Throughput:** How many requests the app can handle /per second.

Why Do We Add Resources (Scale)?

When performance issues appear, many teams first thought may be: **“Let’s add more servers to handle the load!”** which means:

- Adding servers/machines.
- Increasing the no. of containers/pods in Kubernetes.
- More CPU, memory/ storage to the app.

Scaling can help the app handle more users, but it’s not a magic fix for every problem.

Why Scaling Isn’t Always the Solution

Here are some reasons why scaling/adding resource (e.g., more servers or pods) doesn’t solve all performance issues

1. The real problems remain disappeared/hidden

Adding resources doesn’t fix bad code, slow database queries, or poor app design. These problems will still exist, even with more resources.

2. Shared resources turn into overloaded

Even if your app has more servers, it still depends on shared resources like databases or caches. If those can’t keep up, the system will still slow down.

3. It costs a lot

Scaling means spending more money on infrastructure, and if the app isn’t fixed, it’s like pouring water into a leaking bucket—wasteful and expensive.

4. Errors multiply

If the app has issues, like bugs/authentication errors, adding more resources can make the problem worse by spreading it across more servers.

A Better Approach: Fix First, Scale Later

Instead of scaling right away, focus on improving the system’s efficiency. Here’s how:

Instead of just adding more resources, try these steps to improve your app:

1. Optimize code and database query

- Identify the slow parts of the code and improve them.
- Use database indexing to make faster.

2. Use caching

- Use caching to save frequently-used data in tools like Redis or Memcached.
- This reduces the load on the database and speeds up responses for users.

3. Monitor your app

- Use tools like Grafana or Prometheus to see where the system is slow or failing.·
- Look at metrics like memory usage, request latency, and database performance.

4. Fix errors under load

- Investigate why it fails/errors when the app is busy and fix those issues.
- Add retry systems to handle temporary failures better.

5. Balance the load

- Use load balancers to spread traffic evenly across the servers or pods.

CPU, memory, or custom metrics (e.g., request latency).
- Be careful not to set over-scale—set limits to avoid wasting resources.

When to Scale

Scaling is useful when the app is already well-optimized, but traffic is genuinely too high to handle. For example:

- During busy/peak times (e.g., during a product launch or a live event)
- When the app's user base grows steadily over time.

In these cases, scaling works best when combined with proper optimization to avoid overspending or worsening existing issues.

Conclusion: Think Before You Scale

Scaling is useful, but it's not the first step to solving performance issues. Start by improving your app's code, database, and overall architecture. Once the system is efficient, you can scale to meet growing demand without wasting resources or money.

What's Your Experience?

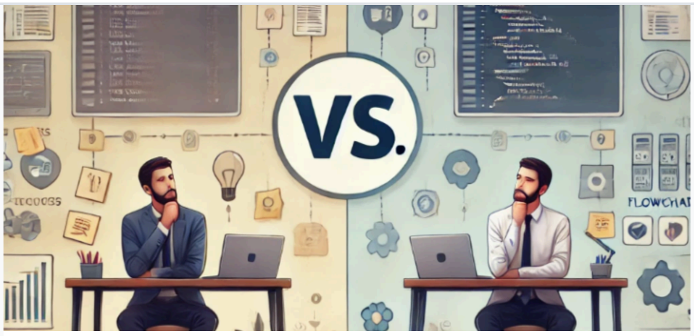
Have you faced performance issues in your projects? How did you resolve them? Share your experiences and tips in the comments below—we'd love to hear from you.

👉 The original content link here

qa · testing tool · softwaretestingtools · jmeter · #sqa · #performancetesting · qatools · loadtesting · softwaredevelopment · qaprofessionals

Share your thoughts Or Start discussion

Related Blogs



CASE STUDY 0 0 27

Can a QA build his career with only manual testing skills or ultimately does he need...

Can someone build a career in QA with only manual testing skills or automation i

Ali Hasan
24 Mar 2025

CASE STUDY 0 0 15

White Box vs Black Box Testing

🚀 White Box vs Black Box Testing: What Every QA Should Know!As a QA Engineer,

Anirudha
24 Mar 2025

Popular Tags

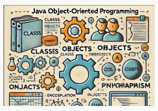
sqa · testing · qa · software testing · qabrains · testing tool · automationtesting · softwaretesting · mobiletesting · selenium

View All

Popular Post

Can a Software Tester Become a Game Tester? Here's What You Need t...

As the gaming industry continues to grow, fueled by innovations in virtual reali



Understanding Java Object-Oriented Programming (OOP) Concepts

Java is a powerful and widely used programming language known for its versatilit

Essential Bugs to Check for in Game Testing: A Guide for Beginners

Game testing is crucial to ensure a smooth, engaging, and bug-free experience fo

Popular Discussion

01 Top Software Testing Interview Questions and Expert Tips from QA Leaders

02 AI tools for QA engineer

03 What is SQL?

04 Appium, WebDriver

05 What are the most effective strategies you've found for balancing speed and...

View All

QA Brains

QA Brains is the ultimate QA community to exchange knowledge, seek advice, and engage in discussions that enhance Quality Assurance testers' skills and expertise in software testing.

QA Topics

Web Testing

Interview Questions

Game Testing

See more →

Quick Links

Discussion

About Us

Terms & Conditions

Privacy Policy

Follow Us

QA BRAINS