Article

Application of Efficient Load Test Strategies in Infrastructure

Buqin Wang [1,*]

[1] Meta Platforms / Infrastructure, Menlo Park, CA, 94025, United States
[*] Correspondence: Buqin Wang, Meta Platforms / Infrastructure, Menlo Park, CA, 94025, United States

Abstract: Load testing is an important means of evaluating the performance of infrastructure under different load conditions. This article discusses the application of efficient load testing strategies in infrastructure, with a focus on analyzing automatic scalability testing, high concurrency load simulation, and virtual machine load testing. It reveals the practice of building diverse load scenarios to test the reliability and stability of infrastructure. Further elaborated on measures to ensure efficient load testing, including utilizing automation and intelligent tools to improve testing efficiency, optimizing resource allocation through elastic resource management and intelligent scheduling, and adopting layered and continuous testing to ensure comprehensive and accurate testing coverage, providing theoretical basis and practical guidance for load testing in practical operations.

Keywords: load testing; infrastructure; automatic scaling; high concurrency; virtual machine

Received: 10 May 2025
Revised: 14 May 2025
Accepted: 29 May 2025
Published: 03 June 2025

1. Introduction

Against the backdrop of the increasing expansion of infrastructure scale and the upgrading of technological difficulty, load testing is an important link in ensuring the stable operation of systems under high loads. By simulating different load scenarios, load testing can effectively evaluate the performance bottlenecks of the system and reveal potential risk points. Especially in technological environments such as cloud computing, big data, and virtualization, traditional load testing methods face new challenges. Efficient load testing not only requires the use of intelligent tools to improve testing efficiency, but also needs to be combined with strategies such as elastic resource management and intelligent scheduling to ensure efficiency and accuracy in a dynamically changing environment.

2. Definition and Principle of Load Testing

Load testing is a type of performance testing, whose core purpose is to test the system's responsiveness and stability under specific loads. By simulating actual user operations or preset load scenarios, load testing can reveal potential constraints on system performance. Its main objective is to measure the response time, throughput, concurrent processing capability, and resource consumption of the system under different load conditions, in order to ensure the stability and reliability of the system under high loads [1].

The principle of load testing is to simulate the running load of the system by creating virtual users or requests, and gradually increase the load during the testing process to monitor the performance of the system. These virtual user requests mimic various behaviors of real users, including database queries, file uploads and downloads, page loading, and other diverse operations. Using load generation tools such as JMeter and LoadRunner, these requests are sent to the target system, and the system's resource usage (such as CPU, memory, disk I/O, and network bandwidth) is monitored in real-time.

When conducting load testing, the key lies in the system's load-bearing limit, which is the maximum concurrent requests that the system can handle, as well as the response time, which is the system's processing speed under high load. In addition, load testing helps to confirm the reliability of the system and verify whether the system can maintain normal operation without failure in the face of continuous or sudden high loads [2]. At the same time, load testing also involves the scalability of the system, observing whether the testing system can effectively expand resources to meet greater demand as the load gradually increases.

3. The specific Application of Efficient Load Testing in Infrastructure

3.1. Automatic Scalability Test

In efficient load testing, automatic scalability testing is an important step in verifying whether the system can dynamically expand or contract resources as needed under load fluctuations. By constructing various load scenarios and monitoring the automatic scaling response of the system, its ability to cope with high concurrency and sudden loads can be evaluated [3].

Assuming that the load of the system at time t is $L(t)$, the resource capacity of the system is $C(t)$, and $C(t)$ automatically adjusts with changes in load. The load threshold and scaling rules can be achieved by assuming that the system has set a load threshold L_{max} and L_{min}. When the system load $L(t)$ exceeds L_{max}, the automatic scaling mechanism will trigger resource expansion. When the load drops to L_{min}, the automatic contraction mechanism will be activated. The scaling operation of the system can be represented by the following formula:

$$C(t) = \begin{cases} C(t-1) + \Delta C, & \text{if} L(t) > L_{max} \\ C(t-1) - \Delta C, & \text{if} L(t) < L_{min} \\ C(t-1), & \text{otherwise} \end{cases} \quad (1)$$

Among them, ΔC represents the amount of resource change during each scaling. The response time $R(t)$ of the system will be affected by the current load $L(t)$ and resource capacity $C(t)$. The relationship between load and response time can be obtained:

$$R(t) = \frac{f(L(t), C(t))}{C(t)} \quad (2)$$

Among them, $f(L(t), C(t))$ is the time function required for the system to process the load $L(t)$. The resource utilization rate $U(t)$ of the system is defined as the proportion of current resource usage:

$$U(t) = \frac{L(t)}{C(t)} \quad (3)$$

When the load $L(t)$ exceeds the resource $C(t)$, the system's resource utilization rate $U(t)$ will exceed 1, indicating system overload and the need for expansion. By simulating different load working conditions, calculating the above formula, and analyzing the effectiveness of scaling strategies, it can be verified whether the system can allocate resources appropriately according to actual needs to achieve efficient load management under various load conditions [4].

3.2. High Concurrency Load Simulation

Conducting high concurrency load simulation experiments is crucial for evaluating the operational efficiency, stability, and reliability of a system under immense access pressure. In such experimental processes, the system must respond to synchronous actions from numerous users or requests, and then quantitatively analyze key performance indicators such as response time, throughput, and resource utilization under high concurrency conditions [5].

Assuming the number of concurrent requests processed by the system at time t is $N(t)$, these requests come from different users or applications. The inflow of requests can be viewed as a stochastic process, and a Poisson process can be used to model the pattern of request arrival. Assuming the arrival rate of requests is λ, the average number of requests received per unit time $N(t)$ follows a Poisson distribution:

$$P(N(t) = n) = \frac{(\lambda t)^n e^{-\lambda t}}{n!} \quad (4)$$

Among them, n represents the number of requests that arrive within the time t, and λ is the average request arrival rate per unit time. And the throughput $T(t)$ of the system is the number of requests processed by the system per unit time. The throughput is directly related to the concurrency capability of the system. When the system load reaches a bottleneck, the throughput will reach its maximum value, which is called the "peak throughput" of the system. Assuming the maximum throughput of the system is T_{max}, the throughput of the system under load $N(t)$ can be approximated by the following formula:

$$T(t) = \min(\lambda t, T_{max}) \quad (5)$$

Among them, $T(t)$ increases with the increase of concurrent requests, but when the maximum throughput is reached, the system's throughput no longer increases. The response time $R(t)$ of the system refers to the time required for the system to process requests. The response time usually increases with the increase of concurrent requests, especially under high concurrent loads. Given that the response time is related to the current load and system resources, it can be represented by the following model:

$$R(t) = \frac{C(t)+\alpha N(t)}{T(t)} \quad (6)$$

Among them, $C(t)$ is the basic response time of the system (the response time when not affected by load), α is the coefficient representing the impact of system load on response time, $N(t)$ is the current number of concurrent requests, and $T(t)$ is the system throughput. By simulating different concurrent load scenarios and applying the above mathematical formulas, the performance of the system under high concurrent loads can be comprehensively evaluated, potential bottleneck problems can be identified, and system optimization design can be guided to improve its performance in actual production environments.

3.3. Virtual Machine Load Test

The virtual machine load test aims to evaluate the performance, resource allocation, and scalability of virtual machines (VM) under different load conditions in a virtualized environment. By simulating different types of workloads, we can comprehensively understand the response and efficiency of virtual machines in processing computing tasks, storage operations, network traffic, and other aspects, thereby optimizing the resource scheduling and management of virtualization platforms.

In a virtualization environment, the resources of each virtual machine (such as CPU, memory, and disk space) are allocated from the resource pool of the physical server. Assuming that the total resources of the physical host are R_{total} and the resource requirements of the virtual machine are R_i. In order to ensure system stability, the resource allocation R_i of virtual machines should meet the following requirements:

$$\sum_{i=1}^{n} R_i \leq R_{total} \quad (7)$$

Among them, n is the number of virtual machines. The CPU, memory, storage, and network bandwidth resources of each virtual machine should be dynamically allocated according to its load requirements. When assuming that the CPU usage of a virtual machine is $U_{cpu}(t)$, under load conditions, the CPU usage varies with the workload of the virtual machine. If the load model follows a negative exponential distribution, the CPU usage of the virtual machine can be expressed as:

$$U_{cpu}(t) = 1 - e^{-\lambda t} \quad (8)$$

Among them, λ is the arrival rate of the load, and t is the time. As the workload increases over time, the CPU usage of the virtual machine gradually increases until it approaches its maximum capacity. The relationship between virtual machine response time and throughput, under high load, the response time $R(t)$ of virtual machines will increase with the increase of system load. Assuming the throughput of the system is $T_{vm}(t)$, the response time can be described by the following formula:

$$R(t) = \frac{f(L(t), R_{total})}{T_{vm}(t)} \quad (9)$$

Among them, $f(L(t), R_{total})$is a function of load and total resources, representing the impact of load on response time, while $T_{vm}(t)$ is the throughput of the virtual machine, affected by the current load $L(t)$. By mathematically modeling resource allocation, CPU usage, memory utilization, I/O performance, and network bandwidth in virtual machine load testing, we can better understand the performance of virtual machines in high load environments, and provide a basis for optimizing virtualization platforms and load balancing strategies.

4. Measures to Ensure Efficient Load Testing in Infrastructure

4.1. Utilize Automation and Intelligent Tools

Automation and intelligent tools are key technologies for improving efficiency and accuracy in load testing. Automated tools greatly reduce the pressure of manual testing and minimize the possibility of operational errors by simulating a large number of concurrent requests, ensuring the consistency and credibility of test results. These tools play an indispensable role in the load testing process, capable of simulating various business scenarios and load patterns, thereby providing accurate performance indicators for system evaluation.

Intelligent tools combine machine learning and artificial intelligence technologies, and can autonomously optimize load testing solutions based on real-time data feedback. For example, intelligent tools can predict potential performance bottlenecks in the system by analyzing historical data, and then allocate load reasonably, adjusting the concurrency level and request method of testing in real time. These tools can not only identify performance barriers in the system, but also provide optimization suggestions based on test results, significantly enhancing the system's performance. For example, common automated load testing tools such as JMeter and Gatling can simulate scenarios of concurrent user requests and collect detailed performance metrics. Additionally, intelligent tools such as Test.ai can use the data generated during the testing process to automatically adjust the load strategy, helping testers reveal potential risk points in the system.

As shown in Table 1, the integration of automation and intelligent tools not only significantly improves the precision of load testing, but also makes the process of identifying and resolving system performance bottlenecks more efficient.

Table 1. Advantages and Characteristics of Automation and Intelligent Tool Functions.

Tool type	Major function	Advantage	Feature Description
Automated testing tools	Simulate concurrent user requests, generate performance reports, and support scenario simulation	Reduce manual intervention, improve testing efficiency, and generate standardized reports	High concurrency simulation, supporting distributed testing, real-time monitoring of performance data
Intelligent testing tool	Automatically optimize testing strategies, predict system bottlenecks, and adjust load generation	Based on historical data and real-time feedback, dynamically adjust testing strategies to accurately predict bottlenecks	AI driven load optimization, intelligent monitoring, and real-time feedback
Performance monitoring tool	Real time monitoring of system, detection of resource consumption, and generation of performance data reports	Provide real-time data support to help identify potential performance bottlenecks	Display performance data in chart format, supporting monitoring in big data environments

4.2. Through Elastic Resource Management and Intelligent Scheduling

Elastic resource management and intelligent scheduling are key means to ensure that the system can cope with high concurrency loads during load testing. Elastic resource management can automatically adjust computing, storage, and network resources according to actual workload changes, expanding resources under high load conditions and reclaiming them when the load decreases to avoid waste. This flexible resource allocation not only helps to continuously optimize system performance, but also helps to reduce operational economic costs.

Intelligent scheduling optimizes the use of resources through reasonable algorithms and strategies, ensuring that various resources are allocated appropriately according to their importance and actual load conditions. For containerization and virtualization environments, platforms such as Kubernetes provide automated resource scheduling capabilities that can automatically start new container instances when system load increases to ensure service persistence and reliability. For example, AWS Auto Scaling and Kubernetes can automatically expand and reduce computing resources based on real-time changes in load, helping maintain service stability under varying load pressures. In some complex business scenarios, intelligent scheduling systems have the ability to detect fluctuations in resource demand and plan resource adjustments in advance, thereby preventing performance degradation caused by sudden load increases.

As shown in Table 2, through elastic resource management and intelligent scheduling, the system can flexibly adjust resource configuration according to load fluctuations, ensuring stability and efficiency in high load environments.

Table 2. Strategy Analysis of Elastic Resource Management and Intelligent Scheduling.

Strategy type	Major function	Advantage	Characteristic description
Elastic Resource Management	Automatically adjust computing, storage, and network resources based on load	Dynamically allocate resources to avoid resource waste and improve performance	Automatically scaling based on peak load to ensure efficient system operation
Intelligent dispatching	Allocate resources based on task priority and load to avoid resource contention	Optimize resource allocation, reduce bottlenecks, and improve resource utilization efficiency	Automatically schedule containers based on load conditions, supporting load balancing between containers and virtual machines
Resource monitoring and early warning	Real time monitoring of resource usage, prediction of future resource demands, and allocation	Provide load change prediction, automate resource adjustment, and improve responsiveness	Predict future load changes, provide system resource allocation suggestions, and avoid bottlenecks

4.3. Adopting Layered and Continuous Testing

Layered and continuous testing is a method of conducting load testing on different levels of a system, such as databases, applications, networks, in sequence. This strategy can help testers analyze and identify performance bottlenecks from various levels, ensuring that each module can operate normally under different load conditions. Through layered testing, developers can accurately identify and eliminate specific performance barriers in the system, avoiding unnecessary time and resource consumption in global testing.

Continuous testing integrates load testing into the CI/CD process, allowing automatic validation of load performance after each system change. This method ensures that every code submission is accompanied by thorough load testing, in order to promptly detect any performance issues that may arise after the system update. Combining layering and continuous testing can ensure that all levels and overall architecture of the system can operate smoothly under load pressure. For example, in a typical layered testing, performance testing is first conducted on the application layer to verify its response time and throughput under high concurrency requests, and then independent testing is conducted on the database layer to analyze its performance when handling a large number of concurrent queries. Continuous testing is automatically conducted after each update, ensuring consistency and reliability of system performance through automated processes.

As shown in Table 3, through layering and continuous testing, the system is able to fully identify potential hazards during load testing and maintain stable performance throughout continuous development iterations.

Table 3. Advantages and Implementation Details of Layered and Continuous Testing.

Strategy type	Major function	Advantage	Characteristic description
Layered testing	Conduct independent load testing for different levels and analyze performance bottlenecks	Accurately locate the bottleneck of each module, optimize layer by layer, and improve the accuracy of testing	Conduct independent performance analysis at the system level to improve optimization efficiency
Continuous load testing	Integrate load testing into the CI/CD process for automated testing	Ensure that system performance does not degrade after each update, and quickly identify and fix performance issues	Automatically trigger testing to ensure stable system load capacity after code changes
Performance analysis tools	Collect performance data and conduct detailed analysis to identify potential bottlenecks	Provide performance data to assist the development team in analyzing and optimizing system performance	Provide system performance reports based on data analysis to help identify potential issues

5. Conclusion

In infrastructure load testing, adopting efficient load testing strategies is the key to ensuring system stability and high performance. The application of automation and intelligent tools can significantly improve the efficiency and accuracy of load testing, reduce human intervention, and quickly identify system performance bottlenecks. The combination of elastic resource management and intelligent scheduling strategies enables the system to adapt to various workload conditions, achieve maximum resource utilization, and avoid waste. Layered and continuous testing ensure stable operation of each level of the system under different loads through refined analysis and automated integration. By utilizing efficient load testing strategies, organizations can ensure the sustainable development and stable operation of their systems.

References

1. T. Fan, W. Guo, Z. Zhang, and Z. Cui, "A many-objective optimization based intelligent algorithm for virtual machine migration in mobile edge computing," *Concurrency Comput.: Pract. Exp.*, vol. 35, no. 23, p. e7770, 2023, doi: 10.1002/cpe.7770.
2. A. R. Hummaida, N. W. Paton, and R. Sakellariou, "A hierarchical decentralized architecture to enable adaptive scalable virtual machine migration," *Concurrency Comput.: Pract. Exp.*, vol. 35, no. 2, p. e7487, 2023, doi: 10.1002/cpe.7487.
3. D. A. Zaitsev, T. R. Shmeleva, Q. Zhang, and H. Zhao, "Virtual machine and integrated developer environment for Sleptsov net computing," *Parallel Process. Lett.*, vol. 33, no. 3, p. 2350006, 2023, doi: 10.1142/S0129626423500068.

4. E. B. Dano, "Systems engineering integration and test challenges due to security measures in a cloud-based system," *INCOSE Int. Symp.*, vol. 32, no. 1, pp. 224–232, Jul. 2022, doi: 10.1002/iis2.12927.
5. A. Hassannezhad Najjari and A. A. Pourhaji Kazem, "A systematic overview of live virtual machine migration methods," *Concurrency Comput.: Pract. Exp.*, vol. 34, no. 17, p. e6915, 2022, doi: 10.1002/cpe.6915.