

Strategies and Practices for Load Test Optimization in Distributed Systems

Buqin Wang

Meta Platforms / Infrastructure, Menlo Park, CA, 94025, US

Abstract

Distributed system load testing is the key to evaluating the performance of a system under high concurrency and extreme loads. With the continuous expansion of system scale, issues such as network latency, bandwidth limitations, difficulty in predicting system capacity, and thread contention and deadlocks have gradually become obstacles to system stability and performance. To address these issues, this article proposes a series of improvement measures, including optimizing network topology and reducing cross node communication latency. Layered testing and progressive load increase help accurately identify system bottlenecks. Using asynchronous processing and refining lock granularity to reduce thread contention and deadlocks. Through these optimization measures, the concurrent processing capability of distributed systems can be enhanced, their scalability and stability can be improved, ensuring smooth operation of the system under high loads.

Keywords

Distributed System; Load Testing; Network Latency; Bandwidth Limitation; Thread Contention.

1. Introduction

With the widespread application of distributed systems in modern enterprise architecture, the scalability, flexibility, and efficiency of the system make it an ideal choice for handling large-scale data and high concurrency tasks. But with the continuous expansion of the system scale, how to ensure that the system can operate stably and efficiently under high loads has become an important issue in design and operation maintenance. Load testing, as a key means of evaluating system carrying capacity and performance bottlenecks, is crucial for improving system reliability. The purpose of this article is to discuss optimization strategies and practices for load testing in distributed systems, analyze common problems encountered during the load testing process, and provide effective solutions to help developers achieve performance prediction during the design and deployment phases, ensuring stable system operation in high concurrency environments.

2. The Necessity of Load Testing for Distributed Systems

With the rapid development of Internet and cloud computing technology, distributed systems have become the core of modern enterprise and service architecture. Distributed systems can distribute computing and storage tasks across multiple nodes, improving the scalability, fault tolerance, and reliability of the system. However, with the continuous expansion of system scale and the gradual upgrading of service load, ensuring the stability and performance of the system in high concurrency, massive data, and complex environments has become a major challenge. In this context, load testing is particularly important as a key method for evaluating the performance of distributed systems, identifying potential bottlenecks, and improving system stability. The main purpose of conducting distributed system load testing is to verify the

performance of the system under different load conditions, including key indicators such as response time, throughput, and concurrent processing capability. By simulating different numbers of user requests, data transmission, and computing tasks, load testing can reveal performance bottlenecks that the system may encounter in actual operation, and help determine the system's scalability and fault tolerance. A major highlight of distributed systems is their ability to scale horizontally, but in practical applications, as the load increases, the scalability of the system may be affected by factors such as network latency, bandwidth limitations, and resource contention. Through load testing, the performance of the system during the expansion process can be evaluated in order to detect and make corrections early. In addition, load testing is also an effective way to verify system stability and reliability, especially when the system encounters high loads, high concurrency, or extreme environments. By simulating long-term high-intensity load operation, load testing can evaluate the stability of the system under extreme conditions, ensuring that the system can operate continuously and stably in real production environments.

3. Problems in Load Testing of Distributed Systems

3.1. Network Latency and Bandwidth Limitations

In distributed systems, network latency and bandwidth limitations become the main obstacles that constrain performance. Especially in large-scale distributed architectures, the communication quality between system nodes directly affects the overall response time and throughput of the system. Network latency refers to the time required data to transmitted node to another, while bandwidth limitation refers to the amount of data that the system can transmit within a certain period of time.

Table 1. Impact of Network Latency and Bandwidth Limitations on Distributed System Performance

Problem	Effect
Network delay	High latency significantly increases request response time, especially in cross regional or cross data center systems. The delay will significantly increase with the increase of communication paths and distances between nodes, leading to a decrease in real-time system performance and a deterioration of user experience.
Bandwidth limitation	Insufficient bandwidth limits the data transmission rate, resulting in a decrease in throughput and affecting the processing of high traffic data. In data intensive applications, bandwidth bottlenecks may result in request timeouts, data loss, or connection interruptions.
Cross node communication bottleneck	When the nodes in the system are widely distributed, especially when deployed across regions, communication between nodes may become a bottleneck, affecting overall system performance.
Network instability	Unstable phenomena such as network fluctuations, jitter, or packet loss can cause system performance fluctuations, which may affect the reliability of requests and increase error rates. In high concurrency scenarios, network instability may lead to a significant decrease in system throughput, resulting in connection disconnections, loss of requests, and in severe cases, system crashes or prolonged unavailability.

As shown in Table 1, network latency may significantly increase system response time, especially in distributed systems deployed across regions where latency is more pronounced. The bandwidth limitation will restrict the throughput of data, thereby affecting the system's data processing capability in high concurrency situations, resulting in a decrease in the speed of processing requests.

3.2. Difficulty in Predicting System Capacity

In distributed systems, capacity prediction often encounters numerous difficulties. The characteristic of distributed architecture is that resources are widely distributed, loads are uneven and dynamically changing, and the system's resource consumption often shows non-linear growth.

Table 2. Difficulties in predicting system capacity and their impacts

Problem	Effect
Nonlinear resource consumption	Resource consumption shows non-linear growth with increasing load, and traditional linear models cannot accurately predict system demand. Under high load conditions, system resource consumption far exceeds expectations, which may lead to performance bottlenecks and resource shortages, affecting the overall stability of the system.
Node resource contention	Multiple nodes share computing, storage, and network resources, and resource competition can lead to resource saturation in some nodes, affecting the performance of other nodes. If the resource contention situation of each node is not accurately evaluated in the prediction, it will lead to capacity planning errors and performance degradation.
Capacity model missing	Lack of effective capacity assessment models leads to incomplete or inaccurate predictions. In distributed systems, resource requirements involve multiple dimensions such as computing, storage, and networking, and a single dimensional capacity prediction model cannot meet the diverse needs of the system.
Differences in different load modes	Different types of loads have significant differences in the consumption of system resources. Some load patterns may lead to system bottlenecks, while others have relatively lower resource requirements. Without accurate evaluation for different load modes, capacity prediction will not be able to adapt to complex load.

As shown in Table 2, the increase in load will lead to a changing trend in the resource requirements and performance indicators of various components within the system, which undoubtedly increases the difficulty of predicting system capacity. When predicting capacity, it is important not only to consider the demand for computing resources, but also to take into account the usage of resources such as storage and networking. Incorrect capacity prediction may lead to resource waste or system overload, and in severe cases, may even cause system crashes.

3.3. Thread Contention and Deadlock

Table 3. Thread contention and deadlock issues and their impacts

Problem	Effect
Thread contention	Multiple threads competing for shared resources (such as CPU, memory, disk, etc.) can cause thread blocking, increase waiting time, and reduce system throughput. Especially in high concurrency environments, thread contention can significantly affect system response time and even cause service unavailability.
Deadlock	A loop waiting is formed between threads, causing all participating threads to enter a blocking state, thereby preventing the system from continuing to execute. Deadlock can cause tasks to fail, systems to stall, and affect the reliability and stability of services.
The locking particle is too large	Excessive lock granularity can lead to multiple threads holding locks for a long time during execution, causing other threads to be unable to execute and resulting in performance bottlenecks. Excessive lock granularity not only affects concurrency, but may also lead to slower system response.
Thread pool resource contention	When concurrent requests increase, insufficient or improperly configured thread pool resources can cause threads to queue and wait, prolonging response time. The competition for thread pool resources makes it difficult for the thread pool to process new tasks in a timely manner, affecting the system's throughput and responsiveness, and may even cause timeout errors.

Thread contention and deadlock are common problems in concurrent systems, especially in distributed systems where multiple threads or processes may access shared resources

simultaneously. This resource competition often leads to performance bottlenecks or system crashes.

As shown in Table 3, through the analysis of thread contention and deadlock issues, it can be seen that these concurrency problems can seriously affect the performance of the system, especially in high concurrency environments. Thread contention and deadlock can lead to a decrease in system throughput, an increase in response time, and even system crashes. Therefore, effectively identifying and solving such problems is of decisive significance for enhancing the concurrent processing capability of the system and ensuring its stable operation.

4. Optimization Strategies for Load Testing in Distributed Systems

4.1. Optimize Network Topology Structure

In distributed systems, optimizing network topology plays a decisive role in improving system performance, reducing latency, and enhancing system scalability. Optimizing the network topology requires a reasonable planning of the distribution of nodes in the network to reduce communication latency and bandwidth bottlenecks. The network connections, traffic transmission paths, and load balancing strategies between each node in the system and other nodes have a direct impact on the overall performance of the system. In practical applications, reducing cross regional communication is a key strategy for reducing latency. Cross regional communication increases the delay caused by physical distance. Therefore, when deploying distributed systems, it is advisable to deploy critical services and data storage in the same region or data center as much as possible to avoid unnecessary network latency.

A reasonable load balancing mechanism is crucial for optimizing network topology. Load balancing can avoid overloading one node by intelligently allocating requests to multiple nodes, ensuring traffic balance throughout the entire system. Common load balancing strategies include polling, minimum number of connections, and IP hashing based. By dynamically adjusting the configuration of the load balancer, it is possible to cope with peak loads and improve system throughput.

In addition, the application of redundancy design and multi-path routing technology is also an effective method for optimizing network topology. Introducing redundant nodes and links in the network can improve the system's fault tolerance, ensuring that the system can continue to operate when a node or link encounters a problem. Through multi-path routing, the optimal data transmission path can be intelligently selected based on network load and operating conditions, further enhancing the system's fault tolerance. To measure the optimization effect of network topology, the following formula can be used to calculate network latency and throughput.

Delay optimization formula:

$$\text{Total Latency} = \sum_{i=1}^n (\text{Latency}_i \times \text{Traffic}_i) \quad (1)$$

Among them, Latency_i is the delay between each network node, Traffic_i is the traffic of the link, and n is the number of nodes in the network.

Bandwidth throughput optimization formula:

$$\text{Total Bandwidth} = \min(\sum_{i=1}^n \text{Bandwidth}_i, \text{Network Capacity}) \quad (2)$$

Among them, Bandwidth_i is the bandwidth of each link, and Network Capacity is the maximum carrying capacity of the system network. By optimizing these parameters, the system's throughput, response speed, and fault tolerance can be significantly enhanced. In load testing, quantifying the effectiveness of network topology optimization helps to deeply analyze system performance bottlenecks and provide scientific basis for system design and optimization.

4.2. Layered Testing and Progressive Load Increase

The strategy of layered testing and progressive load increase is crucial in the performance optimization process of distributed systems. Through hierarchical load testing, the system's performance under different load conditions can be gradually evaluated, while preventing system crashes caused by applying excessive loads at once. The gradual increase in load enables the team to promptly identify and resolve performance bottlenecks at various stages, ensuring stable system operation under high loads. Layered testing divides the system into multiple testing stages and conducts performance testing on each level one by one. The usual layering includes single component testing, which involves conducting stress tests on each independent component in the system (such as databases, API services, etc.) to evaluate the performance of a single component when the load increases. Subsystem testing starts testing the interaction between components, that is, the communication performance between subsystems, based on the qualified detection of individual components, and evaluates the performance after integration. Full system testing is the final stage, where stress testing is conducted on the entire system to simulate high concurrency situations in actual scenarios, in order to comprehensively evaluate the system's load bearing capacity.

Gradual load increase refers to gradually increasing the system's load to observe its operating conditions at different load levels. Through this method, it is possible to prevent the system from experiencing failures due to excessive load at once. Gradual load increase is generally carried out according to the following process. The initial load starts from a lower load and simulates a normal business scenario to ensure that the system can operate normally under low load. To gradually increase the load, test the system's load capacity and stability. High load testing is the final stage, simulating the situation under maximum load to observe whether the system can maintain stable response time and throughput. The step size of progressive load increase can be defined by the following formula:

$$\text{Load Increase Step} = \frac{\text{Max Load} - \text{Initial Load}}{n} \quad (3)$$

Among them, Max Load is the maximum load capacity of the system, Initial Load is the initial load, and n is the number of steps the load increases. The testing time for each stage can be dynamically adjusted based on indicators such as response time and throughput, gradually identifying system bottlenecks.

4.3. Using Asynchronous and Fine-Grained Lock Granularity

Using asynchronous processing and refining lock granularity are important strategies to enhance the concurrent processing capability of distributed systems. When facing high concurrency pressure, the competition between thread synchronization and locks often becomes a performance constraint. Asynchronous processing can effectively reduce thread blocking time, and refining lock granularity can alleviate competition between threads, thereby enhancing system responsiveness and throughput. Asynchronous processing improves overall throughput by placing long-running tasks in the background for execution, avoiding blocking the main thread. In distributed systems, common asynchronous processing methods include the event driven model, which is an asynchronous processing mechanism based on event triggering. This mechanism relies on events to trigger tasks, and once a task is completed, it triggers subsequent tasks. The message queue is used to process requests asynchronously. The system places the requests in the queue, which are then processed asynchronously by the backend worker thread to shorten response time.

In a concurrent environment, excessive granularity of locks can prevent many threads from working in parallel, thereby affecting concurrency efficiency. Refining lock granularity helps reduce lock competition and enhance system concurrency. Specific measures include using read-write locks to allow multiple threads to read resources simultaneously when reading

frequently but writing less, while reducing lock competition during writing. Distributed locks are used in distributed systems, such as RedLock, to ensure system concurrency and prevent global lock bottlenecks. By combining asynchronous and fine-grained locking, the system's throughput and response speed can be significantly improved, thread waiting time can be reduced, and concurrency performance can be optimized. In load testing, the effectiveness of asynchronous and lock optimization can be evaluated using the following formula:

$$\text{Async Performance Gain} = \frac{\text{Blocking Time}}{\text{Non-blocking Time}} = \frac{T_{\text{blocking}}}{T_{\text{non-blocking}}} \quad (4)$$

Among them, T_{blocking} and $T_{\text{non-blocking}}$ are the execution times of synchronous and asynchronous processing, respectively. The optimization effect of refining lock granularity can be measured by the lock competition rate:

$$\text{Lock Contention Rate} = \frac{\text{Time Spent Waiting for Lock}}{\text{Total Execution Time}} \quad (5)$$

By asynchronous processing and refining lock granularity, distributed systems can effectively enhance concurrent processing capabilities, reduce thread blocking time and lock contention, thereby optimizing system response speed, throughput, and scalability. Combining these two optimization strategies in load testing can help development teams identify performance bottlenecks and ensure stable system operation under high load conditions.

Conclusion: Load testing plays a crucial role in evaluating and optimizing the performance of distributed systems. Scientifically planning and executing load testing can accurately identify the performance shortcomings of the system when facing high concurrency pressure, enabling developers to timely locate and solve hidden problems, ensuring the robustness and reliability of system operation. Adopting strategies such as network topology optimization, layered testing and progressive load increase, asynchronous processing and refined lock granularity are important ways to enhance the performance of distributed systems. By optimizing these key aspects, the system's processing capacity can be effectively enhanced, response time can be reduced, and scalability can be optimized, thereby improving the overall user experience and the system's business processing capabilities. However, load testing optimization strategies are not achieved overnight and require continuous practical testing, gradual adjustment and optimization to meet the specific requirements of various application scenarios.

References

- [1] Tingarikar G , Choudhury A .Vibration Analysis-Based Fault Diagnosis of a Dynamically Loaded Bearing with Distributed Defect[J].Arabian Journal for Science & Engineering (Springer Science & Business Media B.V.), 2022, 47(7).
- [2] Jin W , Peng X .SLITS: Sparsity-Lightened Intelligent Thread Scheduling[J].Performance Evaluation Review, 2023, 51(1):21-22.
- [3] Pereira D .IOA ECG webinar - An assessment of the effects of noise and vibration from onshore ground investigations and load testing on seahorses in the Blackwater Estuary[J].Acoustics bulletin, 2023, 49(4):19-20.
- [4] Fu K , Chen H F , Zhao W .Distributed system identification for linear stochastic systems with binary sensors[J].Automatica, 2022(141-):141.
- [5] Shi X , Xu X , Cao J .A collective neurodynamic approach for solving distributed system optimum dynamic traffic assignment problems[J].Neurocomputing, 2022(Apr.28):483.
- [6] Rajeh W. Hadoop Distributed File System Security Challenges and Examination of Unauthorized Access Issue[J]. Journal of Information Security, 2022, 13(2):23-42.