Journal of Electronic Research and Application, 2025, Volume 9, Issue 6
http://ojs.bbwpublisher.com/index.php/JERA
ISSN Online: 2208-3510
ISSN Print: 2208-3502

Research on the Principle Comparison and Comprehensive Application of High-Availability Clusters and Load-Balancing Clusters

Yue Yang*

Inner Mongolia Technical College of Mechanics and Electrics, Hohhot 010010, Inner Mongolia, China

**Author to whom correspondence should be addressed.*

Abstract: In modern distributed systems and cloud computing architectures, high availability and high scalability are core requirements to ensure the continuous and stable operation of services. As key technologies for achieving these two goals, high-availability clusters and load-balancing clusters have significant differences in their design concepts and application scenarios, while also maintaining close connections. This paper aims to conduct an in-depth analysis of the core objectives, working principles, technical advantages and disadvantages, and typical application cases of high-availability clusters and load-balancing clusters. By introducing an analogical model of a "restaurant kitchen," the differences between the two are intuitively explained, and their technical characteristics are compared in detail. Additionally, a detailed practical case is included to specifically demonstrate the collaborative work of high-availability and load-balancing technologies through the construction process of Keepalived and HAProxy. Finally, taking the architecture of a typical e-commerce website as an example, this paper demonstrates the best practice of organically combining the two cluster technologies in a production environment to build a robust and high-performance distributed system. Research shows that understanding the differences between the two and implementing collaborative deployment is the cornerstone of designing modern IT infrastructure.

Keywords: High-availability cluster; Load-balancing cluster; Failover; Distributed system; Architecture design; Performance optimization

Online publication: December 16, 2025

1. Introduction

With the acceleration of enterprise digital transformation, users have put forward extreme requirements for the availability and performance experience of online services. Any service interruption or performance fluctuation may lead to significant economic losses and reputational risks. Against this background, cluster technology, which integrates multiple computing resources to collaborate in providing services, has become an inevitable choice to meet these demands. Among them, high-availability clusters are mainly committed to solving the problem

of service continuity, while load-balancing clusters focus on addressing issues related to service scalability and performance.

Although the two are often mentioned together, beginners and even some architects tend to confuse their fundamental purposes and application boundaries. This paper aims to systematically sort out and compare these two cluster construction methods, clarify their respective technical paradigms, advantages, disadvantages, and applicable scenarios, and provide clear theoretical guidance and practical reference for the architectural design of related systems [1,2].

2. Core principles of high-availability clusters and load-balancing clusters

2.1. High-availability cluster

The core goal of a high-availability cluster is to maximize the system's service uptime, typically achieved through redundancy and automatic failover mechanisms. It adopts an "active-standby" mode to ensure business continuity by eliminating single points of failure. Its essence lies in "redundant backup and active switching."

Nodes in the cluster continuously monitor each other through a dedicated network (heartbeat line). When the active node fails, the standby node detects the loss of the heartbeat signal and immediately triggers the failover process, taking over the identity of the active node (such as IP address, storage resources, and applications) to restore services without user awareness.

The active-standby mode is the most common implementation. Clients do not directly access real physical servers but instead connect to a virtual IP address (VIP), which "drifts" to the standby node when the active node fails.

2.2. Load-balancing cluster

The core goal of a load-balancing cluster is to distribute workloads, thereby improving the system's overall processing capacity and resource utilization. It adopts a "task distribution" mode to enhance throughput through parallel processing. Its essence is "division of labor and collaborative parallel processing."

One or more load-balancing schedulers are deployed at the front end as traffic entry points. Based on preset algorithms (such as round-robin, least connections, hashing, etc.), the scheduler reasonably distributes incoming user requests to multiple backend server nodes with identical functions.

All backend nodes are in an active state and share the workload collectively. The load balancer itself requires a high-availability mechanism (as described later) to avoid becoming a new single point of failure [3–5].

3. Comparative analysis: Multi-dimensional differential review

To more clearly illustrate the differences between the two, this paper conducts a comparative analysis from multiple dimensions and introduces an analogical model.

3.1. Analogical model: Restaurant kitchen

A high-availability cluster is analogous to a head chef and a sous chef in a restaurant kitchen. The head chef is responsible for cooking, while the sous chef stands by. If the head chef encounters an emergency, the sous chef immediately takes over to ensure uninterrupted dish preparation. Its goal is to maintain continuous service, though the sous chef may remain underutilized most of the time.

A load-balancing cluster is analogous to multiple parallel production lines in a restaurant kitchen. Orders are assigned to multiple chefs for simultaneous cooking to improve dish output efficiency and serve more customers. Its goal is to enhance overall efficiency, with all resources fully engaged in production.

3.2. Detailed comparison table

See **Table 1** below.

Table 1. Comparison of high-availability clusters and load-balancing clusters

Comparison dimensions	High-availability cluster	Load-balancing cluster
Core objectives	Ensure business continuity and reduce downtime	Improve processing capacity and throughput, and reduce latency
Core problems solved	Single point of failure	Performance bottlenecks and concurrent pressure
Resource utilization	Low (especially in active-standby mode, standby resources remain idle)	High (all nodes work in parallel)
Performance improvement	Does not directly improve performance, only ensures service availability	Directly and linearly improves system performance
Key technical challenges	Sensitivity and accuracy of fault detection	Session persistence, data consistency, and high availability of the load balancer itself
Typical technologies/ products	Pacemaker/Corosync, Keepalived, Windows Failover Cluster	Nginx, HAProxy, LVS, F5 Big-IP, Cloud Load Balancers
Optimal application scenarios	Stateful services or services with high consistency requirements, such as databases, authentication servers, and critical business applications	High-concurrency access services such as web servers, API services, and stateless microservices

4. Collaboration in practice: Building robust and high-performance architectures

High availability and load balancing are not mutually exclusive options; in modern complex systems, they are often complementary and work in synergy. To specifically demonstrate this point, this chapter provides a complete practical case showing how to combine the high-availability software Keepalived with the load-balancing software HAProxy to build a highly available load-balancing entry point [6].

4.1. Practical case: Building a highly available HAProxy load-balancing cluster

The goal of this case is to deploy HAProxy on two servers (lb01 and lb02) to achieve load balancing, while using Keepalived to provide high availability for these two HAProxy servers themselves, preventing the load balancer from becoming a single point of failure.

Environment preparation:

lb01: IP address 192.168.1.10

lb02: IP address 192.168.1.11

Virtual IP (VIP): 192.168.1.100 (managed by Keepalived, providing an external service entry)

Backend Web servers: 192.168.1.20, 192.168.1.21 (load-balanced by HAProxy)

4.1.1. Load-balancing layer: Installation and configuration of HAProxy (executed on lb01 and lb02)

(1) Install HAProxy (taking CentOS/RHEL as an example):

bash

```
sudo yum install -y haproxy
```

(2) Configure HAProxy: Edit the configuration file /etc/haproxy/haproxy.cfg.

bash

```
# Global configuration
global
    daemon
    log 127.0.0.1 local2
    maxconn 4000
# Default configuration
defaults
    mode http
    log global
    option httplog
    option dontlognull
    timeout connect 5000ms
    timeout client 50000ms
    timeout server 50000ms
# Frontend configuration: Define the listening port for external services and ACL rules (optional)
frontend http_front
    bind *:80
    stats uri /haproxy?stats  # HAProxy statistics page
    default_backend http_back
# Backend configuration: Define the real server pool and load-balancing algorithm
backend http_back
    balance roundrobin  # Use round-robin algorithm
    server web1 192.168.1.20:80 check  # ‘check’ enables health checks
    server web2 192.168.1.21:80 check
```

(3) Start and enable HAProxy:

bash

```
sudo systemctl start haproxy
sudo systemctl enable haproxy
```

At this point, both servers have load-balancing capabilities, but they are independent single points. The next step is to achieve high availability for themselves.

4.1.2. High-availability layer: Installation and configuration of Keepalived implements VIP failover through the VRRP protocol

(1) Install Keepalived (executed on lb01 and lb02):

bash
sudo yum install -y keepalived
(2) Configure Keepalived:
On lb01 (Master), create the configuration file /etc/keepalived/keepalived.conf:
bash
```
global_defs {
   router_id LVS_DEVEL  # Router identifier, unique for each node
}
# Define a script to check the HAProxy process
vrrp_script chk_haproxy {
   script "/usr/bin/killall -0 haproxy"  # Check if the haproxy process exists
   interval 2  # Check every 2 seconds
   weight 2    # If the check fails, reduce priority by 2
}
vrrp_instance VI_1 {
   state MASTER  # Initial state is MASTER
   interface eth0  # Network interface bound to VRRP advertisements
   virtual_router_id 51  # Virtual router ID, must be the same for the same cluster group
   priority 101  # Priority, MASTER should be higher than BACKUP
   advert_int 1  # Advertisement interval (seconds)
   authentication {
      auth_type PASS
      auth_pass 1111  # Authentication password, consistent across all nodes in the cluster
   }
   track_script {
      chk_haproxy  # Call the check script defined above
   }
   virtual_ipaddress {
      192.168.1.100/24  # Defined virtual IP (VIP)
   }
}
```
On lb02 (Backup), the configuration file is basically the same, but the state and priority are different:
bash
```
global_defs {
   router_id LVS_DEVEL02  # Different from Master
}
vrrp_script chk_haproxy {
   script "/usr/bin/killall -0 haproxy"
   interval 2
   weight 2
}
```

```
vrrp_instance VI_1 {
    state BACKUP  # Initial state is BACKUP
    interface eth0
    virtual_router_id 51  # Must be the same as Master
    priority 100  # Priority lower than Master
    advert_int 1
    authentication {
        auth_type PASS
        auth_pass 1111
    }
    track_script {
        chk_haproxy
    }
    virtual_ipaddress {
        192.168.1.100/24
    }
}
```

(3) Start and enable Keepalived (executed on lb01 and lb02):

```
bash
sudo systemctl start keepalived
sudo systemctl enable keepalived
```

4.2. Case verification and demonstration

High availability verification:

In the initial state, the VIP 192.168.1.100 resides on lb01 (Master).

When the client continuously accesses http://192.168.1.100, the service operates normally.

Simulate a failure: Manually stop the HAProxy service on lb01 (systemctl stop haproxy) or shut down the lb01 server directly.

Observation results: Keepalived's chk_haproxy script detects the disappearance of the HAProxy process. The priority of lb01 decreases (101-2=99), which is lower than that of lb02 (100). After a timeout, lb02 (Backup) takes over the VIP. Client access experiences only a brief interruption (usually 1–3 seconds) before recovering, achieving high availability [7,8].

Load balancing verification:

Access http://192.168.1.100/haproxy?stats to view the HAProxy statistics page and confirm that requests are distributed to the backend web1 and web2 servers in a round-robin manner.

Case conclusion:

This case perfectly demonstrates the collaborative relationship between high-availability clusters and load-balancing clusters:

As the core of the load-balancing cluster, HAProxy solves the problem of distributing traffic to multiple backend servers, improving the system's throughput and scalability.

As the core of the high-availability cluster, Keepalived addresses the single point of failure of the HAProxy

load balancer itself, ensuring the continuous availability of the service entry point.

The combination of the two forms a unified entry point with both horizontal scalability and high reliability. This is precisely the synergistic effect of "1+1>2" [9,10].

High availability and load balancing are not mutually exclusive options; in modern complex systems, they are often complementary and work in synergy.

4.3. Extended case: E-commerce website layered architecture

(1) Load balancing layer (entry gateway), it undertakes all user traffic and serves as the first line of defense. Two Nginx servers are deployed on different physical machines or virtual machines. A high-availability cluster is built using Keepalived technology, bound to a virtual IP (VIP, e.g., 192.168.1.100) to provide external services. When one Nginx server fails, Keepalived automatically drifts the VIP to the other, achieving second-level switching, ensuring the entry point never goes down, and solving the single point of failure of the load balancer itself [11,12].

(2) Application service layer (business processing), it processes user requests in a stateless manner, enabling elastic scaling, and can deploy large-scale Tomcat application server clusters. The upper-layer Nginx load balancer distributes user requests (such as product browsing and order placement) to healthy backend Tomcat instances through strategies like round-robin or least connections. This layer focuses entirely on business logic processing. By increasing or decreasing the number of Tomcat instances, it can easily handle traffic peaks such as "Double 11," achieving horizontal scaling and high performance at the application layer.

(3) Data persistence layer (data storage), it ensures the reliability and consistency of core data. The MySQL database adopts a master-slave replication architecture: one master database is responsible for writing (e.g., order creation), and multiple slave databases are responsible for reading (e.g., product queries), realizing read-write separation. Meanwhile, combined with high-availability tools such as MHA (Master High Availability), when the master database fails, a slave database can be automatically promoted to the new master database. This ensures high availability at the database level, avoids full-site service interruption caused by a single point of failure in the data layer, and provides a solid guarantee for core business data [13–15].

In this architecture, high-availability technology ensures the reliability of key nodes (entry gateway and database), while load-balancing technology ensures the scalability of the business processing layer (application services). The two complement each other, jointly building a robust and high-performance distributed system.

5. Conclusion

High-availability clusters and load-balancing clusters are two technically distinct yet equally important solutions in distributed system architectures. The core value of high-availability clusters lies in ensuring survival—addressing node failures through redundancy and failover mechanisms. In contrast, the core value of load-balancing clusters lies in promoting development, improving system capacity and performance through distribution and parallel processing mechanisms.

Successful system architects must deeply understand the essential differences and inherent connections between the two. In practical planning and design, these two technologies should be flexibly applied or combined based on the business characteristics of different components (e.g., stateful or not, critical or not), performance requirements, and cost considerations. Treating high availability as the "security foundation" of the system and

load balancing as the “acceleration engine” for performance is an inevitable path to building modern, elastic, and scalable IT infrastructure.

Disclosure statement

The author declares no conflict of interest.

References

[1] Zhu WB, Kong Z, Kou WZ, et al., 2024, Design of Cluster Service System for Smart Elderly Care New Community Based on Docker Containers. China New Telecommunications, 26(2): 62–64.

[2] Wang CY, Zhuang Y, 2023, Load Balancing Algorithm for Multi-Job Clusters Based on SDN and Improved CSA Algorithm. Computer and Modernization, (11): 28–35.

[3] Yin J, 2025, Research on High Availability Performance Methods for Operator Systems. Microcomputer, (3): 13–15.

[4] Xiao YF, 2023, Research on Data Fault-Tolerance Technology Based on Erasure Codes in Cloud Storage, dissertation, University of Electronic Science and Technology of China.

[5] He YF, Lin N, 2023, Design and Implementation of High-Availability Server Cluster Architecture Based on Linux, Proceedings of the 31st National Academic Conference on Computer New Technology and Education, National University Computer Education Research Association.

[6] Zhang XR, 2023, Research on Container Optimization Scheduling Technology Based on Kubernetes, dissertation, Jiangnan University.

[7] Huang N, 2024, Cluster Network Management System of S1240. China Broadband, 20(8): 40–42.

[8] Huang Y, 2023, Research on Computing Adaptation and Load Balancing Based on CNN Model Segmentation in Mobile Edge Computing, dissertation, Beijing University of Posts and Telecommunications.

[9] Shi HS, 2025, Method, System, Device and Medium for Realizing Effective Isolation of Dual Networks in the Same Cluster: CN202111241557.0, CN113992683B.

[10] Shao CW, 2024, Research and Deployment Practice of High Availability Technology in OpenStack Cloud Platform, (12): 53–55.

[11] Chen J, 2025, Research on Multi-Location and Multi-Center System Architecture Based on MySQL. Mechanical & Electrical Engineering Technology, 54(3): 175–181.

[12] Wu LX, Li GT, Hu Q, et al., 2025, A Method and System for Improving High Availability of Multi-Node Applications in Big Data Clusters: CN201910423927.9, CN110134518B.

[13] Meng T, Guo P, Cheng J, 2025, Method, Device and Medium for Constructing MQTT Broker Cluster Based on Multi-Instance Microservices: CN202311531350.6, CN118827757A.

[14] Zhang XM, Yu ZJ, 2025, Design of Online Interactive Teaching System for Computer Majors Based on Cloud Platform. Wireless Internet Technology, (7): 73–76.

[15] Tuo L, 2024, Research on Intelligent Container Resource Scheduling Strategy Based on LSTM and Genetic Algorithm. Computer Science and Application, 14(12): 132–141.