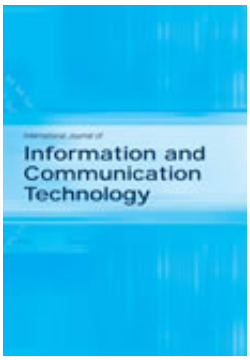
INDERSCIENCE PUBLISHERS
Linking academia, business and industry through research



International Journal of Information and Communication Technology

ISSN online: 1741-8070 - ISSN print: 1466-6642
https://www.inderscience.com/ijict

Virtual simulation of game scene based on communication load balancing algorithm

WenZhen Wang

Article History:

Received:	23 April 2024
Last revised:	22 July 2024
Accepted:	22 July 2024
Published online:	12 September 2024

Virtual simulation of game scene based on communication load balancing algorithm

WenZhen Wang

Animation Art Department,
Zibo Vocational Institute,
Zibo 255300, Shandong, China
Email: 10557@zbvc.edu.cn

Abstract: In order to improve the service effect of multi-player clustering server, this paper studies from the perspective of communication load balancing to improve the classification effect of game resources, and proposes a load balancing algorithm based on consistent hash by analysing the characteristics of high concurrency and interactivity of mobile real-time strategy games. This paper combines the communication load balancing algorithm to improve the work balance of the game servers, and performs virtual simulation of the game scenario to judge the balanced load state of the servers, so as to promote the stable operation of the game system. This paper verifies through experimental research that the virtual simulation method of game scene proposed in this paper based on communication load balancing type algorithm works well. This article effectively adjusts resource coordination during game scene simulation through load balancing algorithms, improves system efficiency, and ensures system stability and reliability.

Keywords: communication; load balancing; game scenarios; virtual simulation.

Reference to this paper should be made as follows: Wang, W. (2024) 'Virtual simulation of game scene based on communication load balancing algorithm', *Int. J. Information and Communication Technology*, Vol. 25, No. 6, pp.18–37.

Biographical notes: WenZhen Wang received the Master's degree from University of Science and Technology of China in 2018. He obtained the reputation of an Associate Professor from the Animation Art Department of Zibo Vocational College where he worked in. His research directions is digital media technology, virtual reality technology, and computer application technology.

1 Introduction

With the popularisation and development of computer and network, network battle games are becoming more and more popular, and the scale of its players is becoming larger and larger. An excellent network battle game, its simultaneous online players can be as high as hundreds of thousands or even millions. In front of the huge number of players, the design of game server is undoubtedly a huge challenge. Through the statistics of major

game developers and online game operators, we can see that using distributed server cluster to provide network services is a good scheme.

The imbalance of game users' demand for stored resources or the difference of processing speed caused by different computer hardware configurations for processing users' resource requirements may cause some computers responsible for processing network information interaction to be overloaded and even become 'hot spot' resources, but other computers have no information to process and are idle.

Traditional caching algorithms or only considering improving object hit rates result in low cache space utilisation; or only consider cache space utilisation, resulting in a low object hit rate. However, due to the large capacity of some objects and limited cache space of servers, coupled with limited network bandwidth resources, the utilisation of cache space has become increasingly important. So it is necessary to balance object hit rate and cache space utilisation, so that the server can respond quickly.

In order to improve the simulation effect of game scenes, this paper studies from the perspective of communication load balancing, and improves the classification effect of game resources and the stability of game running state through communication load balancing.

This article conducts research from the perspective of communication load balancing, aiming to improve the classification effect of game resources and enhance the stability of game operation status through communication load balancing BASED on the analysis of server requirements, the overall design of high concurrency distributed game servers was carried out, and experimental research verified that the communication load balancing algorithm proposed in this paper can play an important role in virtual simulation of game scenes,

This article effectively adjusts resource coordination during game scene simulation through load balancing algorithms, improves system efficiency, ensures system stability and reliability, and is of great significance for subsequent game scene construction, especially for large-scale game scene construction, providing a reliable foundation.

2 Related work

The idea of cluster architecture is to form a server group through a set of servers with identical functions, in which each server provides the same functions and services. There are one or more proxy servers in the cluster that synchronously manage and maintain the entire cluster. When the client attempts to connect to the cluster server, the proxy server will select the appropriate node for the client to connect based on the current health status of each host in the cluster (Havola et al., 2021).

Obviously, using cluster architecture as a game server development architecture has certain advantages. Clusters have to some extent solved the performance, scalability, and reliability issues of single server architecture (Gawel et al., 2022); clustering, as a simple server expansion method, is easy to deploy and cost-effective in practical applications; moreover, there is no need for too many cumbersome configuration documents and dependency relationships in expanding server nodes. All server nodes are managed uniformly by proxy nodes, which to some extent decouples the calling relationships between server nodes; the proxy server of the cluster serves as a communication intermediary node between the client and server nodes, which to some extent isolates the internal and external networks, making it impossible to directly access the server nodes in

the internal network from the external network, effectively ensuring the data security and operational security of the internal network (Wang et al., 2022). In large-scale cluster server architectures, complex data exchange and synchronisation between service nodes invisibly increase the workload of server developers in development and maintenance (Rojas Ferrer et al., 2020).

Distributed architecture is the latest design concept in game server development. Distributed architecture originated from the practice of web development, and common distributed frameworks include Dubbo, MapReduce, and CORBA. These frameworks require developers to split business functions into multiple modules, and modules can call each other through remote procedure calls (RPC). Distributed architecture effectively shields the complexity of communication between nodes for each node running on it, Reduced the difficulty for developers to maintain a single node (Salvini et al., 2022). The advantages of distributed architecture servers are very obvious, especially in terms of effective performance improvement. Firstly, each module of the distributed architecture plays its own role and communicates with each other through efficient network libraries. By allocating business pressure reasonably to each node, the distributed architecture can effectively improve the concurrency and communication performance of the server. Secondly, the distributed architecture has high scalability. By shielding the complexity of communication between nodes, the entire distributed architecture achieves cohesion and transparency. Therefore, developers can easily scale the distributed architecture horizontally without considering communication details, effectively improving development and maintenance efficiency (Cao et al., 2020). The reliability of distributed systems is also very high. Due to the system functions being divided into individual modules, and each functional module having master slave redundancy disaster recovery for polymorphic hosts, when one of the main nodes of the distributed server cluster crashes, a backup slave server can be immediately used to replace it. Therefore, the overall stability of distributed architecture is significantly higher than that of single machine single server architecture and cluster architecture. However, distributed architecture also has its drawbacks that cannot be ignored. The early design of distributed architecture is relatively complex and there are no fixed design rules. Therefore, when designing a game distributed server architecture, designers need to follow the CAP principle in theory. In addition, designers need to comprehensively consider the advantages and disadvantages of existing old architectures and the specific needs of specific businesses in practice, and define specific functional modules for specific businesses Expansion methods and communication interfaces (Mondragón Bernal et al., 2022).

Due to the fact that the logical processing encountered by the server itself is not too complex, its requirements mainly come from the performance aspect: firstly, it is a data compression problem. The connection between the network game server and the client mainly depends on the acceptance and push of communication protocols. In addition, the conversion of scenarios between servers also requires a large amount of protocol forwarding. Due to the strict performance requirements of game servers, when their code logic reaches the bottleneck period, The time consumed by the server for protocol parsing and sending is crucial. In addition, the size of data directly affects the bandwidth consumption during data transmission, which is also crucial for the operational cost of company projects. Traditional transmission formats such as XML and JSON are inefficient and not suitable (Neroni et al., 2021); the second reason is that each player in the game will have a large amount of complex and unordered storage data, and also need

to meet the short-term high concurrency reference scenarios, which greatly increases the difficulty of database design (Mystakidis et al., 2022); the third is the performance requirements for server-side monitoring. The server of online games needs to withstand high concurrency pressure, and its main task is to continuously monitor the system 24 hours a day. Based on the load status of various regional servers, it recommends, jumps, and stops registration of regional servers for users who log in, register, or even play games, in order to alleviate the response pressure of a single server. Even if you enter a single service area, you will face different game scenarios such as battles, shopping malls, and emails, which is a huge test for the server (Gabajová et al., 2021).

A new game often experiences explosive growth in registered users upon its launch, followed by fluctuations in game users due to various factors, ultimately leading to a decline period. In order to meet the needs of users, we increase the number of virtual game worlds by adding game servers. However, these virtual worlds are independent of each other, resulting in the same game map and content in each virtual world. Ultimately, game users in different virtual worlds cannot communicate with each other, and it is also difficult for a game user to switch between different virtual worlds, This affects the user experience (Khan et al., 2021). The fundamental reason for this phenomenon is that the scalability of the game server architecture cannot meet the constantly changing demands of the number of users for server performance. The game service architecture is essentially a distributed system, and achieving a highly available and scalable server architecture is a highly challenging task (Rahouti et al., 2021). If the scalability of the server is not strong enough to meet the constantly changing number of users in a virtual world, game operators can only adjust the number of virtual worlds based on the number of game users, which also puts high requirements on the company's operation and maintenance. There are many design difficulties in the game service architecture, partly due to distributed systems, and partly due to the business characteristics of the game itself, including high availability, scalability, high performance, fault tolerance, product server operation management, anti-cheating, anti-cheating, system security, etc. (Tao et al., 2021). Scalability is one of the main challenges, as it requires users to perceive the state of the entire virtual world while also dealing with a large number of concurrent online users. With the increasing number of game users, virtual elements, and game content in the game world, the amount of information that needs to be updated and sent also greatly increases, ultimately limited by network resources and computer processing power (Akman and Çakır, 2023). The current situation is that developers of system programming do not have training or experience in distributed computing or concurrent programming. Even years of game development technicians find it difficult to handle high concurrency and distributed game servers, and they are better at writing business modules. So it is meaningful to construct a scalable distributed server architecture that is as reliable as enterprise level software, while also meeting the requirements for communication and scalability (Peterson, 2023).

A service-oriented framework. This framework can define the entire game as a virtual world, rather than multiple independent virtual worlds. The work tasks of each server within the cluster group can be planned based on the basic functions of the service and specific game business functions, such as communication servers specifically responsible for communication, replica servers specifically responsible for dungeon battles, guild servers specifically responsible for union related functions, and game map scene map servers (Keil et al., 2021). This functional division improves the scalability of different overall services and provides convenience for future service upgrades and additions.

There are three classifications of server architecture based on different server group architectures: the first is server architecture with routing servers (Lorenzo-Alvarez et al., 2020); the second type is a server architecture without routing servers, and the third type is a world server that serves as the centre of the entire server group. All requests are forwarded through the central server in a star shaped structure, and specific servers only need to handle a single logical function (Wan et al., 2021). The third solution achieves clear division of services, each with specific functions, and the relationships between services are easily defined. Adding new services does not affect previous services. If a service is affected by changes in the number of game users, it can adapt to changes by dynamically increasing or decreasing the number of servers (Segura et al., 2020).

In order to improve the overall performance of the high concurrency server system and improve the speed of server response to requests, it is necessary to introduce caching technology. Generally, mobile game server systems use the least recently used (LRU) algorithm to implement data caching. Cache replacement is based on previous data access, and the principle is to prioritise replacing the data with the least number of visits during a certain period of time. The LRU algorithm has a high efficiency in processing frequently accessed data, but a low hit rate when dealing with occasionally accessed data. Traditional caching algorithms or only considering improving object hit rates result in low cache space utilisation; Or only consider cache space utilisation, resulting in a low object hit rate. However, due to the large capacity of some objects and limited cache space of servers, coupled with limited network bandwidth resources, the utilisation of cache space has become increasingly important. So it is necessary to balance object hit rate and cache space utilisation, so that the server can respond quickly.

3 Game communication load balancing

Congestion is a problem that occurs on a shared network when multiple users compete to access the same resources (bandwidth, buffers, and queues). In a data packet switching network, data packets are moved in and out of the buffer and queue of the switching device as they pass through the network. In fact, data packet switching networks are commonly referred to as ‘queue networks’. The characteristic of data packet switching networks is that data packets may arrive in groups from one or more sources. Buffers help routers absorb burst data packets until they can resolve them on their own. If the business traffic is too high, the buffer will be filled and new data packets will be discarded. Increasing the size of the buffer is not the solution, as an excessively large buffer can cause significant latency.

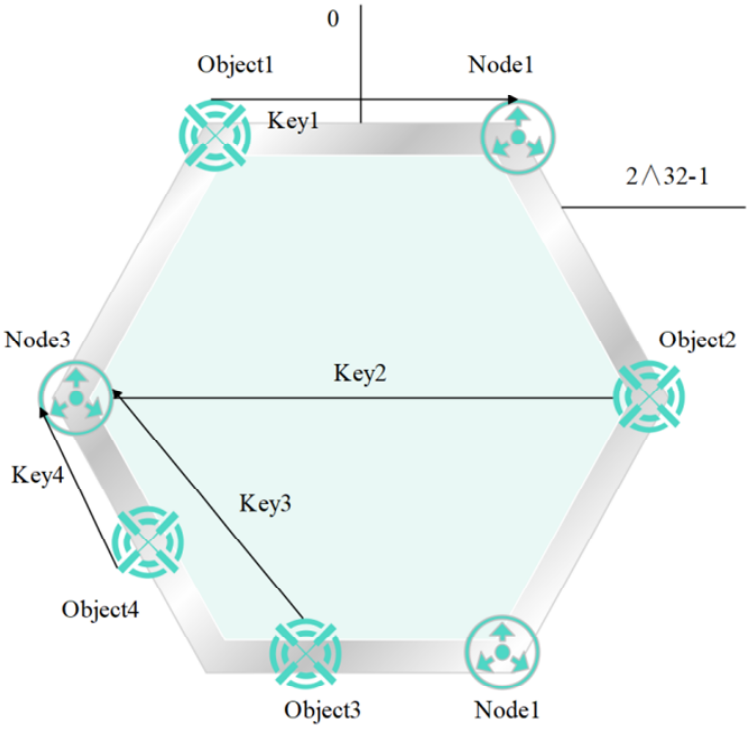
3.1 Optimisation of load balancing mechanism

By analysing the characteristics of high concurrency and interactivity of mobile real-time strategy games, a load balancing algorithm based on consistent hash is proposed. In order to ensure real-time performance, a cache replacement algorithm based on key value evaluation is proposed to achieve the balance between byte utilisation and object hit rate as much as possible, and the effect of the algorithm is verified by experiments.

Because of the portability of the mobile terminal, it can meet the needs of people playing games anytime and anywhere, but also easily lead to a surge in concurrency of the server system, and put forward higher requirements for the concurrency performance of the system, so it is necessary to optimise the load balancing mechanism.

According to the common Hash algorithm, the consistent Hash algorithm maps the key corresponding to nodes to a space with a size of 232 to form a closed ring, as shown in Figure 1.

Figure 1 Schematic diagram of consistent hash algorithm (see online version for colours)



If Node2 is deleted after failure, according to the principle of clockwise migration, object2 will be migrated to Node3, so that only the mapping position of object2 has changed, and other objects do not need to be changed, as shown in Figure 2. If a new node Node4 is added to the cluster, object3 is migrated to Node4, and the adjusted load balancing is shown in Figure 3.

Therefore, because the mapping space is circular, each object can always be assigned to a specific node. The consistency of consistent Hash algorithm is mainly reflected in monotonicity, and the general Hash algorithm will cause many objects to be unable to correspond to the original nodes. However, consistent Hash algorithm is very suitable for distributed clusters, avoiding a large number of data migration and reducing the waste of server resources.

Figure 2 Adjustment diagram after node deletion (see online version for colours)

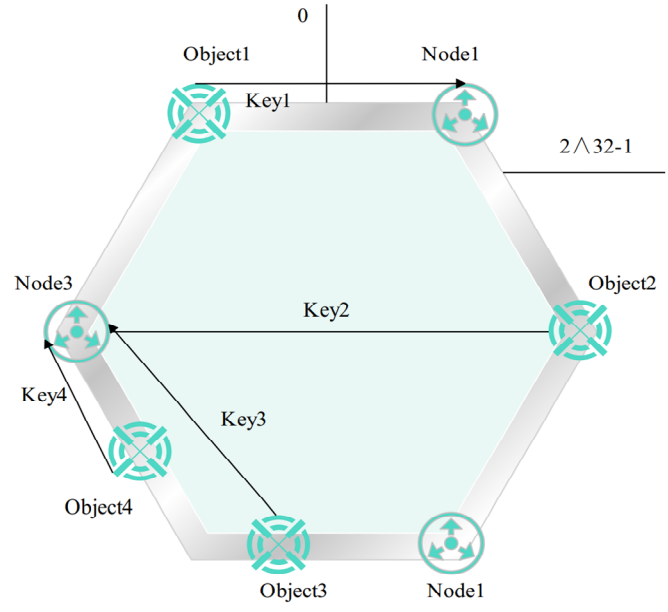
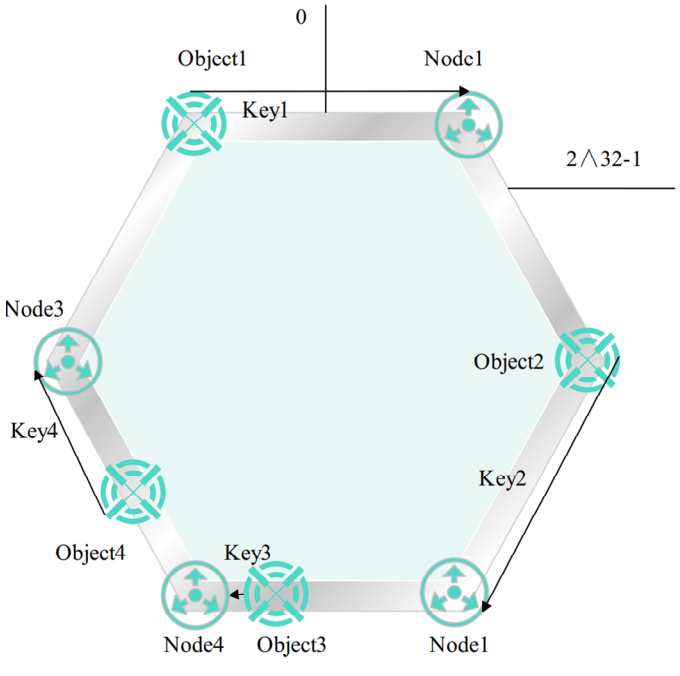


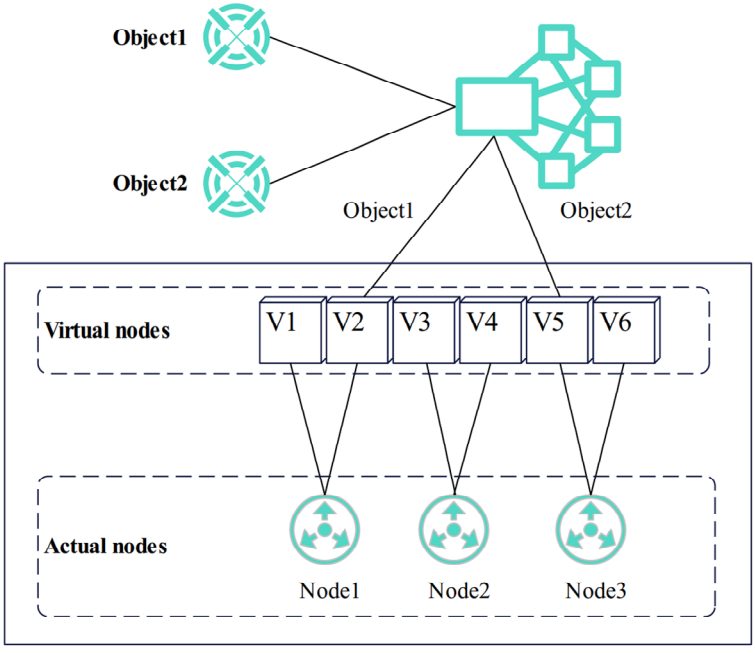
Figure 3 Adjustment diagram after node addition (see online version for colours)



In Figure 4, the virtual nodes of the Node1 physical node are (v1, v2), the virtual nodes of the Node2 physical node are (v3, v4), and the virtual nodes of the Node3 physical node are (v5, v6). Object1 maps to the v5 virtual node, so Object1 object data is allocated to

the Node3 physical node. Object2 maps to the v2 virtual node, so Object2 object data is allocated to the Node1 physical node.

Figure 4 Schematic diagram of virtual node (see online version for colours)

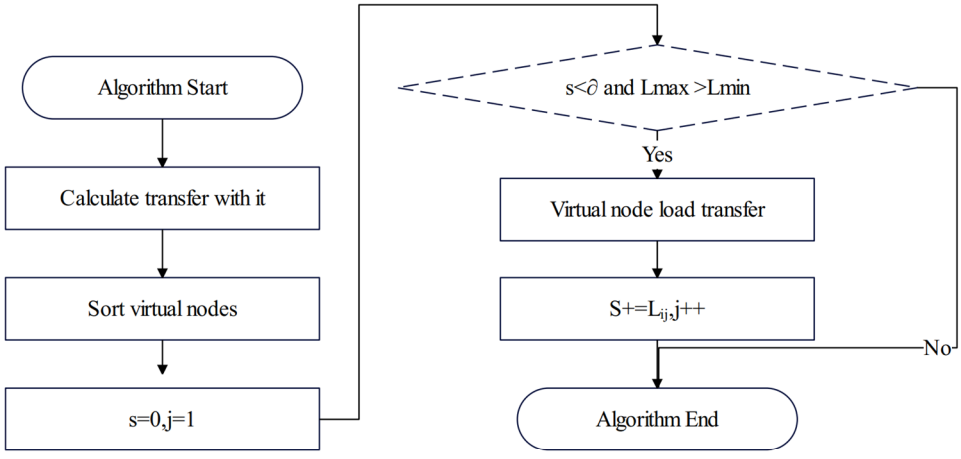


Considering that the performance of the server is mainly determined by the comprehensive conditions such as CPU, memory and bandwidth, r_i represents the number of resources occupied by the actual node i. If we assume that there are N actual nodes and M virtual nodes in the cluster, the total number of resources in the cluster is $R=\sum_{i=1}^{N} r_i$.

The weight of actual node i is $w_i = \frac{r_i}{R}$, and the number of corresponding virtual nodes of this node is $v_i = \frac{r_i}{R} \cdot M$ (Egea-Vivancos et al., 2021).

As the number of requests increases, the load profile of each node varies, with some nodes having high load pressure while others have low load pressure. Therefore, dynamic adjustment of load is required considering that the weights of each node are set differently. The load factor L_i represents the percentage of full load occupied by the actual node i, L_{avg} represents the average load factor of all servers, the load of the node with the largest real-time load factor among all servers is L_{max}, and the load of the node with the smallest real-time load factor among all servers is L_{min}. When $L_{max}–L_{min} > \theta$ (where θ represents the load difference threshold and is an adjustable factor), dynamic adjustment is required as shown in Figure 5.

Figure 5 Flowchart of load dynamic adjustment



The system calculates L_{avg}, L_{max} and L_{min} at regular intervals, and performs dynamic adjustment when it judges L_{max}–$L_{min} > \theta$. First, the system calculates the transfer threshold $\partial = \max\{ L_{max}–L_{avg}, L_{avg}–L_{avg} \}$. If the actual node i with load factor L_{max} is assumed to have K virtual nodes, then $V_{i1}, V_{i1}, \ldots, V_{ik}$ represents the virtual nodes of the actual node i, respectively. After that, the system sorts all the virtual nodes corresponding to the actual node i according to the load rate from the largest to the smallest and initialises the counters $s = 0, j = 1$ (s represents the current transferred load value and j represents the current virtual node number). The system determines that when $s < \partial$ and $j < K$, $L_{max} > L_{min}$ is true, it transfers the load of virtual node V_{ij} of actual node i to the smallest virtual node of the actual node with load rate L_{min} and updates the counter value, otherwise it ends the scheduling.

3.2 Optimisation of data caching mechanisms

High latency can lead to communication delays between clients and servers. In response to this problem, this article starts from the perspective of load balancing technology, introduces virtual nodes on the basis of consistent hashing algorithm, and implements the algorithm through node weight allocation and dynamic load adjustment, and tests the efficiency of the algorithm; a cache replacement algorithm based on key value evaluation is proposed for caching technology, aiming to achieve a balance between byte utilisation and object hit rate, thereby improving system efficiency and reducing system latency.

In this paper, we propose a cache replacement algorithm based on key value evaluation, the basic idea is: calculate the key value evaluation value of each object, the larger the key value is, the more it needs to be saved in the cache. In order to ensure the balance between the number of object hits and object byte utilisation, a byte size threshold S_c is set when calculating the key value. In order to ensure the balance between object hit rate and byte hit rate, the object whose memory is larger than constant is logarithm, so as to increase the key value of the object and increase the probability of saving in the cache and improve the byte hit rate. Saving small objects in the cache does not improve the byte hit ratio relatively, which is also beneficial to ensure the object hit

ratio. The key value evaluation function of the object is designed as shown in equation (1).

$$K_i = \begin{cases} L + \dfrac{F_i \cdot C_i \cdot P_i}{\log(S_i)} - \dfrac{1}{T_i} & (S_i > S_c) \\ L + \dfrac{F_i \cdot C_i \cdot P_i}{S_i} & (S_i \le S_c) \end{cases} \tag{1}$$

Among them, K_i represents the key value of object i, L is the age factor, and the initial value is usually set to 0, F_i is the access frequency of object i over a period of time, C_i is the cost for the system to query object i, S_i is the size of object i, and S_c is set to a constant value, P_i is the number of recent accesses to object i, and T_i is the time of the most recent time that object i was accessed. For those recently accessed objects, the probability of being accessed in the next period of time is relatively high, which is in line with the characteristics of network request access. Thus, equation (1) increases the priority of those recently accessed objects by subtracting the reciprocal of the most recent access time, reducing the probability that they will be replaced out of the cache. In addition, in reality, there may be situations where a request is frequently accessed for several days, but after this period of frequent access, it will not be accessed again for a long time. If the cache server does not notice this phenomenon, the object may be stored in the cache for a long time, resulting in a waste of cache resources. In view of this situation, it is necessary to introduce recent visit times of P_i into equation (1)

In order to investigate the advantages and disadvantages of cache replacement algorithm, we need to analyse the following evaluation indexes: average object hit ratio index α, average byte utilisation index β and hit ratio index γ. For a limited capacity cache system, N requests received over a period of time are considered, where the variable m_i represents the size of the $i(1 \le i \le N)^{\text{th}}$ request object and the variable q_i represents whether the $i(1 \le i \le N)^{\text{th}}$ request object is in the current cache system. If the object exists in the current cache set, $q_i = 1$, otherwise $q_i = 0$. There are:

$$\alpha = \frac{\sum_{i=1}^{N} q_i}{N} \tag{2}$$

$$\beta = \frac{\sum_{i=1}^{N} q_i \cdot m_i}{\sum_{i=1}^{N} m_i} \tag{3}$$

$$\gamma = \frac{w_1 \cdot \alpha}{w_2 \cdot \beta} = \frac{\sum_{i=1}^{N} q_i}{N} \cdot \frac{\sum_{i=1}^{N} m_i}{\sum_{i=1}^{N} q_i \cdot m_i} \tag{4}$$

The hit ratio index γ reflects the strategy of cache replacement algorithm as a whole, where w_1 represents the weight of average object hit ratio and w_2 represents the weight of

average byte utilisation. From the result analysis, if the hit ratio index γ is large, it reflects that small objects request frequently in this time period. On the contrary, if the hit ratio index γ is small, it reflects that large object requests are more frequent in this time period.

Under the conditions of $w_1 = 1$ and $w_2 = 1$, the system selects the network request log file on the system line for a period of time as the request access data sample, and sets the capacity of the cache system to be limited and fixed. Experiments are carried out on LRU algorithm, first input first output (FIFO) algorithm and cache replacement algorithm based on key value evaluation proposed in this paper, and the results are shown in Figure 6 and Figure 7.

Figure 6 Comparison of object hit ratio of three cache replacement algorithms (see online version for colours)

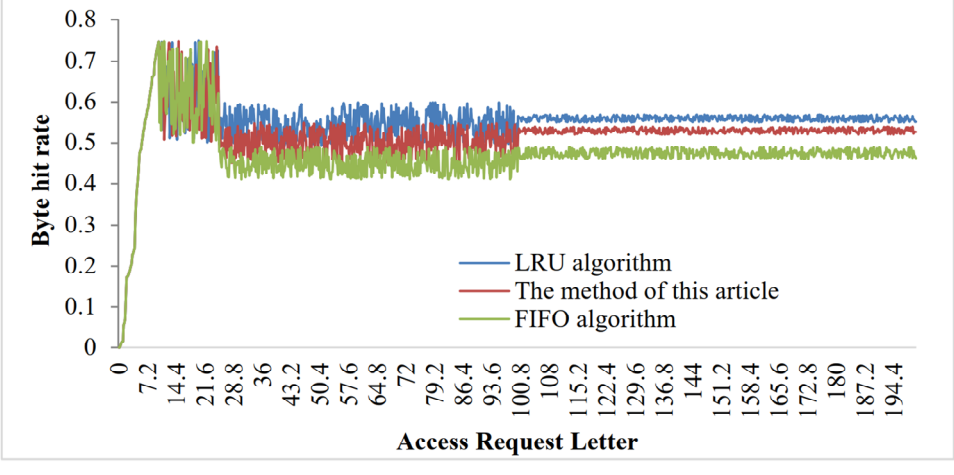


Figure 7 Comparison of byte utilisation of three cache replacement algorithms (see online version for colours)

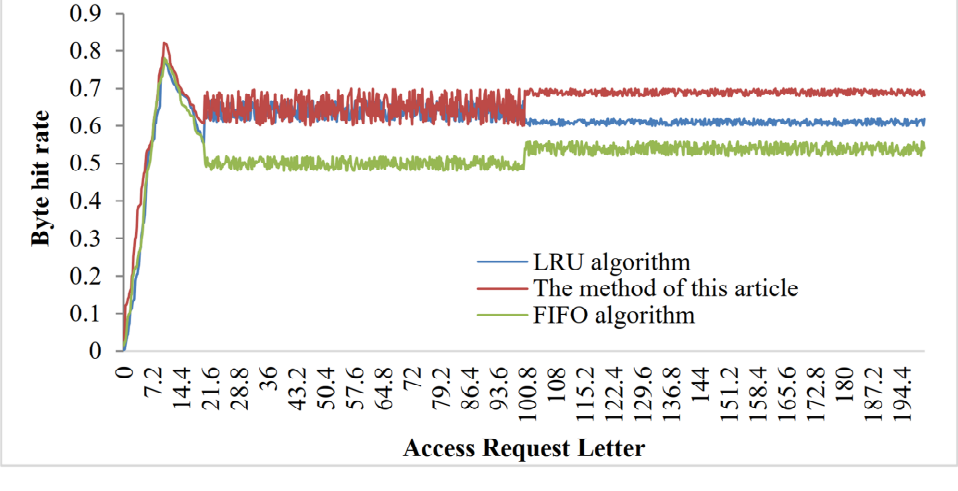
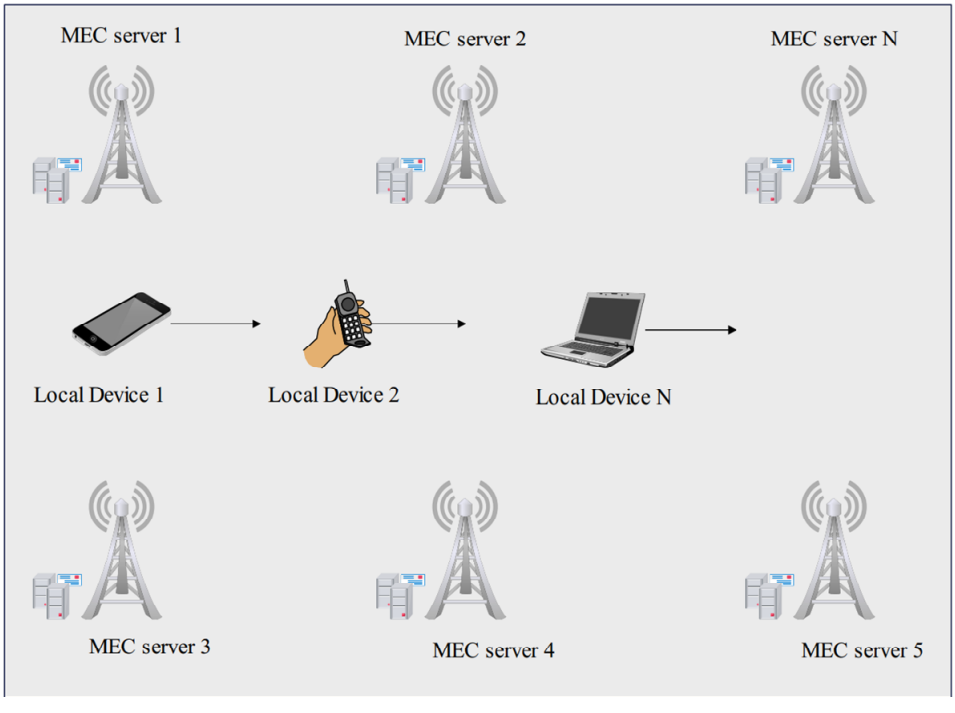


Figure 6 reflects the comparison of object hit ratios for the three cache replacement algorithms over time. It can be intuitively found from Figure 6 that LRU algorithm has the highest object hit rate, followed by the cache replacement algorithm based on key value evaluation proposed in this paper, and FIFO algorithm has the lowest object hit rate. The reason is that LRU algorithm not only considers the access time of objects, but also considers the recent access frequency of objects, and keeps the recently accessed objects in the cache as much as possible, which accords with the repetitive characteristics of web access. Furthermore, Figure 6 reflects the comparison of byte utilisation of the three cache replacement algorithms over time. As can be seen from Figure 7, the cache replacement algorithm based on key value evaluation has the highest byte utilisation, followed by LRU algorithm and FIFO algorithm has the lowest byte utilisation.

3.3 Computing offload strategy in multi-server scenario

The system model is shown in Figure 8. In a certain area, M base station (BS) are evenly distributed, the BS serial number is $m \in \{1, 2, \ldots, M\}$, and MEC servers are deployed beside each BS. The BS and the BS and the MEC server are connected through optical fibre links. There are N mobile devices (MD) randomly distributed in the area, such as mobile phones, tablets or notebook computers. The serial number of MD is $n \in \{1, 2, \ldots, N\}$, and the MD are connected with the BS through wireless links. Because it is a portable mobile device, each mobile device is moving dynamically, and at the same time, it will generate multiple tasks that need to be calculated. These tasks can be processed on the mobile device, that is, locally, or uploaded to MEC server for processing.

Figure 8 Moving edge calculation model diagram (see online version for colours)



If task $J_n^i(t)$ on mobile device n is processed locally that is, $a_n^i = 0$, the local computation delay for task $J_n^i(t)$ is defined as $T_n^{i,local}$. The local computation delay $T_n^{i,local}$. of task $J_n^i(t)$ is directly proportional to the computation amount C_n^i of task $J_n^i(t)$ and inversely proportional to the CPU computation power f_n^{local} of mobile device n which can be expressed as:

$$T_n^{i,local} = \frac{C_n^i}{f_n^{local}} \tag{5}$$

When the task $J_n^i(t)$ on the mobile device n is processed locally, it will consume the energy of the mobile device n, and the amount of energy consumption is not only related to the computation of the task itself, but also related to the chip architecture of the device itself. Local computing energy consumption is calculated using a recognised energy consumption statistical model, that is, $E = \varepsilon f^3 t$ where ε represents the chip architecture parameter of the device itself, f represents the computing power of the device CPU, and t represents the time at which the calculation is performed. The local computing energy consumption of task $J_n^i(t)$ is defined as $E_n^{i,local}$, and the local computing energy consumption $E_n^{i,local}$, of task $J_n^i(t)$ is proportional to the computing amount C_n^i of task $J_n^i(t)$ and the CPU computing capacity f_n^{local} of mobile device n, which can be expressed as:

$$E_n^{i,local} = \varepsilon_n \times \left(f_n^{local}\right)^2 \times C_n^i \tag{6}$$

If the task $J_n^i(t)$ on the mobile device n is unloaded to the MEC server m for processing, that is, $a_n^i = m$, the task needs to be uploaded to the MEC server m first, and the uploading delay of the task has a great relationship with the uploading speed. R_n^m is defined as the data upload speed of the mobile device n to the BS m. According to Shannon formula, the data upload speed R_n^m from the mobile device n to the BS m can be expressed as:

$$R_n^m = W_n^m \times lb\left(1 + \frac{P_n \times h_n^m}{\sigma^2}\right) \tag{7}$$

Among them, W_n^m represents the channel bandwidth from the mobile device n to the BS m, P_n represents the transmission power of the mobile device n and σ^2 represents the noise variance of the channel. h_n^m represents the channel gain of the mobile device n to the BS m, which adopts the Rayleigh channel model.

The upload delay caused by the task $J_n^i(t)$ uploading from the mobile device n to the BS m is defined as $T_{n,m}^{i,up}$, and the upload delay $T_{n,m}^{i,up}$, of the task $J_n^i(t)$ is directly proportional to the data amount D_n^i of the task $J_n^i(t)$ and inversely proportional to the data upload speed R_n^m of the mobile device n to the BS m, which can be expressed as:

$$T_{n,m}^{i,up} = \frac{D_n^i}{R_n^m} \tag{8}$$

When task $J_n^i(t)$ arrives at MEC server *m*, MEC server *m* first allocates its own resources to task $J_n^i(t)$, and then starts to execute the task. If the size of computing resources allocated by MEC server *m* for task $J_n^i(t)$ is $f_{n,m}^i$, and $T_{n,m}^{i,exe}$ is defined as the execution delay of task $J_n^i(t)$ in MEC server *m*, then the execution delay $T_{n,m}^{i,exe}$ of task $J_n^i(t)$ is directly proportional to the calculation amount C_n^i of task $J_n^i(t)$ and inversely proportional to the size $f_{n,m}^i$, of computing resources allocated by MEC server *m* for task $J_n^i(t)$, which can be expressed as:

$$T_{n,m}^{i,exe} = \frac{C_n^i}{f_{n,m}^i} \tag{9}$$

$T_n^{i,unload}$ defined as the edge computation delay of task $J_n^i(t)$. The edge computation delay $T_n^{i,unload}$ is equal to the sum of task upload delay $T_{n,m}^{i,up}$ and task execution delay $T_{n,m}^{i,exe}$, which can be expressed as:

$$T_n^{i,unload} = T_{n,m}^{i,up} + T_{n,m}^{i,exe} \tag{10}$$

At time *t*, if MEC server m has processed all the tasks assigned to it before time *t*, then the computational resources of MEC server m are all the computational power of MEC server m, that is, f_m^{server}. If MEC server m has not processed all the tasks assigned to it before time *t*, some of the tasks are still being processed waiting for the completion of the processing and transmission of the results back. At this time, the computational resources of MEC server m are not f_m^{server}. At time *t*, define the task queue of MEC server *m* as $Q_m(t)$, which denotes the sum of resources that the current MEC server *m* still needs to process. Specifically, $Q_m(t)$ is the queue of the amount of resources that have not been processed before moment *t*–1, that is, $Q_m(t–1)$, plus the amount of newly added task resources at moment *t*–1, and then subtracts the amount of task resources that have been completed during the time period $t–1+\tau$. The task queue recursion relation for MEC server *m* at time *t* can be expressed as follows:

$$Q_m(t) = Q_m(t-1) + Q_m^{add}(t-1) - Q_m^{done}(t-1) \tag{11}$$

Among them, $Q_m^{add}(t-1)$ is the amount of new task resources added by MEC server *m* at time *t*–1, and $Q_m^{done}(t-1)$ is the amount of task resources completed by MEC server m in time period $t–1+\tau$. This resource is the sum of the amount of task resources that can be accomplished by the new tasks generated at time *t*–1, and the amount of old task resources that can be accomplished in time period $t–1+\tau$. It can be expressed as:

$$Q_m^{add}(t-1)=\sum_{n=1}^{N}\sum_{i=1}^{K_{n-1}(t)} f_{n,m}^{i} \tag{12}$$

$$Q_m^{add}(t-1)=\sum_{n=1}^{N}\sum_{i=1}^{K_{n-1}(t)} O_n^i f_{n,m}^{i}+Q_m(t-2) \tag{13}$$

Among them, O_n^i is defined as a flag of whether task $J_n^i(t-1)$ is completed in a single time slot. If task $J_n^i(t-1)$ can be completed in a single time slot, then $O_n^i=1$, whereas if task $J_n^i(t)$ cannot be completed in a single time slot, then $O_n^i=0$ can be expressed as:

$$O_n^i=\begin{cases}1, & T_{n,m}^{i,exe}\le\tau\\ 0, & others\end{cases} \tag{13}$$

4 Server design and scene simulation

On the basis of consistent hashing algorithm, this paper introduces virtual nodes and addresses the problem of performance differences between nodes that traditional consistent hashing algorithms cannot solve. It sets weight allocation and realises dynamic load adjustment. Virtual nodes are copies of actual nodes in Hash space, where a physical node corresponds to several virtual nodes. After introducing a virtual node, it is necessary to first insert the hash value of the object into a virtual node middleware, and then map it to a physical node.

4.1 Game server design

Based on the analysis of server requirements, the overall design of high concurrency distributed game server is carried out. The overall architecture diagram of the server is shown in Figure 9. Based on the different development technologies, the architecture of the server is divided into network data transmission service layer, data access service layer, communication service layer, general component layer, load balancing component and game service layer from bottom to top.

4.2 Results

The experiment uses the average load rate as the indicator of load balancing. The load rate represents the percentage of actual nodes occupying full load, while the average load rate represents the average of all actual node load rates. Prepare a cluster of 8 servers for load balancing testing in an online environment. The total number of virtual nodes in the cluster is 80, with an average of 10 virtual nodes controlled by each physical node. Simulate sending 10,000 long connection requests to the test cluster simultaneously using another server program.

In this paper, the server is composed of multiple server groups. As shown in Figure 10, there are multiple service modules in each server group, including GateServer

node group, login service module, player matching service module, game scene service module, RabbitMQ server and DBServer.

Figure 9 Overall architecture diagram of high concurrency distributed game server (see online version for colours)

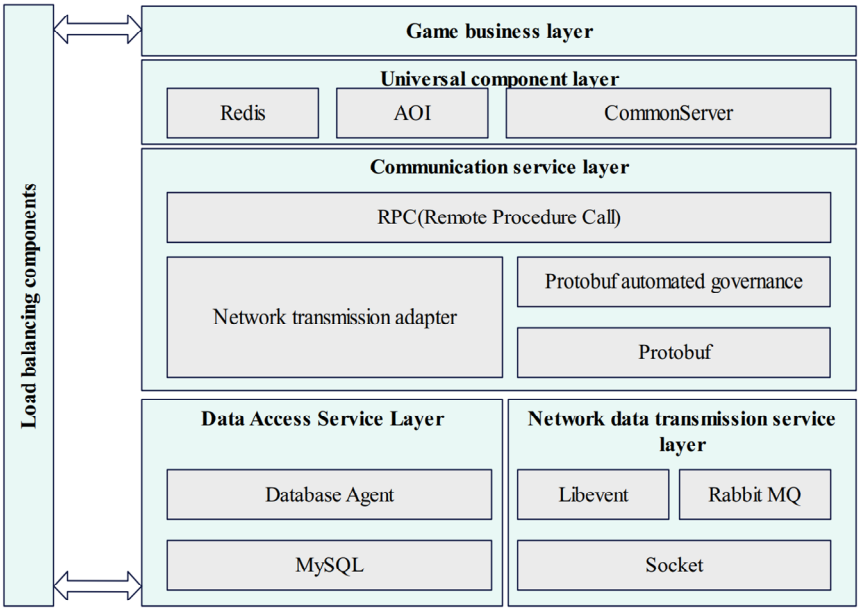


Figure 10 Topology diagram of high concurrency distributed game server (see online version for colours)

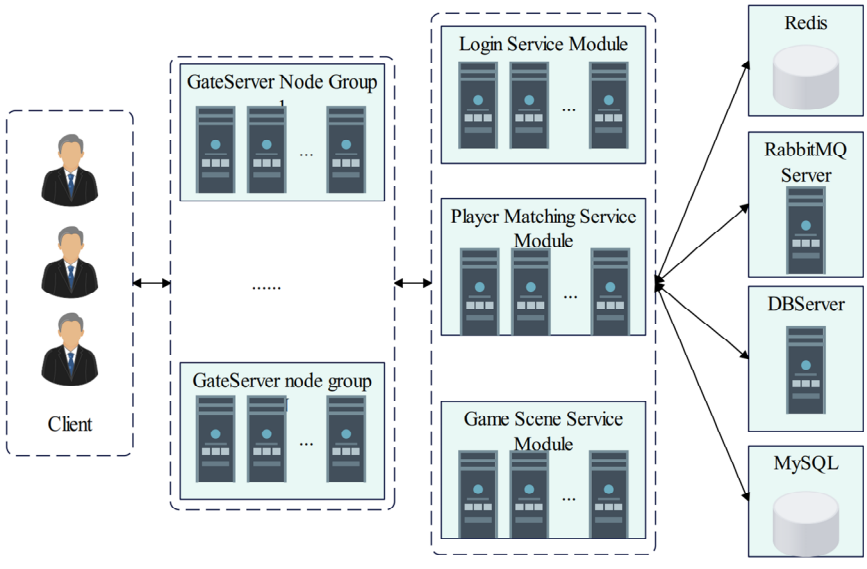


Figure 11 Schematic diagram of virtual simulation of game scene (see online version for colours)

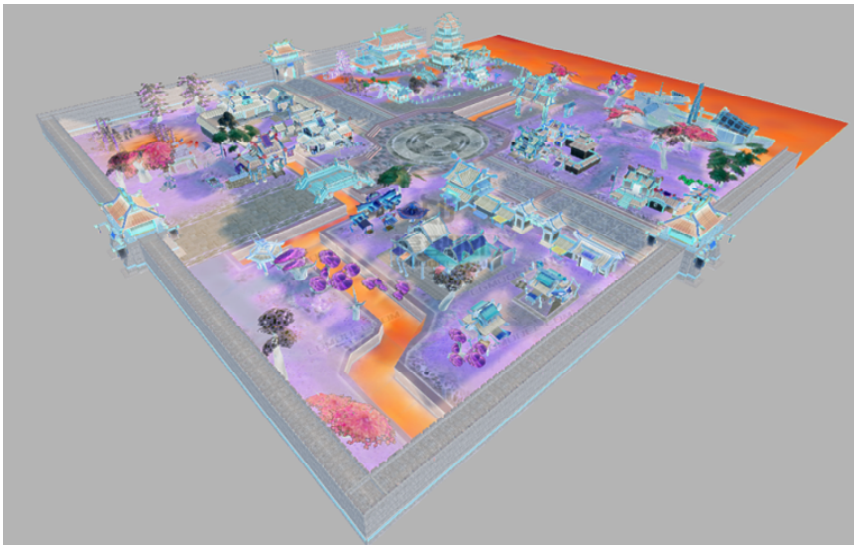


Figure 11 shows the application of the communication equalisation algorithm in the virtual simulation of the game scene, and the experimental research is carried out in the following.

In order to verify the accuracy of the load balancing algorithm, four servers are started, and the server configurations are not very different, and the four servers are using DELL 490 workstations, dual-channel Intel Quad-core Xeon E5405 CPUs, with a total of 8 cores, a main frequency of 2GHz, and a memory of 16GiB. Moreover, each server is initially assigned different levels and numbers of tasks, and each server calculates the load and uploads it to the load controller every 3s, which uses the predictive adaptive load balancing algorithm proposed in this paper to count the load redundancy of each server node. Start 4 clients, each client sends a task with different difficulty and execution time, the tasks are A, B, C, D level, the client cycle sends to ensure that the server uninterrupted processing. At this time, the load balancer is also in a busy state. At a randomly selected time, the loads in the text of the four server records are written into Table 1 at 2s intervals according to this moment.

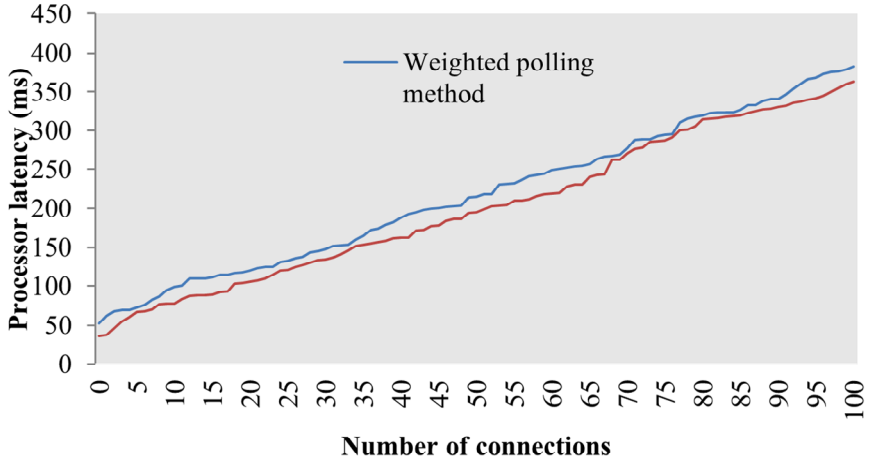
Table 1 Statistics of server load

Server A	*Server B*	*Server C*	*Server D*
44.804	363.853	121.422	202.576
80.921	212.948	152.126	172.256
111.847	141.986	132.058	121.846
152.025	121.877	121.331	162.469

The efficiency of the communication framework's network layer processing and the accuracy of the proposed load balancing algorithm based on the predictive adaptive mechanism are unilaterally tested through the previous aspects. In order to verify the overall efficiency of the communication framework and its practical application in the game server requirements, this communication framework is used for efficiency testing,

comparing the processing delay of the communication framework applying the weighted polling method and the algorithm in this paper, and the result records are plotted as a line graph in Figure 12:

Figure 12 Comparison of processing delay (see online version for colours)



4.3 Analysis and discussion

From the Figure 3, The test hardware is as follows: The server workstation is a dual Intel quad-core Xeon E5405 CPU with 8 cores, 2GHz main frequency and 16GiB memory. Test software: The operating system is CentOS 6.4.

From the Table 1, it can be seen that the predictive adaptive algorithm combining dynamic and static can even out the load imbalance as much as possible when there is a load imbalance in the server. By assigning tasks in a fuzzy predictive manner, it can make the server under pressure more evenly, and strongly avoids the situation where a node is assigned a large number of tasks per unit of time because of its better processing performance, resulting in a spike in load.

From the Figure 12, it can be seen that the processing delay of the communication framework adopting the weighted rotation method algorithm is slightly larger than that of the algorithm adopted in this paper. The reason for this is that it fails to average the node load of the game server group well, and when the server load increases and the number of concurrent requests increases, the processing efficiency will obviously decline, which leads to an increase in latency. However, the algorithm proposed in this paper takes a load balancing fuzzy prediction approach at the scheduling layer. Although the accuracy is slightly off, it can predict the load increase of each node more realistically by testing close to the real load consumption.

Through the above research, it is verified that the communication-based load balancing algorithm proposed in this paper can play an important role in virtual simulation of game scenes, promote the load balancing of servers and improve the running efficiency of the system.

This article conducts research from the perspective of communication load balancing, aiming to improve the classification effect of game resources and enhance the stability of

game operation status through communication load balancing Based on the analysis of server requirements, the overall design of high concurrency distributed game servers was carried out, and experimental research verified that the communication load balancing algorithm proposed in this paper can play an important role in virtual simulation of game scenes.

5 Conclusions

The overall performance of cluster architecture is determined by the upper limit of agent node performance, which leads to the upper limit of scale-out ability of cluster server architecture. That is to say, when the server-side architecture is extended to a certain extent, adding new service nodes will not have obvious performance improvement. In order to improve the simulation effect of game scene, this paper studies from the perspective of communication load balancing, and improves the classification effect of game resources and the stability of game running state through communication load balancing. Based on the analysis of server requirements, this paper designs a highly concurrent distributed game server, and verifies that the communication-based load balancing algorithm proposed in this paper can play an important role in virtual simulation of game scenes through experimental research.

Through the experimental study, it is verified that the communication-based load balancing algorithm proposed in this paper can play an important role in virtual simulation of game scenes, promote the load balancing of servers and improve the running efficiency of the system.

This article effectively adjusts resource coordination during game scene simulation through load balancing algorithms, improves system efficiency, ensures system stability and reliability, and is of great significance for subsequent game scene construction, especially for large-scale game scene construction, providing a reliable foundation.

The mobile real-time strategy game server system designed and implemented in this article has been applied to enterprise online projects. However, when facing more complex and ever-changing demand scenarios in the future, the system still needs to be further improved. The game server system may face the problem of cheating by cheats. In order to ensure the security of this system, game monitoring modules will continue to be developed in the future to prevent players from cheating.

References

Akman, E. and Çakır, R. (2023) 'The effect of educational virtual reality game on primary school students' achievement and engagement in mathematics', *Interactive Learning Environments*, Vol. 31, No. 3, pp.1467–1484.

Cao, S., Nandakumar, K., Babu, R. and Thompson, B. (2020) 'Game play in virtual reality driving simulation involving head-mounted display and comparison to desktop display', *Virtual Reality*, Vol. 24, No. 3, pp.503–513.

Egea-Vivancos, A. and Arias-Ferrer, L. (2021) 'Principles for the design of a history and heritage game based on the evaluation of immersive virtual reality video games', *E-learning and Digital Media*, Vol. 18, No. 4, pp.383–402.

Gabajová, G., Krajčovič, M., Matys, M., Furmannová, B. and Burganová, N. (2021) 'Designing virtual workplace using unity 3D game engine', *Acta Tecnología*, Vol. 7, No. 1, pp.35–39.

Gawel, A., Strykowski, S. and Madias, K. (2022) 'Implementing sustainability into virtual simulation games in business higher education', *Education Sciences*, Vol. 12, No. 9, pp.599–606.

Havola, S., Haavisto, E., Mäkinen, H., Engblom, J., and Koivisto, J. M. (2021) 'The effects of computer-based simulation game and virtual reality simulation in nursing students' self-evaluated clinical reasoning skills', *CIN: Computers, Informatics, Nursing*, Vol. 39, No. 11, pp.725–735.

Keil, J., Edler, D., Schmitt, T. and Dickmann, F. (2021) 'Creating immersive virtual environments based on open geospatial data and game engines', *KN-Journal of Cartography and Geographic Information*, Vol. 71, No. 1, pp.53–65.

Khan, N., Muhammad, K., Hussain, T., Nasir, M., Munsif, M., Imran, A.S. and Sajjad, M. (2021) 'An adaptive game-based learning strategy for children road safety education and practice in virtual space', *Sensors*, Vol. 21, No. 11, pp.3661–3668.

Lorenzo-Alvarez, R., Rudolphi-Solero, T., Ruiz-Gomez, M.J. and Sendra-Portero, F. (2020) 'Game-based learning in virtual worlds: a multiuser online game for medical undergraduate radiology education within second life', *Anatomical Sciences Education*, Vol. 13, No. 5, pp.602–617.

Mondragón Bernal, IF., Lozano-Ramírez, N.E., Puerto Cortés, J.M., Valdivia, S., Muñoz, R., Aragón, J., ... and Hernández, G. (2022) 'An immersive virtual reality training game for power substations evaluated in terms of usability and engagement', *Applied Sciences*, Vol. 12, No. 2 pp.711–720.

Mystakidis, S., Besharat, J., Papantzikos, G., Christopoulos, A., Stylios, C., Agorgianitis, S. and Tselentis, D. (2022) 'Design, development, and evaluation of a virtual reality serious game for school fire preparedness training', *Education Sciences*, Vol. 12, No. 4, pp.281–290.

Neroni, M.A., Oti, A. and Crilly, N. (2021) 'Virtual reality design-build-test games with physics simulation: opportunities for researching design cognition', *International Journal of Design Creativity and Innovation*, Vol. 9, No. 3, pp.139–173.

Peterson, M. (2023) 'Digital simulation games in CALL: a research review', *Computer Assisted Language Learning*, Vol. 36, Nos. 5–6, pp.943–967.

Rahouti, A., Lovreglio, R., Datoussaïd, S. and Descamps, T. (2021) 'Prototy** and validating a non-immersive virtual reality serious game for healthcare fire safety training', *Fire Technology*, Vol. 57, No. 6, pp.3041–3078.

Rojas Ferrer, C.D., Shishido, H., Kitahara, I. and Kameda, Y. (2020) 'Read-the-game: system for skill-based visual exploratory activity assessment with a full body virtual reality soccer simulation', *PloS One*, Vol. 15, No. 3, pp.e0230042–e0230052.

Salvini, G., Hofstede, G.J., Verdouw, C.N., Rijswijk, K. and Klerkx, L. (2022) 'Enhancing digital transformation towards virtual supply chains: a simulation game for Dutch floriculture', *Production Planning and Control*, Vol. 33, No. 13, pp.1252–1269.

Segura, R.J., del Pino, F.J., Ogáyar, CJ. and Rueda, A.J. (2020) 'VR-OCKS: a virtual reality game for learning the basic concepts of programming', *Computer Applications in Engineering Education*, Vol. 28, No. 1, pp.31–41.

Tao, G., Garrett, B., Taverner, T., Cordingley, E. and Sun, C. (2021) 'Immersive virtual reality health games: a narrative review of game design', *Journal of Neuro Engineering and Rehabilitation*, Vol. 18, No. 2, pp.1–21.

Wan, B., Wang, Q., Su, K., Dong, C., Song, W. and Pang, M. (2021) 'Measuring the impacts of virtual reality games on cognitive ability using EEG signals and game performance data', *IEEE Access*, Vol. 9, No. 2, pp.18326–18344.

Wang, J., Liang, H.N., Monteiro, D., Xu, W. and **ao, J. (2022) 'Real-time prediction of simulator sickness in virtual reality games', *IEEE Transactions on Games*, Vol. 15, No. 2, pp.252–261.