IRSTI 20.53.01

10.51889/2959-5894.2024.87.3.012

M. Zhanuzakov[1*], G. Balakayeva[1]

[1]Al-Farabi Kazakh National University, Almaty, Kazakhstan
[*]*e-mail: zhanmuha01@gmail.com*

A COMPARATIVE STUDY OF LOAD-BALANCING ALGORITHMS FOR RELIABILITY OF ENTERPRISE INFORMATION SYSTEMS

Abstract

Enterprise information systems are always prone to high loads during active sessions. This issue is becoming more and more actual with the increase of digital enterprise systems. The key to the solution is the use of load-balancing algorithms, which ensure system stability at high workloads. This work analyzes common load balancing algorithms and provides a comparison analysis. The paper also performs a complexity analysis of load balancing algorithms, differentiating between the static and dynamic algorithms. The results show that static algorithms are easy to implement, yet they are unable to adapt to different changes. Dynamic load balancing algorithms require deeper knowledge of the system, but at the same time providing higher efficiency in changing environments. This research paper also provides a recommendation for selecting a load-balancer for enterprise information system.

Keywords: load-balancing, algorithms, enterprise, information system, reliability, fault-tolerance

М. Жанузаков[1], Г.Балакаева[1]

[1]Әл-Фараби атындағы Қазақ ұлттық университеті, Алматы қ., Қазақстан

КӘСІПОРЫН АҚПАРАТТЫҚ ЖҮЙЕЛЕРІНІҢ СЕНІМДІЛІГІН ҚАМТАМАСЫЗ ЕТУГЕ АРНАЛҒАН ЖҮКТЕМЕНІ ТЕҢДЕСТІРУ АЛГОРИТМДЕРІН САЛЫСТЫРМАЛЫ ЗЕРТТЕУ

Аңдатпа

Кәсіпорынның ақпараттық жүйелері белсенді сессиялар кезінде әрқашан жоғары жүктемелерге ұшырайды. Бұл мәселе цифрлық кәсіпорын жүйелерінің масштабталуымен өзекті бола түсуде. Бұл мәселеге шешім - жоғары жүктеме кезінде жүйенің тұрақтылығын қамтамасыз ететін жүктемені теңестіру алгоритмдерін қолдану. Бұл мақала жүктемелерді теңдестірудің жалпы алгоритмдерін талдайды және салыстырмалы талдауды ұсынады. Сондай-ақ, жұмыста статикалық және динамикалық алгоритмдерді ажырата отырып, жүктемені теңдестіру алгоритмдерінің күрделілігіне талдау жасалады. Нәтижелер статикалық алгоритмдерді жүзеге асыру оңай екенін көрсетеді, бірақ олар әртүрлі өзгерістерге бейімделе алмайды. Динамикалық жүктемені теңестіру алгоритмдері өзгеретін ортада тиімділікті қамтамасыз ете алады, алайда жүйені тереңірек білуді талап етеді. Бұл зерттеу жұмысында сонымен қатар кәсіпорынның ақпараттық жүйесі үшін жүктеме теңгерімін таңдау бойынша ұсыныстар берілген.

Түйін сөздер: жүктемені теңдестіру, алгоритмдер, кәсіпорын, ақпараттық жүйе, сенімділік, ақауларға төзімділік

М. Жанузаков[1], Г.Балакаева[1]

[1]Казахский Национальный Университет имени аль-Фараби, Алматы, Казахстан

СРАВНИТЕЛЬНОЕ ИССЛЕДОВАНИЕ АЛГОРИТМОВ БАЛАНСИРОВКИ НАГРУЗКИ ДЛЯ ОБЕСПЕЧЕНИЯ НАДЕЖНОСТИ КОРПОРАТИВНЫХ ИНФОРМАЦИОННЫХ СИСТЕМ

Аннотация

Корпоративные информационные системы всегда подвержены высоким нагрузкам во время активных сессий. Эта проблема становится все более актуальной с ростом количества цифровых корпоративных систем. Ключом к решению является использование алгоритмов балансировки нагрузки, которые обеспечивают стабильность системы при высоких рабочих нагрузках. В данной работе анализируются распространенные алгоритмы балансировки нагрузки и проводится

сравнительный анализ. В статье также проводится анализ сложности алгоритмов балансировки нагрузки, в котором проводится различие между статическими и динамическими алгоритмами. Результаты показывают, что статические алгоритмы просты в реализации, но они не способны адаптироваться к различным изменениям. Алгоритмы динамической балансировки нагрузки требуют более глубокого знания системы, но при этом обеспечивая более высокую эффективность в изменяющихся средах. В этой исследовательской работе также даются рекомендации по выбору метода балансировки нагрузки для корпоративной информационной системы.

Ключевые слова: балансировка нагрузки, алгоритмы, предприятие, информационная система, надежность, отказоустойчивость

Main provisions

The idea of the article is to analyze and compare different load-balancing algorithms for enterprise information systems. The research uses complexity analysis to evaluate load-balancing algorithms` performance. It assesses their fault-tolerance capabilities and discusses difference between static and dynamic types of algorithms. Results of the study show that among dynamic algorithms, The Least Connection is considered most reliable, while among static algorithms, Round-Robin outstand other methods in terms of reliability.

Introduction

The current enterprise information systems are designed to feed many users and execute complicated data calculations that cannot be performed by a single machine. Nowadays, enterprise systems are distributed systems comprising several nodes, each node is considered as an independent computer. When the primary server is busy handling resourceful calculations, requests from other clients keep coming to that server and all these requests are put into the queue. The idea of the load balancing algorithm is to distribute these requests to other servers, so that the load on each server is near equal [1].

Load balancing algorithms can be divided into two main types: the operated media [2]. The goal of a static load balancing algorithm is uniformly spreading incoming requests to a pool of existing servers which is achieved by defining particular parameters or using fixed set of rules. In contrast, the dynamic load balancer is continuous and real-time, which monitors inbound traffic and adjusts the servers dynamically by determining their load value, for instance, CPU, memory or number of active connections.

In the article by Beniwal P. et al., authors compare load-balancing algorithms by different parameters such as overhead, response time etc. Another review done by Sidra Aslam et al., reviews different methods for load-balancing between 2004-2015 and its relevance in cloud computing. Similar research was carried out by M. Randles et al., where the authors investigate Honeybee Foraging Behaviour, Biased Random Sampling and Active Clustering methods in cloud computing.

In this article, we make a comparative review of the most popular load-balancing algorithms and make a complexity analysis of each. In the methods section, we outline the methods used to compare and analyze the algorithms. The results of the analysis are described in the results sections. The discussion part underlines the interpretations of the findings and suggests further scientific directions of the research.

Research methodology

The compared features are the time complexity of the algorithm for handling requests, support of fault-tolerance and type of the algorithm (static or dynamic). This section provides a list of common algorithms and methods that are chosen for analysis with a brief description for each one.

Round Robin

Round Robin is a simple load balancing method that distributes requests evenly among a set of servers. Each server takes turns handling the next request [3]. The distribution pattern in this case is static, meaning it does not change because of the status of the system or various conditions affecting

workload. After task assignment is decided, the process repeats indefinitely without any change that may come because of workload or resource availability.

Random

Load balancer will randomly generate numbers to pick servers from a pool for each incoming request.

Threshold Algorithm

Threshold Algorithm runs by having threshold values for each processor, monitoring their workload continuously, but dynamically registers thresholds for redistributing tasks when necessary [4]. The goal of this algorithm is averting from overload modes of the processors through enforcing limited load levels for each one, resulting in the highest possible usage of resources. If there is a condition when processors are above their predetermined threshold of load, then the tasks will be migrated to other processors which have a lower load to maintain balance. Periodic revisions of thresholds allow for adaptability towards new system or workload patterns that are constantly under evolving [5]. This dynamic nature lets Threshold Algorithm to adapt to the variations in workload, alterations in system capacities and processor's capacity differences among processors.

IP hash

The IP Hash Load Balancing algorithm is a method applied to route inbound requests or traffic from various sources onto numerous servers or resources having worked out the hash value of the source IP address of each one of them. The concept is to repeat the mapping process uniformly for each individual IP address to the distinct server to create a unique route for request from the same IP address either by configuring local area network (LAN) enable server or by primary Domain Name Service (DNS) server. This feature assists in preservation of session affinity which is also referred to as "sticky sessions", and such applications are mostly used in stateful applications or session that need to be maintained through more than one request.

Least Connections

The load balancing technique, the Least Connections algorithm, tries to convey the accessing requests during the incoming connection to a pool of servers as evenly as possible. It is achieved by directing any request to resources that are in idle mode connected to the server with the least number of connections among available ones. This stop helps to prevent any individual server to be eclipsed and offers the better use of the resources which are available. In case a new connection is made, the load balancer orders server checking of the connection counts. On receiving the request, the load balancer analyzes it and subsequently forwards it to the server having the lowest number of active connections.

Least Response Time

Least Response Time (LRT) algorithm is a static load balancing techniques but with a dynamic side that summarizes sending requests to experimental servers depending on the average time of their response [9]. This is done to make sure that requests get sent to the servers which are the best in speed of processing and thus takes less amount of time to be processed faster thus culminating in users' wait time decreasing and application performance as well.

Next section describes the results of the complexity analysis of these algorithms.

Results of the study

Round Robin

The overall performance of the Round Robin algorithm depends specifically on the quantity of requests and the quantity of servers [10, 11]. The following is the pseudo code of the algorithm:

```
servers = [1, 2, ..., N^th server]
pointer = 0
function handleRequest(request):
    server = servers[pointer]
    pointer = (pointer + 1) % len(servers)
    sendRequestToServer(request, server)
```

```
while True:
    request = receiveRequest()
    handleRequest(request)
```

The Initialization step consists of an array of servers and a pointer. It always proceeds by constant time, regardless of the number of servers. Therefore, it has constant O(1) complexity. The time complexity of handling a single request depends on two factors: being on an address that gets accessed for a constant amount of time O(1)); moving the pointer through adding 1 to and performing a modulo operation with the length of the servers list. Both of them are regarded as having constant time form O(1). Thus, the performance characteristic of processing one request is O(1). Because the complexity of handleRequest is O(1) so the time complexity of loop primarily depends on the count of request each time. If there is a loop including the N requests, the time complexity is O(N) for the main loop. Considering all factors, one can see that the Round Robin algorithm has an overall time complexity of O(N) where N is the total number of requests.

Random

The time complexity of the Random load balancing algorithm is constant (O(1)). This means the time it takes to select a server for a request doesn't depend on the number of servers in the pool. Pseudo code of the algorithm:

```
function choose_server(servers):
  random_index = generate_random_number(0, len(servers) - 1)
  return servers[random_index]
```

Threshold Algorithm

Threshold algorithm for load balancing in a programming language has a constant time complexity of (O(1)) while monitoring the load and comparing it to the thresholds. But, the complexity of actions (e.g., locating a useable resource for migrating) depends on its implementation. In the best case, where one would have a centralized resource list, the complexity could be O(1). While in practice there is a search-communication overhead introduced by more complex systems and distributed networks, this can still translate to a complexity of O(N) in the worst case. In other words, it relates to the time that is required to find the most effective resources. Pseudo code of the algorithm:

```
resources = list of available resources
overloadThreshold = predefined threshold value
underloadThreshold = predefined threshold value

while True:
    for each resource in resources:
        currentLoad = measureLoad(resource)

        if currentLoad > overloadThreshold:
            underloadedResource = findUnderloadedResource()
            if underloadedResource is not None:
                migrateTasks(resource, underloadedResource)

        if currentLoad < underloadThreshold:
            attractTasks(resource)
```

IP hash

```
function hash(ipAddress)
  result = 0
  for each byte in ipAddress:
    result = result XOR byte
  return result
```

```
function distributeRequest(clientIpAddress, servers)
  hashValue = hash(clientIpAddress)
  serverIndex = hashValue % number_of_servers
  return servers[serverIndex]
clientIp = "10.0.0.1"
servers = [server1, server2, server3]
chosenServer = distributeRequest(clientIp, servers)
```

The Hash function selection is a key part of which influences the effectiveness of this algorithm. While XOR function working on byte-level leads to time complexity of (O(n)), when a more robust substitute like SHA-256 is used, it gets translated into average O(1) time complexity though the inner operations are involved. The others, such as computing the hash value and the server selection, usually take constant time O(1), as they are relatively simple requirements. The algorithm by definition uses constant space (O(1)) for variables like the hash value and server location. However, the space complexity of the server list is what determines space consumption since the list structure will also require space to operate with. For instance, the array which stores server records require space complexity of O(n) (n - the number of server entities).

Least Connections

```
function get_connection_count(server):
  // retrieve the actual number of connections on the server
  return connection_count
function choose_server(servers):
  min_connections = 0
  chosen_server = null
  for server in servers:
    connection_count = get_connection_count(server)
    if connection_count < min_connections:
      min_connections = connection_count
      chosen_server = server
  return chosen_server
servers = [server1, server2, server3]
incoming_request = "Request data"
chosen_server = choose_server(servers)
```

The function get_connection_count could be implemented in a variety of ways, changing its complexity. Usually, iterating though servers and comparing connections counts is O(n) in the worst case (n is defined here as the number of servers). The algorithm uses a constant space (O(1)), as it is operating only with its temporary variables. The server list may carry its own space complexities based on the data structure applied (e.g. with an array the space complexity will be O(n)).

Least Response Time

```
function get_average_response_time(server):
  return average_response_time
function choose_server(servers):
  min_response_time = 0
  chosen_server = null
  for server in servers:
    response_time = get_average_response_time(server)
    if response_time < min_response_time:
      min_response_time = response_time
      chosen_server = server
  return chosen_server
```

The complexity of the process of iterating through servers and checking a response time is considered to be identical to the Least Connections algorithm, whose complexity is of the order of O(n) in worst-case scenarios. Space complexity of the algorithm is also similar to Least Connections.

Ensuring fault-tolerance

Several load balancing algorithms have fault-tolerance mechanisms designed to enable service availability even when failover occurrences happen among the individual servers. Health Checks based Round Robin has the capacity to not only remove faulty servers from the pool but also able to re-install them in the pool when their "health" becomes better. Least Connection with Failover will prefer machines having less connections but can be further improved to switch the load to a back-up server in case the primary server fails.

Table 1. Complexity analysis results

Algorithm	*Time complexity for single request*	*Time complexity for N requests*	*Space complexity*	*Fault - tolerant*	*Type*
Round robin	*O(1)*	*O(N)*	*O(n)*	*yes (with Health Checks implementation)*	*static*
Random	*O(1)*	*O(1)*	*O(n)*	*no*	*static*
Threshold Algorithm	*O(n)*	$O(N^2)$	*O(n)*	*no*	*dynamic*
IP hash	*O(n)*	$O(N^2)$	*O(n)*	*no*	*static*
Least Connections	*O(n)*	$O(N^2)$	*O(n)*	*yes*	*dynamic*
Least Response Time	*O(n)*	$O(N^2)$	*O(n)*	*no*	*dynamic*

Table 1 summarizes the results obtained during this study. Each algorithm is assessed by the following characteristics: time and space complexities, fault tolerance of the algorithm and type (static or dynamic).

Discussion

In this section, we discuss and try to choose a load-balancing algorithm(s) for enterprise systems that ensures both reliability and fault-tolerance of the system.

Static load-balancing algorithms are simple to implement and effective if the system has a predictable number of users [12]. In terms of speed static algorithms such as Random and Round-Robin are both effective, but only the Round-Robin is considered a fault-tolerant approach. Static algorithm utilizes randomness as its simplicity's basis. Even though the main objective is balanced distribution to every server in the long run, the existence of individual incapacities of the servers, periodic creation of temporary hotspots on specific servers, and absence of dynamic adaption to specific server health and workload are restrictions of these type of algorithms.

Dynamic load-balancing algorithms on the other hand are adaptive to changes in the number of active users [13]. The disadvantage of these algorithms is the difficulty of their implementation and complexity. The complexity of all algorithms is quadratic, thus, we have to choose the one with reliability and fault-tolerance. The Least Connection algorithm has an implemented fault-tolerance,

which ensures the system will continue to operate even with the occurrence of failure. This fully satisfies the requirement we have set for choosing a load-balancing algorithm.

Using this paper, readers might be interested in developing new load-balancing algorithms and comparing it with the existing ones to further improve effectiveness and reliability of information systems.

Conclusion

In conclusion, we have made a comparative study of load-balancing algorithms in the context of high loaded enterprise information systems. The importance of these algorithms is the reliability of servers during active sessions and fault-tolerance. We provide a comprehensive analysis of these algorithms in terms of efficiency and usage.

One criterion when comparing these kinds of algorithms is the type of algorithm. Load-balancing algorithms have two primary types: static and dynamic. Static load-balancing algorithms are effective when the system has a predictable number of requests during sessions. The advantage of these algorithms is its simplicity and efficiency. The disadvantage is the inefficiency during changes in the systems hardware or software. This issue can be handled using dynamic load-balancing algorithms. However, they are more difficult to utilize.

Each algorithm was assessed using time and space complexity analysis using the pseudocode. Time complexity of the algorithm is linear of most static load-balancing algorithms when handling N number of requests, and quadradic for dynamic ones.

Another attribute is the fault-tolerance capabilities of the algorithm. Only Round Robin and Least Connections algorithms provide mechanisms for fault handlings.

In summary, when choosing suitable algorithm for enterprise level systems, we recommend using either Round Robin or Least Connections depending on the system requirements. Round Robin is proven to be simple yet effective in a fixed environment, while Least Connections provide adaptability to changes.

References

[1] Beniwal P., Garg A. A comparative study of static and dynamic load balancing algorithms //International journal of advance research in computer science and management studies. – 2014. – Т. 2. – №. 12. – С. 1-7.

[2] S. Aslam and M. A. Shah, "Load balancing algorithms in cloud computing: A survey of modern techniques," 2015 National Software Engineering Conference (NSEC), Rawalpindi, Pakistan, 2015, pp. 30-35, doi: 10.1109/NSEC.2015.7396341.

*[3] Pradhan, Pandaba, Behera, Prafulla Ku., Ray, B.N.B. "Modified Round Robin Algorithm for Resource Allocation in Cloud Computing." *Procedia Computer Science*, vol. 85, 2016, pp. 878-890. ISSN: 1877-0509, https://doi.org/10.1016/j.procs.2016.05.278.*

[4] S. Chowdhury and A. Katangur, "Threshold Based Load Balancing Algorithm in Cloud Computing," 2022 IEEE International Conference on Joint Cloud Computing (JCC), Fremont, CA, USA, 2022, pp. 23-28, doi: 10.1109/JCC56315.2022.00011.

[5] Bura, Deepa, Meeta Singh, and Poonam Nandal. "Analysis and Development of Load Balancing Algorithms in Cloud Computing." In Research Anthology on Architectures, Frameworks, and Integration Strategies for Distributed and Cloud Computing, edited by Information Resources Management Association, 1177-1197. Hershey, PA: IGI Global, 2021. https://doi.org/10.4018/978-1-7998-5339-8.ch056

[6] Ghomi E. J., Rahmani A. M., Qader N. N. Load-balancing algorithms in cloud computing: A survey //Journal of Network and Computer Applications. – 2017. – Т. 88. – С. 50-71.

[7] Vashistha J., Jayswal A. K. Comparative study of load balancing algorithms //IOSR Journal of Engineering. – 2013. – Т. 3. – №. 3. – С. 45-50.

[8] Radojević B., Žagar M. Analysis of issues with load balancing algorithms in hosted (cloud) environments //2011 Proceedings of the 34th international convention MIPRO. – IEEE, 2011. – С. 416-420.

[9] Xu M., Tian W., Buyya R. A survey on load balancing algorithms for virtual machines placement in cloud computing //Concurrency and Computation: Practice and Experience. – 2017. – Т. 29. – №. 12. – С. e4123.

[10] Saumendu Roy, Dr. Md. Alam Hossain, Sujit Kumar Sen, Nazmul Hossain, and Md. Rashid Al Asif. 2019. "Measuring the Performance on Load Balancing Algorithms". Global Journal of Computer Science and Technology 19 (B1):41-49.

[11] E. Gures, I. Shayea, M. Ergen, M. H. Azmi and A. A. El-Saleh, "Machine Learning-Based Load Balancing Algorithms in Future Heterogeneous Networks: A Survey," in IEEE Access, vol. 10, pp. 37689-37717, 2022, doi: 10.1109/ACCESS.2022.3161511.

[12] Mbarek, Fatma, Volodymyr Mosorov. "Load Balancing Based on Optimization Algorithms: An Overview." Institute of Applied Computer Science at the Faculty of Electrical, Electronic, Computer and Control Engineering, Lodz University of Technology, Lodz, Poland. 2019. https://doi.org/10.26636/jtit.2019.131819.

[13] M. H. Kashani and E. Mahdipour, "Load Balancing Algorithms in Fog Computing," in IEEE Transactions on Services Computing, vol. 16, no. 2, pp. 1505-1521, 1 March-April 2023, doi: 10.1109/TSC.2022.3174475.