Cloud Load Balancing Algorithms Performance Evaluation Using a Unified Testing Platform

Bingyan Li, Weiliang Liu, Salim Nader, Jiaqi Song, Chengyu Zhang, Michal Aibin
Khoury College of Computer Sciences at Northeastern University, Vancouver, Canada
{li.bingy, liu.weilia, nader.s, song.jiaqi1, zhang.chengyu1, m.aibin}@northeastern.edu

***Abstract*—Due to its improved response time, availability, and efficiency, load balancing emerged as an essential framework for designing high-performance distributed computing systems. This paper introduces a unified testing platform that objectively compares load balancing algorithms and measures their performance. It employs various request patterns and load types to simulate real-world conditions. We evaluate a selection of static and dynamic algorithms on throughput, response time, and failure rate metrics. The results show that most, but not all, dynamic algorithms perform better than static ones.**

***Index Terms*—load balancing, algorithms, cloud computing**

I. Introduction

In the paradigm of modern computing, cloud computing has come to be considered the backbone of most technology services [1]. The last two decades, in particular, saw an enormous rise in the utilization of cloud computing in all sorts of applications [2]–[5]. When one thinks of streaming, mapping, or social media, it is easy for the average user to forget that the brunt of the computational load is being offloaded to the cloud. In 2022, the total spending on cloud computing infrastructure reached $500 billion according to market research firm Gartner, with a projected 21% year-on-year increase in 2023 [6]. The most common current model for the cloud is in the form of substantial network-connected server farms with hundreds of thousands, if not millions, of machines. Cloud service providers like Microsoft and Amazon operate multiple geographically dispersed data centers [7].

With this level of demand and expansion comes the need to satisfy many customer- and end-user-based metrics. Among these metrics are response time, availability, and scalability. This is where the concept of load balancing comes in. A load balancer's job is to distribute a workload evenly to a set of servers. Cloud computing providers have found that this helps ensure service continuity and high availability.

When load balancing is applied, it provides the flexibility needed to scale an application to a larger cluster, data set, or user base. Moreover, load balanced systems result in better response times for their users. Let alone the benefit of reduced energy consumption of load balanced systems. For these reasons, load balancing has become an essential requirement of any meaningful cloud computing effort.

In general, there are two types of load balancing techniques: static load balancing and dynamic load balancing [8]. Static load balancing distributes tasks without considering the current state of each server, while dynamic load balancing considers this information when distributing workloads.

Static load balancing algorithms have been well-developed in the past and are widely adopted in the industry nowadays. In recent years, researchers have been developing more effective load balancing algorithms to achieve evenly distributed workloads and faster execution. However, due to the growth in cloud computing technologies and the rising demands of business organizations, traditional static load balancing methods fail to fulfill several quality of service (QoS) requirements [9]. Studies have been conducted to improve the existing static load balancing algorithms [10], [11]. More research focuses on dynamic load balancing algorithms [12]–[14]. Some proposed algorithms were compared to state-of-art algorithms in benchmarks, with the result showing those proposed algorithms can achieve better service provisioning, lower resource utilization, and higher overall performance [11], [14], [15].

Although more and more new load balancing algorithms which aim to solve specific problems have been proposed, it is still being determined how to select a suitable one from them once we encounter a new problem. This is because they were only compared with some classical algorithms, and the measurement environment was always different, such as various application backgrounds or cloud computing platforms. In our study, we conduct a comparative analysis of multiple load balancing algorithms with defined metrics to figure out the best strategy for different workload strengths and load patterns under a unified architecture and computing environment. Various load patterns, i.e., the uniform pattern, the tide pattern, and the spike pattern, which correspond to different real-life cases, are simulated to measure the performance of load balancing algorithms in our study. For instance, when a brand-new album by a top singer is released, the cloud-based music-streaming platform that publishes it might experience a daunting spike which could cause a perceptible delay for users. Based on the comparison under various performance measurements, a heuristic strategy is proposed for system designers to select the appropriate load balancing algorithm.

II. Related Works

A. General Surveys

Ghomi *et al.* introduced the general model of a distributed system and load balancing [8]. The model sends user requests to a central server, where the load balancing algorithm is applied. Requests are then dispatched from the central server into different worker nodes based on their loads. Several state-of-art static and dynamic load balancing algorithms were com-

979-8-3503-7099-7/24/$31.00 ©2024 IEEE

pared in their work. The results showed that each algorithm has its strengths and weaknesses in different applications.

P. Kumar and R. Kumar presented a survey of state-of-the-art cloud load balancing techniques and defined the challenges they try to attend to [9]. The study classified the algorithms into static, following a fixed set of rules, and dynamic, responding to the system's current state. We divided our related works, based on similar approach.

B. Dynamic Algorithms

The control mechanism for dynamic load balancing can be distributed or non-distributed. The strategies to make a load distribution decision are divided into information, transfer, and location. The issues of dynamic load balancing algorithms based on the system's current status were explained in detail by Alakeel *et al.* [12], alongside with the topics of load measurement, performance measurements, and system stability issues.

Rahmeh *et al.* presented a technique where the increment and decrement of the node's in-degree, which denotes whether the node completed a job or received a new job, were performed by Biased Random Sampling – where each node's selection depended on the free resources available to it [13].

Ren *et al.* introduced a dynamic load balancing algorithm with load forecasting [14]. The proposed algorithm can reduce load imbalance and improve service quality by using an exponential smoothing forecasting method for load prediction plus load estimation.

C. Other Algorithms and Approaches

In a distributed system, the heterogeneous environment uses machines with different capacities, in contrast to the homogeneous environment in which machines with similar capacities are used. Kapoor *et al.* [16] propose a cluster-based load balancing algorithm that works in the heterogeneous environment by grouping machines with similar capacities, which provides better performance than the traditional throttled load balancing algorithm. Their work also defined critical performance metrics, including response time, execution time and throughput.

Nishant *et al.* modified the Ant Colony Optimization (ACO) load balancing algorithm by continuously updating a single result set instead of their own result set [17]. This modified algorithm is suitable for the situation where the type of loads on nodes varies. Compared to classical methods, it ensured more smooth functioning of the cloud.

Wang *et al.* introduced a new genetic algorithm for load balancing [18]. They used a greedy algorithm to initialize the problem and a double-fitness function to evolve solutions by eliminating the poorest and mutating the best ones. The solutions refer to node-job allocations, and the fitness function represents node and job performance metrics.

Chen *et al.* described a cloud load balancing (CLB) algorithm [10]. The algorithm is based on the weighted round robin algorithm but keeps track of each node's load and allocates only half of the least loaded nodes each round.

In the paper by P. Kumar *et al.* [11], the authors design and implement a multi-scheduling load balancing (MTBLB) algorithm which contains three methods – awscloudwatch, cron job, and round robin. A comparison was made between MTBLB and round robin on the AWS platform, and the results showed that MTBLB has a lower response time, which makes it faster than the round robin algorithm.

To improve the performance of the distributed system, Jawad and Mahdi provide a load balancing algorithm with fuzzy logic (RLBF) [15], which transfers the values of capacity and queue size to low, medium, or high and then determines the state of the route. Compared to static and dynamic algorithms, RLBF showed higher throughput and shorter delay with the help of fuzzy logic to make decisions.

In the paper by Wang *et al.* [19], opportunistic load balancing (OLB) and load balance min-min (LBMM) algorithms were combined to form a new load balancing algorithm which integrated their advantages. Overall, OLB ensures that each node in the system works well, and LBMM can achieve a minimum task execution time.

Nahir *et al.* introduced a replication-based scheme to improve load balancing performance [20]. This approach creates several replicas for each task and sends them to different servers. If one of the replicas arrives at the head of the queue, then remove the other replicas.

D. Gaps in Previous Research

Although lots of approaches were discussed in the literature, some things need are not covered in their work. In [14], the forecasting results look promising in an experimental environment, but they might not perform well with request patterns that are not in the training set, such as random spiking. Some dynamic load balancing algorithms need to actively track server load information and remaining resources to make the optimal choice when dispatching tasks [19]. The problem with this approach is that server resource information is reported periodically with a defined interval, which could make the information used for decision-making outdated, leading to imbalanced workloads among servers. Complicated algorithms can take longer to compute than static algorithms such as round robin [14], [17], [18]. This will result in the load balancer being throttled in real-world applications where the cluster scale is significantly larger than the testing environment used in the literature. Furthermore, as mentioned in [11], the processing cost is an important aspect of load balancing. In our comparative study, we benchmark several load balancing algorithms using multiple load patterns to simulate real-world applications and provide insight into their strengths, limitations, and potential optimization strategies.

III. Problem Statement

We aim to study and compare the performance of several state-of-the-art load balancing algorithms. The goal is to develop recommendations that would aid system designers in choosing a suitable load balancing algorithm for their specific

needs. The following three subsections provide details on the platform and testing strategies.

A. Platform and Workflow

The application will be deployed on Amazon Elastic Compute Cloud (Amazon EC2) on the Amazon Web Services (AWS) platform for our study. The application contains three parts, a client, servers, and a load balancer - the focus of our study.

1) Platform: EC2 is a platform that provides adjustable cloud computing capabilities and is designed to simplify web-scale computing for developers. Furthermore, EC2 reduces the time to acquire and spin up new server instances to minutes, allowing us to quickly adjust the scale and architecture to our needs. We will deploy a single load balancing instance and several server instances for our purposes. The requests are generated and sent through the Internet using a request client application that connects to the load balancing instance.

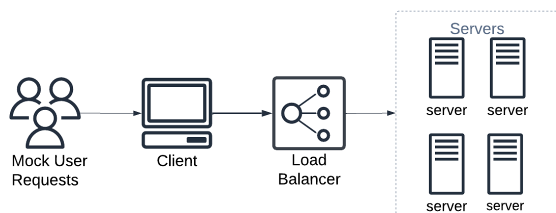

Fig. 1. Request Flow

2) Workflow: As shown in Fig. 1, to test different patterns of request flow, mocked user requests will be sent through a request-client application to the load balancer, where different load balancing algorithms will be used to conduct the comparative study. Then, the load balancer will distribute the tasks to different servers. For example, there is a listening platform for music that collects data on its listeners. Every time a listener starts playing a song on the client site, the time spent listening to the song and the listener ID are sent to the server, and the load balancer will decide which server will accept it.

B. Performance Testing Patterns

In our work, three performance testing patterns with tuned parameters will simulate the different real-world load cases. Moreover, random computation, memory, and I/O workload will be used to reflect various types of requests. For instance, in navigation applications, requesting a map needs more I/O workload to access the database, while requesting navigation routing has the burden of computation. The information about the workload needed to satisfy each client's requests is bundled with the request itself. The request indicates for each of the three categories (CPU, memory, and I/O) the workload level on a scale from 1 to 10.

1) Uniform Pattern: The Uniform pattern simulates an application accessed by a stable request flow in a time, shown in Fig. 2. For example, streaming sensor images transferred from a meteorological satellite is stable. Another example is long polling in the Internet of Things (IoT) to get information in time, where the intelligent devices send requests to the controller in a period so that the request rate is approximately constant. To make the testing reflect the difference between load balancing, the request rate is tuned deliberately to near the peak load in order to increase resource utilization.

2) Tide Pattern: The Tide pattern, shown in Fig. 3, aims to present a pattern similar to a tidal cycle. Many daily life applications correspond to this pattern. For instance, the subway fare gates experience morning and evening peaks in one day while both have similar intensity. Another application case is the restaurant food ordering app where the customer traffic is higher during lunch and supper, and the customers requests may queue up. Here we use two overlapping Gaussian distributions and repeat them to simulate those cases. It has two peak periods and a lower but not lowest valley between two peaks which denotes the off-peak period. The request flow of the interval between two patterns is lowest in order to represent the case at night. The peak load surpasses the server capacity to show if the load balancing could handle the crowded requests.

3) Spike Pattern: When a long-awaited album is released, or a shopping festival starts, the customer volume will surge in an extremely short time as a crowd of people rush to access the server, and then decrease gradually because the customers leave when they get the thing happily or do not purchase successfully. This is a typical long-tailed distribution. Here, we use the chi-squared distribution, which belongs to the family of long-tail distributions, to simulate this spike pattern, shown in Fig. 4. The value of the chi-squared distribution increases quickly and then reduces smoothly. The peak of the spike pattern is far beyond the server's capacity to stress test the performance of load balancing algorithms.

C. Performance Metrics

Load balancing algorithms assign tasks to a set of servers evenly. An ideal load balancing target is minimizing resource consumption while maximizing the efficiency of serving requests. There are various metrics for measuring efficiency since each has its preference for different application backgrounds. Metrics that would be used to measure the performance of load balancing in our paper are introduced below.

1) Throughput: Throughput is defined as the rate at which the traffic is passed through the server, measured in bits per second or requests per second for different targets. Even though throughput is measured in bits per second, it is usually considered a multiplication of the packet transmission rate and size. Throughput can be calculated as follows:

$$Throughput = Rate_{Packet\ Transmission} \times Size_{Payload} \quad (1)$$

Throughput is significant in sites where file and streaming media traffic patterns dominate.

2) Response Time: The time that the load balancers respond to a request is the response time. This includes a sum total of waiting time, transmission time and service time. It can be measured by subtracting the time the load balancer receives

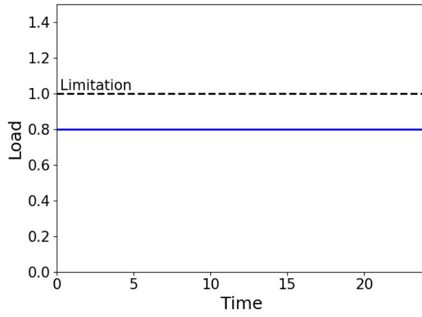

Fig. 2. Uniform Pattern

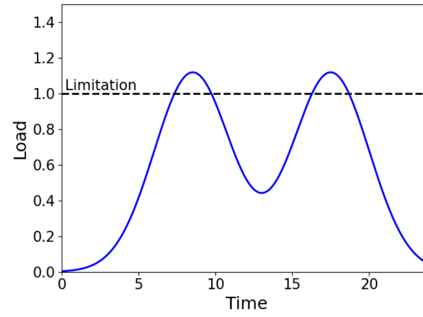

Fig. 3. Tide Pattern

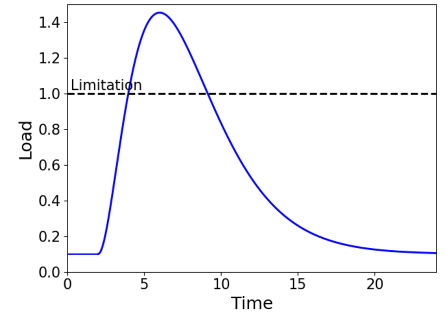

Fig. 4. Spike Pattern

a request from when a response is received from the server, indicating completion. Statistics for average, minimum, and maximum response times are recorded.

$$Response\ Time = T_{server\ response} - T_{client\ request} \quad (2)$$

3) Failed Connection Count: This metric measures the number of rejected connections. It inspects the various reasons for rejected requests, such as overloaded servers, not optimized load balancers, or uneven load distribution. Besides, failed connection count also helps to find whether the applications are scaling appropriately or not.

IV. Algorithms

A. Least Usage First

The first algorithm that we proposed is an algorithm that prioritizes the server with the lowest resource usage, named Least Usage First. The algorithm works in two phases, predict and sync. The sync phase utilizes predefined server APIs to acquire the current resource usage, including CPU, memory and IO. Dividing usage information by the current number of connections each server has, we can estimate how many resources one request will take for every server.

$$UsagePerRequest = \frac{Total_{usage}}{Number_{connection}} \quad (3)$$

Before the next sync phase, the algorithm proceeds to predict phase. When a server is assigned an incoming request, its usage will increase by the estimated usage per request calculated in the last sync phase. Similarly, when a server finishes a request and returns the response, its usage will decrease by the estimated usage per request. In this manner, the algorithm can predict servers' current usage. To choose a server for handling requests, the algorithm will always pick the server with the lowest usage.

B. Least Recently Used Based On Least Connection

The second load balancing algorithm is called Least Recently Used Based on Least Connection. In this algorithm we implement a hash map to store the connection count for each server and another hash map to store the time length for each server that has not been chosen. At each request, it chooses the server with the least count of requests it is currently processing; if there are several servers with that least count, then the algorithm chooses the server which has not been used for the longest time among those.

C. Least Average Response Time First

The Least Average Response Time First algorithm measures the average response time within a sliding window by subtracting the request time from the response time for a past fixed number of requests and selects the server with the least average response time.

We first set the average response time to 0 and choose servers in turns for the cold start situation. If the request is still in processing, its response moment is set to the current moment. Based on the assumption that each request requires a similar quantity of computing resources, this algorithm utilizes the average response time to indicate the server's performance. In our experiment, this algorithm might be susceptible to the sliding window to fall into a case that always chooses one server. An alternative solution is taking the reciprocal of the average response time as the weight.

D. Consistent Hashing Algorithm

The IP Hashing algorithm can take the client and server IP address to generate a unique hash key which is used to allocate the client to a particular server. However, there are still some things that could be improved. Firstly, the newly added server will make the initially calculated hash value inaccurate, and the hash key map must be updated. Secondly, downtime or shrinking can be attributed to deleting service nodes, leading to a large-scale update of the hash value. The Consistent Hashing algorithm solves these problems by constructing a ring-shaped hash space instead of a linear hash space, and the entire hash space is constructed as a first-placed ring.

E. Estimated Finish Time

The Estimated Finish Time algorithm, as discussed in [21], records each server's estimated finish time. For a server, this time represents the time of completion of all requests allocated to it by the algorithm. The algorithm always chooses the server with the soonest estimated finish time. This ensures that the request is currently allocated and will be given to the server to finish it as soon as possible.

Every time a request reaches the load balancer, the algorithm calculates the estimated completion time of the request on each server and adds it to the existing finish time estimate. The estimated finish time of a request on a given server (Equation 4) is calculated as the size of the request, for example, the number of instructions involved in fulfilling it, divided by the computational capacity of the server, which is measured in MIPS (millions of instructions per second). Accounting for the server capacity enables the algorithm to operate in a heterogeneous server pool. The algorithm then finds the server with the lowest sum of the estimated finish time and requests' completion time. That server is allocated the new incoming request and its estimated finish time is incremented by the estimated time to complete this request.

$$T_{request} = \frac{\textsc{Load}(Request)}{\textsc{Capacity}(Server)} \quad (4)$$

F. Baseline Algorithms (Controls)

1) Round Robin: Round Robin algorithm is a basic static load balancing algorithm that forwards requests to each server in a group of servers in turns. This algorithm does not consider the performance difference among servers.

2) Random: The random load balancing algorithm is a static algorithm that randomly chooses a server from the pool and allocates the incoming client request to it. It is meant as a baseline or control in the context of this study.

V. Results

Our testing platform consists of a single load balancing instance and eight worker instances of varying capacities for processing requests. For the following test results, the requesting client was configured to send 70k requests per second on the spike pattern, 55k on the tide pattern, and 40k in the uniform pattern. Requests have randomized parameters, including CPU, memory, and IO ranging from 1 to 10, indicating the task load levels.

A. Response Time and Failure Rate

In Fig. 5, Fig. 7, and Fig. 9, blue bars denote the average response time while red bars denote the failed request rate for each algorithm. The results of those patterns show some similarities. The out-performer is the Estimated Finish Time algorithm. Then, the Least Recently Used algorithm and the Least Usage algorithm show similar but slightly weaker performances. The above three algorithms are better than comparing groups, so they suit this simulated environment. The Least Response Time algorithm is better than the Random algorithm but worse than the Round Robin algorithm. Moreover, the Consistent Hashing algorithm is worse than the Random algorithm. Also, since overloading the servers results in more failed requests, there is a positive correlation between the average response time and failure rate.

B. Throughput

Fig. 6, Fig. 8, and Fig. 10 depict the throughput curve along with time in the uniform, tide, and spike pattern test, respectively. The grey line shows the request pattern we simulated, where the number of issued requests hovers near the capacity limitation of our testing environment in the case of the uniform pattern, briefly surpasses it in the case of the tide pattern and far surpasses it in the case of the spike pattern. Therefore, in the starting stage, each algorithm temporarily follows the request pattern line until it reaches its peak throughput and then keeps at that maximum capacity until it processes the backlog of requests it has issued. The three best performers, the Estimated Finish Time, the Least Recently Used, and the Least Usage algorithms, have the ability to keep the throughput at the maximum capacity. The Least Response Time and the Consistent Hashing algorithm could not assign the tasks appropriately, so they fluctuate frequently but do not get the maximum capacity point leading to the intensive requests overwhelming servers. In the worst case, these algorithms complete the backlog of requests later than the end of the request pattern by up to 75 seconds.

VI. Discussion

A. Static Algorithms

Static algorithms are widely used in cloud computing platforms today. Round Robin, a popular static algorithm, is the default load balancing algorithm on AWS' elastic load balancer [22]. It performs reasonably well given its simple nature, and an improvement is the Weighted Round Robin algorithm which adds a weight factor to the servers, thus giving more powerful ones proportionally more requests. The Consistent Hashing algorithm guarantees that the request from a specific IP address is sent to the same server, so the external session storage is unnecessary. However, this algorithm does not send the request by evaluating the server's capacity, so overloading is possible. Furthermore, from the throughput, the Random algorithm and Consistent Hashing algorithm showed similar results since the general idea of them are both assigning the request to a server directly. However, to assign to the specific server, a consistent hashing algorithm has a hashing calculation process and comparisons among the hash value of requests and servers' IP, which explains why a consistent hashing algorithm has a longer response time and higher failure rate for all three patterns.

B. Dynamic Algorithms

As for dynamic algorithms, they can achieve better performance at the cost of more complex implementation. Despite the implementation details, the difference in performance ultimately comes to what information the dynamic algorithm can access. In our experiment, the Estimated Finish Time algorithm analyzes the requested content to evaluate the resource usage of each request. This algorithm relies on clearly defined, parsable requests to estimate the server's load accurately. The Least Usage algorithm periodically polls the server's load information and chooses the server with the least estimated

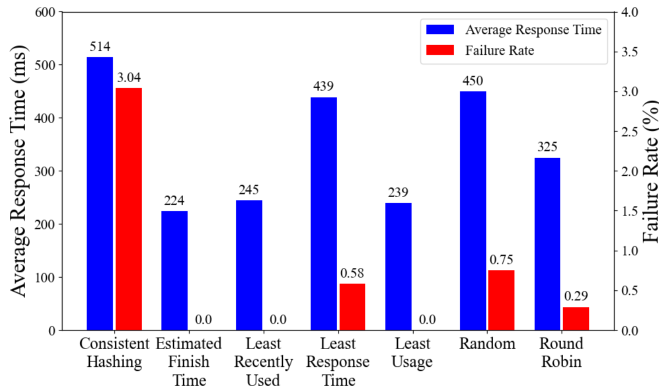

Fig. 5. Response Time and Failed Request Count Results for Uniform Pattern

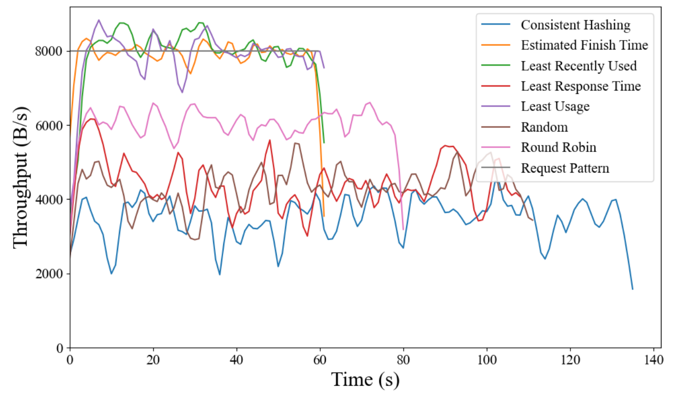

Fig. 6. Throughput Results for Uniform Pattern

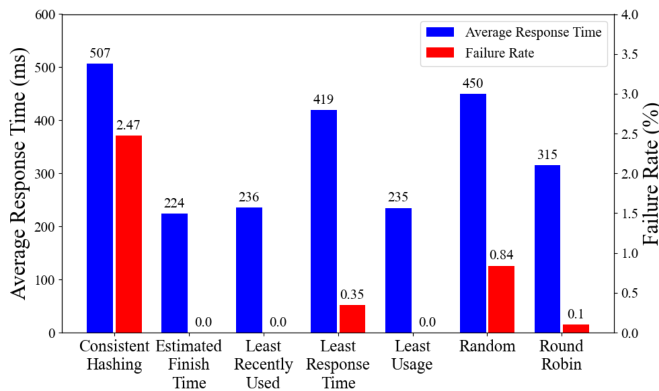

Fig. 7. Response Time and Failed Request Count Results for Tide Pattern

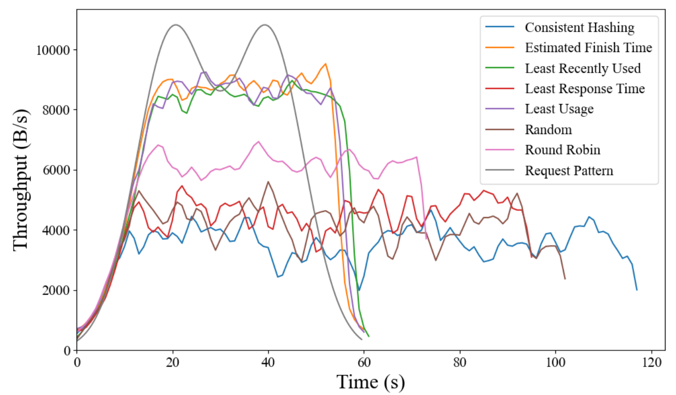

Fig. 8. Throughput Results for Tide Pattern

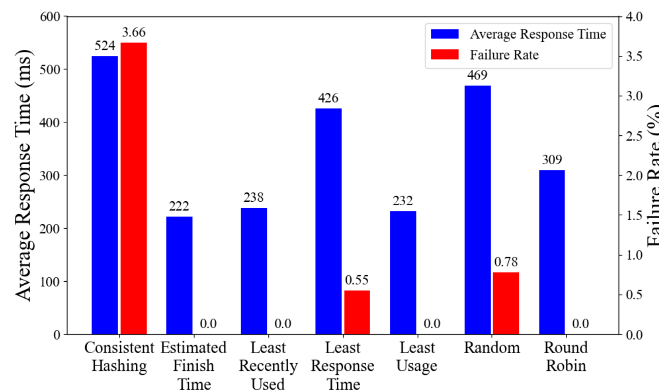

Fig. 9. Response Time and Failed Request Count Results for Spike Pattern

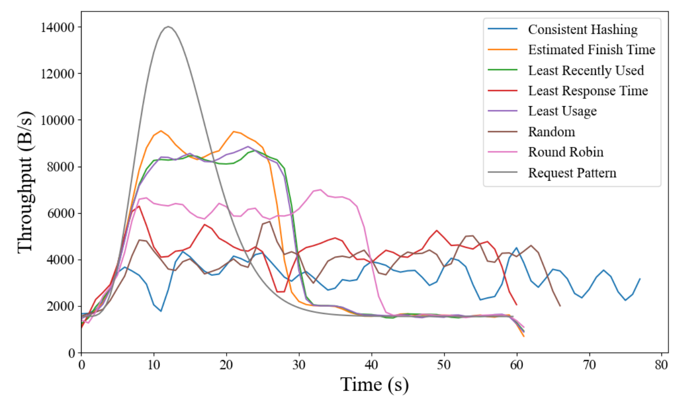

Fig. 10. Throughput Results for Spike Pattern

load. This algorithm requires a pre-deployed resource monitor on each server, which is some extra work if the platform does not come with real-time resource monitor and query APIs. The Least Recently Used Based On Least Connection algorithm chooses the server with the least active connection count; if there are several servers with the least connection count, then it chooses the server with the longest unused time. This algorithm can minimize the chance of server overload and improve the average response time performance. However, its main limitation is that it assumes all requests have the same load and thus cannot quickly adapt to fluctuations.

On the other hand, the Least Average Response Time algorithm is purely results-based without the need to know each server's status. It simply prioritizes the server that responds faster, which makes it easy to implement. However, it sometimes overloads the server with the lowest response time because the algorithm relies on a moving average which could be more responsive. Therefore, choosing what information to access can significantly affect the scope and complexity of the algorithm.

C. Scalability

When choosing a load balancing algorithm, developers should also consider future scalability. There are two ways of scaling, horizontal scaling – adjusting the number of servers, and vertical scaling – adjusting the capacities of existing servers. Depending on the pricing of the cloud computing platforms, system designers need to choose the best scaling solution with a higher performance-cost ratio. However, static algorithms do not work well with vertical scaling, as the algorithms do not know the capacity difference between servers. Even though upgrading all servers is an option, downtime might be infeasible for the business.

On the other hand, dynamic algorithms can handle both scaling methods well. Not only do dynamic algorithms support vertical scaling without downtime, but upgrading a subset of servers is also possible. This provides much more flexibility to the scaling solutions, making dynamic algorithms superior to static algorithms.

VII. Conclusion

In this study, we explored the design of a unified testing platform for load balancing algorithms. To exhibit the use of this platform, five load balancing algorithms were benchmarked against two baseline algorithms on metrics of throughput, response time, and failure rate. Overall, dynamic algorithms are better than static algorithms in terms of adapting to heterogeneous environments, performance fluctuations, and future scaling. Nevertheless, that ultimately comes at the cost of increased complexity and the requirement of varying degrees of information.

References

[1] L. Qian, Z. Luo, Y. Du, and L. Guo, "Cloud computing: An overview," in *Cloud Computing First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009, Proceedings*, vol. 5931, 01 2009, pp. 626–631.

[2] J. Bourne, "The decade in cloud: Analysing the 'remarkable transformation' through saas, iaas and paas rise," Jan 2020. [Online]. Available: https://www.cloudcomputing-news.net/news/2020/jan/07/decade-cloud-analysing-remarkable-transformation-through-saas-iaas-and-paas-rise/

[3] M. Aibin and M. Blazejewski, "Complex elastic optical network simulator (ceons)," in *2015 17th International Conference on Transparent Optical Networks (ICTON)*, 2015, pp. 1–4.

[4] M. Aibin, "Dynamic routing algorithms for cloud-ready elastic optical networks," *Ph. D. dissertation*, 2017.

[5] M. Karimibiuki, M. Aibin, Y. Lai, R. Khan, R. Norfield, and A. Hunter, "Drones' face off: Authentication by machine learning in autonomous iot systems," in *2019 IEEE 10th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*, 2019, pp. 0329–0333.

[6] "Gartner forecasts worldwide public cloud end-user spending to reach nearly $500 billion in 2022," Apr 2022. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2022-04-19-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-reach-nearly-500-billion-in-2022

[7] M. Aibin and K. Walkowiak, "Resource requirements in fixed-grid and flex-grid networks for dynamic provisioning of data center traffic," in *2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2016, pp. 1–4.

[8] E. J. Ghomi, A. M. Rahmani, and N. N. Qader, "Load-balancing algorithms in cloud computing: A survey," *Journal of Network and Computer Applications*, vol. 88, pp. 50–71, 2017.

[9] P. Kumar and R. Kumar, "Issues and challenges of load balancing techniques in cloud computing: A survey," *ACM Computing Surveys (CSUR)*, vol. 51, no. 6, pp. 1–35, 2019.

[10] S.-L. Chen, Y.-Y. Chen, and S.-H. Kuo, "Clb: A novel load balancing architecture and algorithm for cloud services," *Computers & Electrical Engineering*, vol. 58, pp. 154–160, 2017.

[11] P. Kumar, D. M. Bundele, and D. Somwansi, "An adaptive approach for load balancing in cloud computing using mtb load balancing," in *2018 3rd International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE)*, 2018, pp. 1–5.

[12] A. M. Alakeel *et al.*, "A guide to dynamic load balancing in distributed computer systems," *International Journal of Computer Science and Information Security*, vol. 10, no. 6, pp. 153–160, 2010.

[13] O. A. Rahmeh, P. Johnson, and A. Taleb-Bendiab, "A dynamic biased random sampling scheme for scalable and reliable grid networks," *INFOCOMP journal of computer science*, vol. 7, no. 4, pp. 1–10, 2008.

[14] X. Ren, R. Lin, and H. Zou, "A dynamic load balancing strategy for cloud computing platform based on exponential smoothing forecast," in *2011 IEEE international conference on cloud computing and intelligence systems*. IEEE, 2011, pp. 220–224.

[15] M. M. Jawad and N. M. Mahdi, "Prototype design for routing load balancing algorithm based on fuzzy logic," in *2019 4th Scientific International Conference Najaf (SICN)*, 2019, pp. 92–96.

[16] S. Kapoor and C. Dabas, "Cluster based load balancing in cloud computing," in *2015 Eighth International Conference on Contemporary Computing (IC3)*, 2015, pp. 76–81.

[17] K. Nishant, P. Sharma, V. Krishna, C. Gupta, K. P. Singh, Nitin, and R. Rastogi, "Load balancing of nodes in cloud using ant colony optimization," in *2012 UKSim 14th International Conference on Computer Modelling and Simulation*, 2012, pp. 3–8.

[18] T. Wang, Z. Liu, Y. Chen, Y. Xu, and X. Dai, "Load balancing task scheduling based on genetic algorithm in cloud computing," in *2014 IEEE 12th international conference on dependable, autonomic and secure computing*. IEEE, 2014, pp. 146–152.

[19] S.-C. Wang, K.-Q. Yan, W.-P. Liao, and S.-S. Wang, "Towards a load balancing in a three-level cloud computing network," in *2010 3rd International Conference on Computer Science and Information Technology*, vol. 1, 2010, pp. 108–113.

[20] A. Nahir, A. Orda, and D. Raz, "Replication-based load balancing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 494–507, 2016.

[21] N. K. Chien, N. H. Son, and H. Dac Loc, "Load balancing algorithm based on estimating finish time of services in cloud computing," in *2016 18th International Conference on Advanced Communication Technology (ICACT)*, 2016, pp. 228–233.

[22] "How elastic load balancing works." [Online]. Available: https://docs.aws.amazon.com/elasticloadbalancing/latest/userguide/how-elastic-load-balancing-works.html