The Evolution of Algorithms and Techniques of Load Balancing in Distributed Systems

SHARAN KRISHNA, California Polytechnic State University, USA
AARAV SHARMA, California Polytechnic State University, USA

Load balancing has been a critical component in the advancement of distribution systems, cloud computing, and more. This survey paper explores the evolution of load balancing starting from the earliest models, hitting a key transition point, and adapting to the complex and dynamic nature of the modern world. Specifically, the paper will dive into different styles of algorithms and techniques over the years in an attempt to solve the challenge of distributing workloads effectively.

ACM Reference Format:
Sharan Krishna and Aarav Sharma. 2018. The Evolution of Algorithms and Techniques of Load Balancing in Distributed Systems. *ACM Trans. Graph.* 37, 4, Article 111 (August 2018), 13 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

As computing demands have grown over the years, efficient load balancing has become a necessity in distributed systems. Load balancing is a method of optimally distributing workloads across multiple computing resources, improving performance, scalability, and fault tolerance. With modern applications like cloud computing and data centers relying on distributed computing, the importance of more intelligent load-balancing strategies has increased. Without effective load balancing, systems face bottlenecks, increased latency, and a general lack of efficiency, degrading performance and reliability.

The evolution of load balancing starts with the history of distributed computing, transitioning from early queuing-based models to modern-day AI-integrated algorithms. In the beginning, static algorithms assigned workloads based on fixed probabilities or simple rules, as shown in queuing theory models. Over time, more complex techniques emerged, using randomized algorithms, consistent hashing, and even reinforced learning to allow systems to dynamically adjust task distribution based on real-time workload conditions.

This survey paper explores the historical development, modern advancements, and future of load balancing in distributed computing. We begin by examining early models, such as queuing-based approaches and deterministic job-routing strategies that laid the foundation for today's techniques. Next, we analyze more modern algorithms, like data-efficient reinforcement learning, stochastic coordination, and cloud-optimized load balancers, which improve scalability and efficiency. Finally, we discuss emerging trends and future research directions in load balancing, including AI optimization, real-time adaptive balancing, and more strategies designed for next-generation cloud computing.

This paper aims to highlight the practical implications and theoretical advancements of load balancing. Our discussion will emphasize key trade-offs between techniques, such as latency reduction, fault tolerance, and scalability to provide insights into how modern distributed systems can balance workload effectively.

1.1 Context

Load balancing is a vital aspect of modern computing and is crucial to the performance, scalability, and reliability of networking and distributed and parallel computing. As the volume and complexity of tasks have grown exponentially, the importance of efficiently distributing workloads has also grown. Our topic focuses on the evolution of load-balancing techniques and algorithms to show how they have changed to follow the evolution of computing to where we are today. We plan on covering the entire evolution of load balancing, ranging from simple strategies for early distributed systems to complex algorithms in cloud computing and data centers.

Not only will we discuss the different load-balancing techniques in history and the modern world, but we will also go into detail about how and why they have changed, including integration in networking and distributed computing and use cases in homogeneous systems. Overall, we aim to do more than just answer the question 'What is a load balancer; we want to explain its presence in primitive strategies and the modern world. Therefore, we can shed light on an important part of distributed computing and provide insight into the future of load-balancing algorithms.

1.2 Paper Selection

1.2.1 Models for dynamic load balancing in a heterogeneous multiple processor system. Found through Google Search. This paper is about early load-balancing techniques in distributed systems, mainly queuing models and job-routing strategies designed to balance workloads across machines. This paper will be extremely good for our survey paper as it laid the groundwork for the current state of distributed computing, such as with the cloud. This paper will allow us to connect early states of load balancing to modern ones from other papers we will cite and compare and contrast them to show the evolution of the techniques. It will serve an introductory purpose to our paper and explain the basis of load balancing, but will not be too heavily relied on compared to more modern papers.

1.2.2 The Role of Load Balancing Algorithms in Next Generation of Cloud Computing. Found through Google Search. This paper categorizes load balancing techniques into 3 models and analyzes them to provide insight into how they work and compare them against each other. This paper is extremely relevant to our survey

Authors' Contact Information: Sharan Krishna, California Polytechnic State University, San Luis Obispo, USA, krishna.sharan@gmail.com; Aarav Sharma, California Polytechnic State University, San Luis Obispo, USA, aaravsharma927@gmail.com.

ACM 1557-7368/2018/8-ART111
https://doi.org/XXXXXXX.XXXXXXX

paper since it not only will serve as a basis for the current state of load balancing in distributed computing, but it also provides insights into where the future of cloud computing will go. This will help us not only make connections with the evolution of load balancing algorithms, but also extend into what the future might hold for the techniques. This paper will be heavily relied on since it represents the modern world of load balancing and thoughts on its future.

1.2.3 Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. Found through Google Search. This paper introduces the concept of consistent hashing, which is a load-balancing technique designed to distribute requests evenly across a changing set of web servers. This algorithm is positioned between the early load-balancing techniques of the late 1970s and the current state of them today, which means it represents an important advancement in load-balancing itself. This paper and its analysis of this pivotal technique will help us bridge the gap between the decades between different load-balancing techniques show their evolution. This paper will not be heavily relied on but will serve as a bridge between the early state of load balancing and the modern world.

1.2.4 Stochastic Coordination in Heterogeneous Load Balancing Systems. Found through Google Search. This paper provides an optimization solution to load-balancing issues in distributed systems, presenting an algorithm based on research, mathematical analysis, and extensive simulations. Being a recent paper, the work done here will highlight the current state of load balancing algorithms and allow us in turn to highlight how load balancing has evolved through the decades. The extensive work done on this specific algorithm will help us show the progression from the older algorithms to now. This paper will serve a smaller role compared to other papers on modern load balancing, but will still be essential in explaining how it has evolved.

1.2.5 BCLB: A Scalable and Cooperative Layer-4 Load Balancer for Data Centers. Found through Google Search. This paper explores how load balancers work in modern data centers and the performance-based challenges that need to be addressed. The authors propose a new approach to load balancing, and since this paper is very recent, it can provide insight into the types of load balancers that may be built going forward. We anticipate this paper not being a core paper, but we believe it is still valuable due to its recency.

1.2.6 Load Balancing for Communication Networks via Data-Efficient Deep Reinforcement Learning. Found through Google Search. This paper introduces an approach to using reinforcement learning to address load-balancing inefficiencies in cellular networks. This is a compelling paper for us because it shows how some researchers are trying to design load-balancing algorithms from a different angle. With the rapid developments in machine learning and deep learning, we believe that this type of approach might be further explored. Although this is a fairly interesting paper, we may swap it for a different paper.

1.2.7 The power of two choices in randomized load balancing. Found through Google Search. This paper is a very influential and cited paper in the realm of load balancing where the author discusses a simple approach of using randomness of two choices. This paper introduces a foundational and effective approach to load balancing and helps contrast the more complex algorithms. In addition, it acts as a transition point between early load-balancing models and modern approaches used in distributed systems.

1.2.8 Load is not what you should balance: Introducing Prequal. Found through Google Search. This paper discusses a load balancer build by Google Research that focuses on reducing real-time request latency instead of balancing CPU load. The Prequal system is currently used to load balance within YouTube which provides insight into how load balancing works practically and at scale. This paper will be core to our discussions because it is a recent and applicable use of where load balancing is at the industry level.

1.3 Key Issues

Our survey paper with a historical view of the development of load-balancing algorithms, where we discuss the progression of techniques and foundational ones like queuing models from paper 1.2.1 and consistent hashing from paper 1.2.3. Next, we move into a bridge to modern load balancing, talking about the power of two choices from paper 1.2.7. Finally, we go into depth on more current techniques, focusing on details for modern algorithms like data-efficient reinforcement learning and randomized models. Specifically, we want to discuss the common and differing goals in these techniques, like improving scalability and latency. This is a transition into the modern world of load balancing, also including cloud computing and YouTube's Prequal. We examine how load balancing adapts to the changing requirements of modern-day distributed systems with specifics like latency optimization and different overall topologies.

2 Historical Development of Load Balancing Algorithms

The early development of load-balancing techniques was driven by the need to optimize task distribution in heterogeneous environments. Initial research focused on queuing models, which tried to improve system efficiency by dynamically distributing tasks over multiple processors. As distributed systems scaled and the internet introduced new challenges like server hot spots and content delivery issues, new techniques like consistent hashing were developed to manage workloads in a decentralized manner. These foundational techniques laid the groundwork for modern load-balancing algorithms.

2.1 queuing Models and Early Load Balancing Strategies

One of the earliest models of dynamic load balancing was proposed by Chow and Kohler (1979), who introduced queuing-based approaches for job scheduling in heterogenous multiprocessor systems. Their work categorized load balancing into two strategies: deterministic and nondeterministic, with each of them designed to minimized job turnaround time while maintaining overall system efficiency.

2.1.1 Nondeterministic Load Balancing Strategies. Nondeterministic Load Balancing strategies used probability-based job routing, meaning that incoming tasks were assigned to processors according

to predefined probabilities instead of real-time system conditions. The core idea was to statistically distribute workload rather than react dynamically to individual task queues.

- Uniform Probability Model: Jobs are assigned to each processor with an equal probability, which works well for homogenous processors, but leads to inefficiencies in heterogenous systems.
- Weighted Probability Model: Assigns tasks based on processor speeds. For example, a processor twice as fast as another receives jobs twice as frequently. This is more efficient than uniform models, but it fails to adjust to dynamic workload changes.

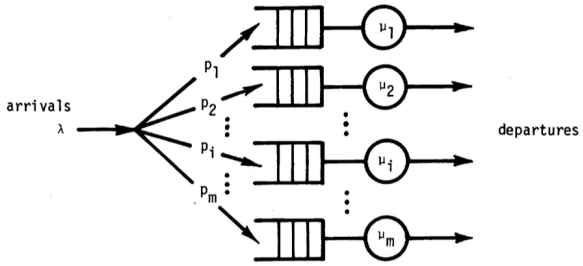


Fig. 1. Heterogeneous system with nondeterministic routing. (Chow and Kohler, 1979)

2.1.2 Deterministic Load Balancing Strategies. In contrast with probabilistic methods, determinstic approaches dynamically assign job assignments based on real-time system conditions. Chow and Kohler proposed three main deterministic strategies:

- Minimum Response Time (R) Policy: In this strategy, you assign each incoming job to the processor with the shortest estimated queue time. This approach reduces the latency per job, but it requires centralized knowledge of each processor's queue length. This works best in low-latency environments, but it suffers under high scheduling overhead.
- Minimum System Time (T) Policy: Here, you assign jobs to balance overall system use instead of individual response times. This is more globally efficient but harder to implement in real-time settings.
- Maximum Throughput (TP) Policy: This allocates jobs based on arrival rates and processor speeds to maximize system throughput. This performs best under high system loads, making sure that all processors remain active and properly used. However, it does not consider individual job urgency, making it less suitable for latency-sensitive applications.

While these early queuing-models demonstrated the effectiveness of dynamic versus static job allocation, they struggled with scalability and adaptability in distributed systems. These systems needed decentralized and fault-tolerant load-balancing techniques; this led to the development of consistent hashing, a breakthrough in caching and resource allocation.

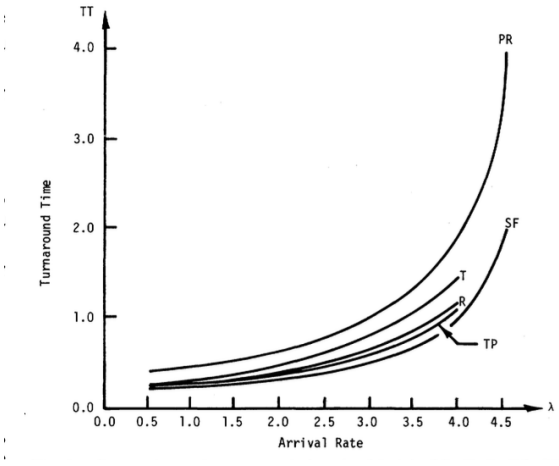


Fig. 2. Comparison of performance of the different Deterministic Routing Policies (Chow and Kohler, 1979)

2.2 Consistent Hashing and Decentralized Load Distribution

With the expansion of the World Wide Web, new challenges emerged:

- Server hot spots: Popular websites could easily become overwhelmed, especially during major online events
- Scalability: Traditional load balancing relied on centralized knowledge of system state, which became less feasible as networks grew
- Fault tolerance: If a server failed, massive task remapping was required in traditional hash-based distribution methods.

Traditional hashing techniques assigned objects, like website requests and cached data, to servers using a modulo-based hash function such as:

$$h(object) = hash(object) * mod(N) \quad (1)$$

where N is the number of available servers. The main issue with this approach was instability - if a server was added or removed, nearly all objects would need to be remapped to new servers. This required a massive redistribution of the cached data, leading to delays and inefficiencies.

In 1997, Karger et. al introduced consistent hashing, a technique that allowed for minimal disruption when adding or removing servers. The key innovation was mapping both servers and objects to a continuous hash space (a ring structure) instead of assignment them based on a static modulo function.

Steps in Consistent Hashing:

- Each server is assigned a position on a circular hash space in a range of 0 to 4294967296. For example, server A receives a value of 120, and server B receives a value of 450.
- Each object is hashed onto the same circular space. For example, Object X receives a hash value of 400.
- Finally, an object is assigned to the next available server in the clockwise direction. Since object X hashes to 400, it is assigned to server B, since it is the closest at 450.

There are numerous advantages to consistent hashing. The first is minimal disruption, where if a server is removed, only a small fraction of objects must be reassigned. Another is scalability, by which consistent hashing will work efficiently in large distributed environments like Content Delivery Networks and Cloud Computing. A final one is fault tolerance, where servers can fail without causing a system-wide remapping. This is because if a server fails, only the objects assigned to that server need to be reassigned to the next available server in the clockwise direction.

Both queuing-based approaches and consistent hashing introduced fundamental concepts that remain vital to modern load balancing, including the move to dynamic adaption, decentralization, and fault tolerance and elasticity. Queuing models established the importance of real-time load awareness, which has influenced modern AI-driven adaptive balancing today. Consistent demonstrated that local decision-making could provide global efficiency, which has paved the way for peer-to-peer and cloud-based architectures of the modern world. Through addressing different challenges, queuing models and consistent hashing have provided the building blocks for modern load-balancing techniques.

3 The Transition to Modern Load Balancing

As the years went by and technology in distributed systems and in general advanced, the lack of efficiency in these traditional load balancing systems like queuing models and consistent hashing became increasingly obvious. Queuing models often required a centralized state awareness, while consistent hashing reduced remapping overhead, but didn't explicitly minimize the queue imbalance across nodes. Their inability to adapt to real-world situations and the obvious issue of inefficiency in high workloads became a bigger issue, which led to the introduction of randomized load-balancing strategies. Such strategies use probabilistic selection mechanisms to improve the efficiency of the algorithms while maintaining the same scalability improved by consistent hashing.

3.1 The Power of Two Choices

One of the most impactful developments in this transition was the Power of Two Choices, introduced by Michael Mitzenmacher in 2001. This algorithm demonstrated that choosing between just 2 randomly selected servers rather than assigning a job at random results in an exponential improvement in load distribution. This principle laid the groundwork to modern-day load-balancing techniques in cloud computing and data centers, optimizing performance while further minimizing queuing delays from before.

The Power of Two Choices Algorithm modifies standard random load balancing:

- A new job arrives at a system with n servers
- Instead of being randomly assigned to any available server as done before, it chooses the least loaded of two randomly selected servers

The system followed a first-in, first-out (FIFO) queuing approach, and exponentially distributed service times in the process. Mitzenmacher illustrated this simply in the paper with the supermarket model. For any fixed time T and d >= 2, the expected time a customer spends in an initially empty supermarket system over the first T units of time is bounded above by

$$\sum_{i=1}^{\infty} \lambda^{\frac{di-d}{d-1}} + o(1) \tag{2}$$

where o(1) is n -> infinity and may depend on T and lambda. This approach leads to exponential reductions in queue length variance, making it a rather useful technique in dynamic, large-scale systems.

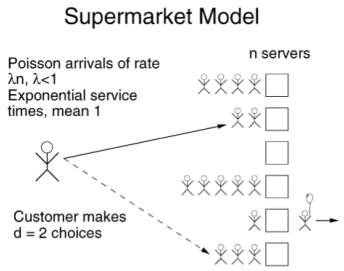


Fig. 3. Supermarket Model with d = 2 (Mitzenmacher 2001)

To continue analyzing the queue dynamics in the power of two choices, Mitzenmacher modeled the system using jump Markov processes and differential equations.

3.1.1 Markov Process Formulation. A density-dependent family of Markov chains Xn is a sequence of jump Markov Processes such that: the state space of Xn is:

$$E_n = E \cap \{n^{-1}k : k \in \mathbb{Z}^D\} \tag{3}$$

and the transition rates of Xn are:

$$q_{x,y}^{(n)} = n\beta_n^{(y-x)}(x), \quad x, y \in E_n \tag{4}$$

While complicated, these formulas show how jobs move between queue states in a probabilistic setting.

3.1.2 Deterministic Limiting System. As the system size grows, a deterministic limiting system is introduced to approximate behavior:

$$\begin{cases} \frac{ds_i}{dt} = \lambda(s_{i-1}^d - s_i^d) - (s_i - s_{i+1}) & \text{for } i \geq 1, \\ s_0 = 1. \end{cases} \tag{5}$$

with the boundary condition being s0 = 1. Simply put, this models how the job arrival rates and server selection time influence the queuing behavior.

3.1.3 Fixed Point and Exponential Decay of Queue Lengths. A key result from the previous analysis was the existence of a fixed point that satisfies the equation:

$$s_i = \lambda^{d^i - 1/d - 1} \tag{6}$$

The resulting equation and analysis show that long queues become exponentially less frequent with this algorithm, which is the fundamental reason why the Power of Two Choices algorithm significantly reduces congestion.

3.1.4 Expected Time in System and Scaling Behavior. Using the equations he found, Mitzenmacher could compute the expected time a customer spends in the system:

$$T_d(\lambda) \equiv \sum_{i=1}^{\infty} s_i^d \tag{7}$$

which, for large n, simplifies to

$$1/\log(d) \tag{8}$$

.

With the results of these equations, Mitzenmacher provided results for simulations based on the supermarket model. He ran simulations with both 100 and 500 queues respectively, which demonstrated the impact of having only two choices. Rather than just choosing 1 server every time, having the choice between two based on load decreased the average time overall, as shown by the trend in the graph.

The Power of Two choices algorithm represents an important bridge between traditional and modern load balancing. Exponentially reducing queue imbalances provides a scalable and low-overhead solution for distributed systems. It also proved scalability through Mitzenmacher's equations, with the system converging to the derived fixed point. This was a very important milestone in load balancing, with applications in modern day cloud computing and also paving the way to it.

TABLE 1
Average Time in the Supermarket Model: 100 Queues

Choices	λ	Simulation	Prediction	Rel. Error (%)
2	0.50	1.2673	1.2657	0.1289
	0.70	1.6202	1.6145	0.3571
	0.80	1.9585	1.9475	0.5742
	0.90	2.6454	2.6141	1.1981
	0.95	3.4610	3.3830	2.3028
	0.99	5.9275	5.4320	9.1227
3	0.50	1.1277	1.1252	0.2146
	0.70	1.3634	1.3568	0.4858
	0.80	1.5940	1.5809	0.8314
	0.90	2.0614	2.0279	1.6533
	0.95	2.6137	2.5351	3.1002
	0.99	4.4080	3.8578	14.2607
5	0.50	1.0340	1.0312	0.2637
	0.70	1.1766	1.1681	0.7250
	0.80	1.3419	1.3289	0.9789
	0.90	1.6714	1.6329	2.3564
	0.95	2.0730	1.9888	4.2363
	0.99	3.4728	2.9017	19.6825

Fig. 4. 100 Queue Simulation Table (Mitzenmacher 2001)

TABLE 2
Average Time in the Supermarket Model: 500 Queues

Choices	λ	Simulation	Prediction	Rel. Error (%)
1	0.99		100.00	
2	0.99	5.5413	5.4320	2.0121
3	0.99	3.9518	3.8578	2.4366
5	0.99	3.0012	2.9017	3.4305

Fig. 5. 500 Queue Simulation Table (Mitzenmacher 2001)

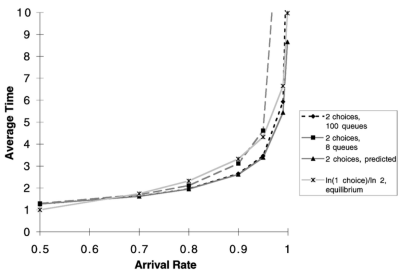


Fig. 6. Graph of Arrival Rate vs. Average Time (Mitzenmacher 2001)

4 Modern Load Balancing

In the modern day, load balancing has many applications, from cloud computing and data centers to reinforcement learning. With the transition from simpler algorithms to the power of two choices and beyond, load balancing was able to take a large step and enter the modern day.

4.1 Load Balancing in Cloud Computing

Cloud Computing has revolutionized the way computing resources are allocated, managed, and used. However, a fundamental challenge in cloud computing is the idea of load balancing to ensure that workloads are evenly distributed across multiple servers to optimize resource use, minimize response time, and prevent system overload. Efficient load balancing mechanisms contribute to increased performance, scalability, and reliability in cloud environments, which are essential in the modern day.

In their paper *The Role of Load Balancing Algorithms in Next Generation of Cloud Computing*, Basetty Mallikarjuna and Arun Kumar Reddy Doddi double down on the need for load balancing. In a cloud computing system, computing resources such as virtual machines, storage units, and processing nodes operate within a shared infrastructure. Due to consistent fluctuating demand, ensuring a balanced workload distribution is essential to improving response times, minimizing resource wastage, optimizing energy consumption, reducing processing delays, and enhancing quality of service (QoS). Cloud computing environments are dynamic, which means that static resource allocation strategies, like preassigning resources without real-time adjustments, are often inefficient. Instead, the cloud needs dynamic load-balancing algorithms to adapt to the real-time demand.

Mallikarjuna and Doddi broadly classify load balancing in cloud computing into 3 categories:

- **Static Load Balancing (SLBA)**: These algorithms distribute workloads based on predefined rules and do not adapt dynamically. Some examples are Round Robin, Min-Min, and Max-Min algorithms.
- **Dynamic Load Balancing (DLBA)**: These algorithms adjust resource allocation in real-time based on system load. Examples include Genetic Algorithm (GA), Active Clustering Algorithm (ACA), and Ant Colony Optimization (ACO).

- **Nature-Inspired Load Balancing Algorithms (NILBA)**: These algorithms take inspiration from natural processes like honey bee foraging (N-LBHBF) and Osmosis (N-OLB).

Static Load Balancing Algorithms (SLBA)	Dynamic Load Balancing Algorithms (DLBA)	Nature Inspired Dynamic Load Balancing Algorithms (DNILBA)
Round Robin Algorithm (RRA) [Sharma S, Singh S, Sharma M. 2008]	Active Clustering Algorithm (ACA) [RamPrasadPadhy et al., 2011]:	Osmosis Load Balancing Algorithm (N-OLB) [P.V Krishna & B.Mallikarjuna 2015)
Opportunistic Load Balancing Algorithms (OLA) [Wang SC, Yan KQ, Liao WP, Wang SS 2010]	Random Biased Sampling (RBS) [Rahmeh et al., 2008]	
Min Min Algorithm (MiMiA) [Kokilavani et al., 2011]	Genetic Algorithm (GA) Algorithm [HuJinhua et al., 2010]	Beecolony Optimization Algorithm (N-BCO) [B.Mallikarjuna, P.V Krishna 2018]
Max Min Algorithm (MxMiA) [UpendraBhoi et al., 2013]	Ant Colony Optimization Algorithm (ACOA) [RatanMishra and AnantJaiswal 2012]	Load Balancing Honey Bee Foraging Equal Time Allocation Policy(N-LBHBF)[P.V Krishna & B.Mallikarjuna 2018]
Two Phase Algorithm (TPA) [KaranpreetKaur et al., 2013]	Honey Bee Foraging Algorithm (HBFA) [Kirshna et al., 2013]	

Fig. 7. Load Balancing Algorithms in Cloud Computing (Mallikarjuna and Doddi 2019)

Mallikarjuna and Doddi researched and tested many algorithms of all these types, including the ones mentioned above as examples. They formed a table to provide a comparative analysis of various load-balancing approaches in terms of key performance metrics like throughput, resource utilization, and scalability.

Parameter	S-RRA	S-OLA	S-MiMiA	S-MxMiA	S-TPA	D-ACA	D-GA	D-HBFA	D-ACA	N-OLB	N-BCO	N-LBHBF
Through Put	✓	✗	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓
Overhead	✓	✗	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓
Resource Utilization	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓	✓
Response Time	✓	✗	✓	✓	✓	✗	✗	✗	✗	✓	✓	✓
Scalability	✗	✗	✗	✗	✗	✓	✓	✓	✗	✓	✓	✓
Performance	✓	✗	✓	✓	✓	✓	✓	✓	✗	✓	✓	✓
Fault Tolerance	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓
Fault Tolerance	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓

Fig. 8. Metrics of Various Load Balancing Algorithms (Mallikarjuna and Doddi 2019)

From this table and their testing, they found that DNLBA and DLBA were the most appropriate for cloud computing - **dynamic load balancing**.

As they continue in their paper, Mallikarjuna and Doddi talk specifically about next generation cloud computing. Cloud computing itself has evolved significantly over the years, introducing a variety of architectures and approaches to optimize performance, resource utilization, and cost efficiency. Next-generation cloud computing aims to improve these existing models by using things like multi-cloud environments, hybrid cloud solutions, federated cloud computing, and emerging edge computing technologies. In all of this, load balancing is vital to ensure optimal performance, latency, and fault tolerance.

4.1.1 Multi-Cloud Load Balancing. Multi-cloud environments leverage services from multiple cloud providers, allowing organizations to optimize costs, improve redundancy, and enhance performance. However, managing workloads across different cloud infrastructures poses significant challenges due to differing pricing models and performance metrics.

The MODA Clouds Load Balancer addresses these challenges by sending requests to application servers based on specific load-balancing policies. It consists of a load-balancing controller that manages real-time decisions on request distribution and a reasoner that adapts weights dynamically based on heterogeneity in cloud resources. In such multi-cloud environments, HAProxy is a widely used open-source solution that enables TCP and HTTP-based load balancing, which ensures traffic distribution across cloud-based applications.

4.1.2 Hybrid Cloud Load Balancing. Hybrid cloud architectures combine public and private cloud resources, allowing organizations to keep sensitive workloads in private clouds while leveraging public cloud scalability for high-demand applications. Load balancing in hybrid cloud environments requires specialized algorithms to:

- Allocate resources dynamically between private and public clouds.
- Ensure seamless application performance during high-traffic periods.
- Address security concerns when shifting workloads.

A key example of hybrid cloud load balancing is the CliQr platform, which enables businesses to manage workloads effectively across public and private clouds. Statistics show that 63% of enterprises in the healthcare and energy sectors have adopted hybrid cloud environments, highlighting its growing importance.

4.1.3 Federated Cloud Load Balancing. Federated cloud computing involves multiple cloud providers collaborating under a single framework to provide seamless service interoperability. The Open-Cloud Computing Federation (OCCF) integrates various cloud computing service providers (CCSPs) to ensure:

- Workload distribution across federated resources.
- High service availability and reduced latency.
- Interoperability between different cloud platforms.

The main goal of load balancing in federated clouds is to prevent local resource overload by intelligently distributing requests across the entire cloud federation. This approach ensures higher consumer satisfaction and optimized facility utilization, making federated cloud an essential component of next-generation cloud computing.

4.1.4 Micro Cloud and Cloudlet Load Balancing. With the increasing demand for low-latency computing, microclouds and cloudlets have emerged as a solution to minimize data transfer delays and reduce reliance on central cloud infrastructure. These small-scale cloud environments are deployed closer to users and offer real-time computational capabilities. Some key benefits of micro clouds and cloudlets are lower latencies, decentralization, and energy efficiency since these are smaller scale and allow data to be closer to users. Technologies like Raspberry Pi and Odroid boards are being used to build micro clouds, particularly in edge computing applications. These systems help decentralize computing by offloading tasks from data centers and end-user devices to micro-cloud environments.

4.1.5 Ad-Hoc Cloud Load Balancing. Ad-hoc clouds emerge from the dynamic orchestration of cloud resources in real-time. Unlike traditional cloud environments, where resource allocation is predefined, ad-hoc clouds allow for spontaneous resource pooling based on immediate demand. Some key characteristics of Ad-Hoc Load Balancing are dynamic resource allocation, decentralized load distribution, and fault tolerance. A practical implementation of ad-hoc cloud load balancing can be seen in mobile ad hoc networks (MANETs), where devices contribute spare computing resources to support nearby users. This model is particularly useful in disaster recovery scenarios, where rapid deployment of cloud resources is necessary.

4.1.6 Heterogeneous Cloud Load Balancing. Heterogeneous cloud computing refers to cloud environments that integrate diverse computing resources, including different processor types (i.e. GPUS, CPUS, TPUS), Multiple hypervisors (VMWare, KVM, Xen), and hybrid software architectures. Load balancing in such environments is complex due to the varying capabilities of the underlying hardware. GPU-accelerated clouds, for example, require scheduling algorithms that account for different processing speeds and power consumption levels.

After covering many different methods of load balancing in Next-generation cloud computing, Mallikarjuna and Doddi also touch on the future of load balancing in cloud computing. The future of load balancing in cloud computing is being shaped by emerging trends in the field such as:

- **AI and Machine Learning-Based Load Balancing**: AI-powered algorithms predict workload trends and adjust resource allocations dynamically.
- **Edge and Fog Computing**: Load balancing strategies will extend beyond centralized data centers to edge devices, optimizing latency-sensitive applications.
- **Serverless and Software-Defined Computing**: Load balancing in serverless architectures will focus on function-level optimizations, reducing computational overhead.
- **Energy-Aware Load Balancing**: Future algorithms will prioritize energy efficiency, reducing the carbon footprint of cloud infrastructures.

The main point is that load balancing is a crucial component of cloud computing. As cloud computing evolves, different forms of load balancing will evolve along with it to always distribute workloads and follow different, cutting-edge algorithms.

4.2 Algorithmic Optimization

A major step in the modern development of load balancing is creating more efficient and clever algorithms that can handle the load problems of today but apply the underlying ideas presented in the solutions such as consistent hashing and the power of two choices. Many researchers in academia are focusing their efforts on transforming the complex nature of a load balancing system into an optimization problem. They often look at some of the poorly distributed workloads in modern industry-based systems and optimize performance metrics such as response time, delays, and server imbalance. Even though these solutions do not yet been applied to the real-world balancers, it provides insight into how load balancing

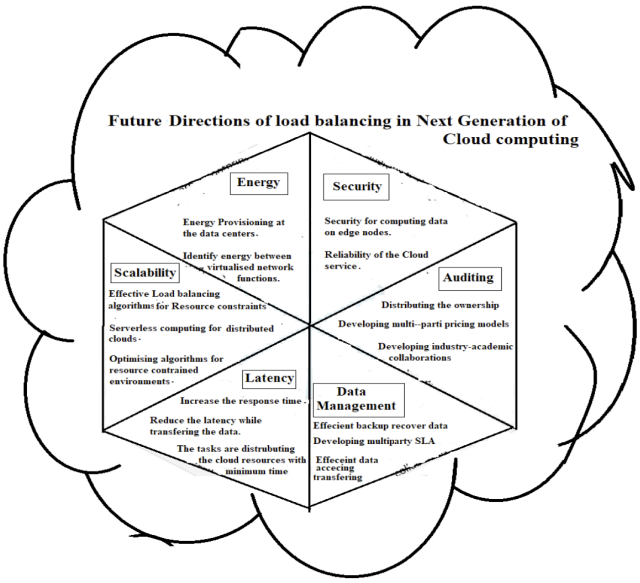


Fig. 9. Future of Load Balancing(Mallikarjuna and Doddi 2019)

can be optimized for performance at a mathematical and theoretical basis.

4.2.1 Stochastic Coordination. Guy Goren, Shay Vargaftik, and Yoram Moses, introduced an algorithmic-based load-balancing solution in their paper entitled, *Stochastic Coordination in Heterogeneous Load Balancing Systems*. Their paper underscored that load balancing is extremely challenging due to the distributed designs with heterogeneous servers and multiple dispatchers. These dispatchers must make independent and immediate decisions in an environment where they are expected to keep request rates high and response times low. To help mediate this challenge, Goren, Vargaftik, and Moses formulated the load balancing problem as a stochastic optimization problem and focused on creating an algorithmic solution based on mathematical analysis. It was also important for the researchers to create a computationally efficient solution because they knew that industry professionals would be opposed to deploying a solution to real-world systems that worsened the time complexity.

4.2.2 The Herding Problem. The core issue that Goren, Vargaftik, and Moses pointed to in many traditional load balancing systems was the "herding" problem. The "herding" problem occurs when many load balancers within the same system evaluate a certain server to have great performance metrics. Each of those load balancers will send their request to that server, without realizing that other load balancers also directed their requests to that server. In other words, the time sending the request is greater than that of a load balancer detecting that the server is not a good choice to send the request. As a result, the server receives a large number of requests from load balancers, slowing down the processing time per request and creating more delayed responses.

Specifically, when examining existing "join-the-shortest-queue (JSQ)"-based approaches, the dispatchers' information is highly correlated. This means that multiple dispatchers receive similar updates about queue lengths from the servers, which causes the dispatchers sends tasks to the same "shortest" queue creating a sudden load spike to that server. To combat the problems of JSQ, shortest-expected-delay (SED) approaches were introduced that uses the normalized queue lengths instead of the actual queue lengths as the metric to determine which queue is shortest. Although the SED approaches helped with the "herding" problem in an environment with heterogeneous servers, the issue persisted with multiple, distributed dispatchers. Despite several recent efforts to solve "herding", no approach has resolved the issue, and all remain susceptible to correlated dispatcher decisions.

4.2.3 How Stochastically Coordinated Dispatching (SCD) Works. With trying to solve the "herding" problem in mind, Goren, Vargaftik, and Moses devised an algorithm called Stochastically Coordinated Dispatching (SCD) that focuses on achieving ideally balanced assignments. The goal of an ideally balanced assignment (iba) is to minimize the difference between the highest loaded and lowest loaded servers. The server load here is defined as the expected time it would take to process the jobs currently in its queue. An assignment that satisfies the below mathematical formula would be an iba.

$$\max \min_{s \in S} \frac{q_s + \bar{a}_s}{\mu_s}$$

$$\text{s.t.} \quad \sum_{s \in S} \bar{a}_s = \sum_{d \in D} a^{(d)} \quad \text{and} \quad \bar{a}_s \geq 0 \quad \forall s \in S.$$

In SCD, the load balancing system contains n servers and m dispatchers operating that operate on discrete and synchronized rounds. Each of these servers has a FIFO queue, and the values of queue lengths are known to all dispatchers at every round. Each round consists of arrivals (requests), load balancer dispatching, and departures (responses), and servers have an expected time-independent processing rate. The optimization problem is set up like so:

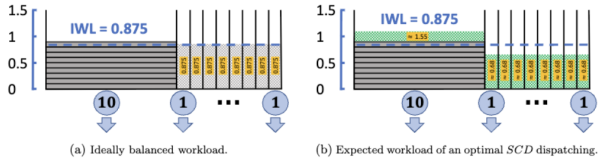
$$\min_{P} \quad f(P) = (a-1) \sum_{s \in S} \frac{1}{\mu_s} \cdot p_s^2 + \sum_{s \in S} \frac{2(q_s - \mu_s \text{IWL}) + 1}{\mu_s} \cdot p_s$$

$$\text{subject to} \quad \sum_{s \in S} p_s - 1 = 0,$$

$$p_s \geq 0 \quad \forall s \in S.$$

Here are the general steps to SCD:

(1) In every round, the SCD updates the queue lengths of every server. Here, the servers are sorted by the queue length divided by their processing capacity. This ensures the load balancers know which server is likely to have the shortest expected processing time, not the shortest queue like in traditional algorithms
(2) Then, SCD is given a batch of jobs and it estimates the arrival rate for these jobs. It computes what the researchers defined as the "Ideal Workload" based on the current queue lengths and server capacities
(3) The algorithm calculates the probabilities at which jobs must be routed to each server. This is accomplished according to the estimated load and processed queue information in order to further optimize the load balancing.
(4) This is achieved by assigning probabilities to all the servers such that jobs are sent in a randomized way to servers with shorter normalized queue lengths in proportion to their processing capacity
(5) A server is randomly selected based on the calculated probabilities upon arrival of jobs and those jobs are sent to the selected server for processing.



(a) Ideally balanced workload. (b) Expected workload of an optimal *SCD* dispatching.

Fig. 10. Comparison of most optimal workload and SCD optimized workload (Goren, Vargaftik, Moses 2021)

4.2.4 SCD Evaluation. The overall complexity of the SCD algorithm is dominated by sorting, leading to a time complexity of O(nlogn) in each dispatching round.

Steps with Large Time Complexity

- Updating queue lengths and computing ideal workload: O(n)
- Sorting the servers based on queue lengths and processing rates: O(nlogn)
- Estimation of total incoming jobs across all dispatchers: O(m)

In Figure 11, the pseudocode-based algorithm for SCD is devised where computing ideal workloads and computing probablities are separate algorithms that are called in the overall algorithm:

Algorithm 2: StochasticallyCoordinatedDispatching (SCD) Procedure.

input: $\mathcal{S}$, $\{\mu_s\}_{s \in \mathcal{S}}$
for *each round* $t = 1, 2, \ldots$ **do**
  Update queue-lengths $\{q_s\}_{s \in \mathcal{S}}$
  $\mathcal{S}_1 \leftarrow$ Sort $\mathcal{S}$ according to $\frac{q_s}{\mu_s}$
  $\mathcal{S}_2 \leftarrow$ Sort $\mathcal{S}$ according to $\frac{2q_s+1}{\mu_s}$
  upon *receiving jobs* $a^{(d)}$ **do**
    Estimate a by $a_{\text{est},d}$
    IWL $\leftarrow$ ComputeIdealWorkLoad$(\mathcal{S}_1, \{q_s\}_{s \in \mathcal{S}}, \{\mu_s\}_{s \in \mathcal{S}}, a_{\text{est},d})$
    $P \leftarrow$ ComputeProbabilities$(\mathcal{S}_2, \{q_s\}_{s \in \mathcal{S}}, \{\mu_s\}_{s \in \mathcal{S}}, a_{\text{est},d}, \text{IWL})$
    for *each job in* $a^{(d)}$ **do**
      draw a server s according to P
      send job to server s

Fig. 11. Algorithm for Stochastically Coordinated Dispatching (Goren, Vargaftik, Moses 2021)

Through the researcher's evaluation and simulation, they compared SCD against some of the other approaches that had difficulty in mediating the "herding" problem in a heterogeneous multi-dispatcher setting. The two main performance criteria are the average response time over all client requests and the response times'

tail distribution. We see that in Figure 12, SCD has a shorter average response time and significantly shorter response time delays than the traditional approaches.

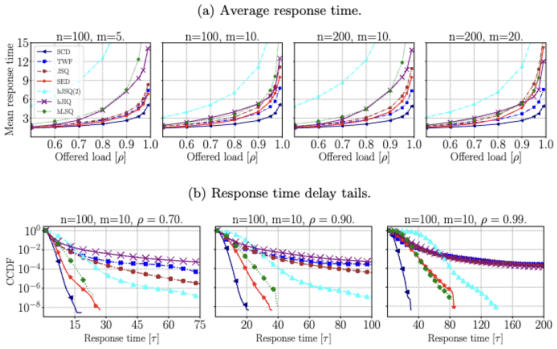


Fig. 12. Comparing Average Response Times and Response Time Delays Across Algorithms(Goren, Vargaftik, Moses 2021)

Even with the promising results that the researchers displayed, it does not come without its shortcomings. With SCD, all dispatchers need to be connected to all servers and the work that a job requires is very depending on the server processing that job. It is important to state that these are simulated results and there are no guarantees about how this algorithm will perform in the real-world.

Looking at the big picture, this paper showed the potential problems of load balancers following a set of predefined round-robin-like rules that dominate many industry-level load balancer systems today. We saw with the power of two choices that adding some randomness to the load balancing system helps shorten response times and reduce delays and the SCD algorithm operates in the spirit of power of two choices with the use of optimized probabilities. More papers like Goren, Vargaftik, and Moses's could potentially lead to changes in how industry professionals view load balancing.

4.3 Network and Hardware Optimizations

With the importance of load balancing to data centers and data centers arguably being the most crucial infrastructure in the digital world, the need to built a high-performance load balancer cannot be understated. Even though there is a large focus on optimizing the specific algorithms that load balancing systems operate on, network and hardware optimizations can lead to practical benefits that industry professionals can get behind. As a result, researchers in academia are looking to optimize load-balancing systems through a network and hardware lens. These hardware and network optimizations can help deal with the increasing network traffic and resource demands and ensure data centers can continue to scale.

4.3.1 Hardware vs Software Load Balancing. In the paper, *BCLB: A Scalable and Cooperative Layer-4 Load Balancer for Data Centers*, researchers Shu Yang, Xinze Wu, and Laizhong Cui categorize load balancers into two primary types: hardware-based (dedicated physical load balancers) and software-based load balancers (SLBs) where they touch upon each of their advantages and disadvantages.

Hardware-based load balancers provide faster forwarding speeds and are designed to handle high volumes of network traffic effectively. In addition, they are equipped to manage complexity in packet handling. Even though these load balancers are effective, they are very expensive and can act as single points of failure within a data center's infrastructure. The key disadvantage was the hardware-based load balancers were not as adaptable to frequent network rules and protocol changes.

Software-based load balancers are usually deployed on regular servers and unlike hardware-based ones, one can easily support newer protocols. Since these software-based load balancers utilize the server's infrastructure, they are far less costly, which makes them attractive to use in data centers. However, these load balancers have much more limited compute capacity due to the server's network interface card's limitations. In addition, software-based load balancers experience more performance bottlenecks leading to severe response delays.

The researchers also discussed cooperative schemes that distribute load balancing rules among servers, but through their own research found that they can be sensitive to traffic fluctuations and rule updates. As a result, the researchers hope to formulate a load balancer that can operate at the hardware-level where it can not only bridge the best parts of both types of load balancers but operate using a cooperative scheme

4.3.2 Bit-based Collaborative Load Balancer. Taking the considerations of hardware and software load balancing, Yang, Wu, and Cui devised a solution called the Bit-based Collaborative Load Balancer (BCLB). BCLB is a load balancer that leverages existing switches to enhance layer-4 (transport layer) load balancing capabilities. To optimize for the BCLB's performance, the researchers formulated an optimization problem.

Here are the two key features that make the BCLB effective and highly efficient:

(1) **Cooperative Scheme**: As opposed to many of the traditional load balancers that distribute the load balancing rules across different machines, BCLB requires multiple switches to communicate with each other to check different bits. This breaks down the task into smaller parts where each switch can focus on one specific part of the packet header.
(2) **Bit-based Checking**: BCLB takes the information in the network and transport layer headers known as the 5-tuple(source IP, destination IP, source port, destination port, and protocol type) and stores it in a trie structure within the switches (Static Random Access Memory (SRAM). Each switch is responsible for checking specific bits in this trie allowing the switches to collaboratively process incoming data packets. This means that each switch does not need to process the entire set of load-balancing rules. These rules refer to the decisions the load balancer has to make about how to distribute incoming network traffic across multiple switches.

4.3.3 LB Software Architecture in a Data Center. BCLB operates under a Software Defined Networking (SDN) framework, which helps separate the control plane and the data plane in network devices. BCLB has a central controller that is responsible for managing

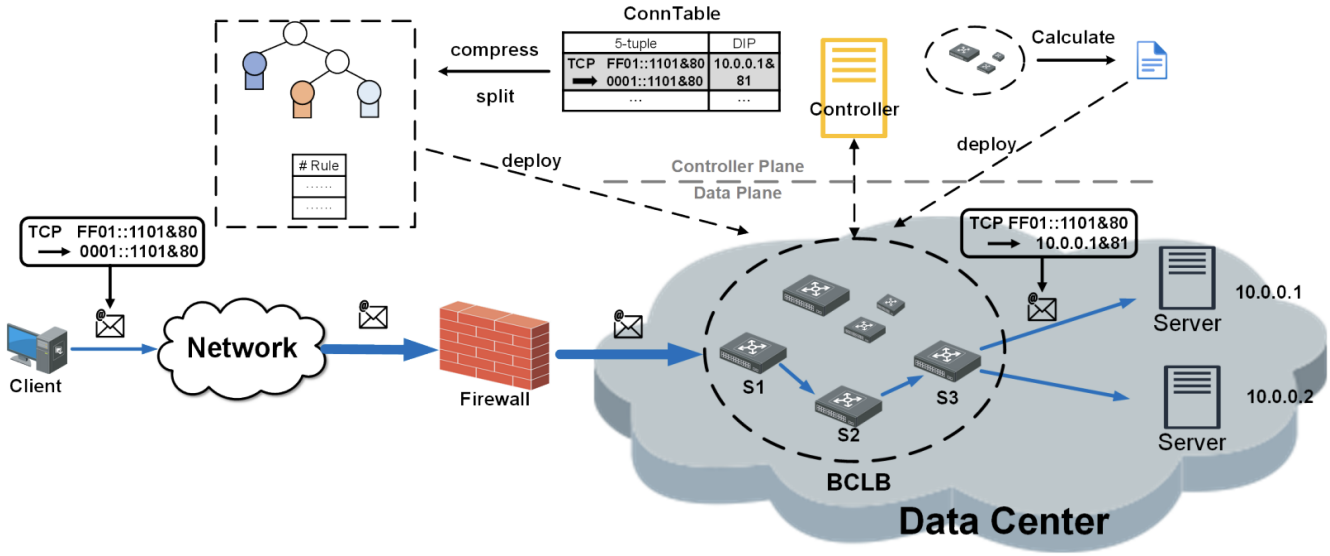


Fig. 13. BCLB Architecture in a Data Center (Yang, Wu, Cui 2024)

the distribution of configurations to the switches, including the bit allocations and lookups in the trie. At a more detailed level, BCLB uses Distributed Bloom Filter Intersection (BFI) to decompose the 5-dimensional trie into individual data structures for efficient lookup in SRAM. This creates a system where switches can check partial bits and passing intermediate states to the next hop switch. Figure 13 shows a great diagram of BCLB's role in the flow of network traffic from the client to the end servers. It also depicts the split of the controller and data plane responsibilities of each load balancer.

4.3.4 Linear vs Multiple-Edge Topology. The paper discusses two key ways in which the switches can be arranged.

One arrangement is using a linear topology where switches are directly connected to each one-by-one in a chain. This allows for a straightforward flow of packets from the ingress switch to the egress switch. The researchers developed an algorithm called the Linear-balance Algorithm suited for the linear topology that allows for the cooperative scheme between switches. It ensures that each switch checks its designated bits correctly while balancing the load across the switches. The goal is to prevent any single switch from becoming overloaded while others remain underutilized. The complexity of this algorithm is approximated to $O(|V| \times |T|^2)$, where $|V|$ is the number of switches and $|T|$ is the number of bits to be processed.

The other arrangement is the multi-edge topology where multiple ingress routers feed traffic into the data center. Each router can connect to multiple switches creating a tree-like structure seen in Figure 14. For this topology, the paper introduces the Multi-root-balance Algorithm that accounts for the variability in network traffic load across each of the routing paths. The algorithm examines the sub-trees of the load balancer networks and then calculates the minimal and maximal load on each switch to help prevent bottlenecks. The overall time complexity of the multi-root-balance algorithm is: $O(|S| \times |V| \times |T|^2)$ where $|S|$ is the number of ingress switches, $|V|$ is the number of switches and $|T|$ is the number of bits to be processed.

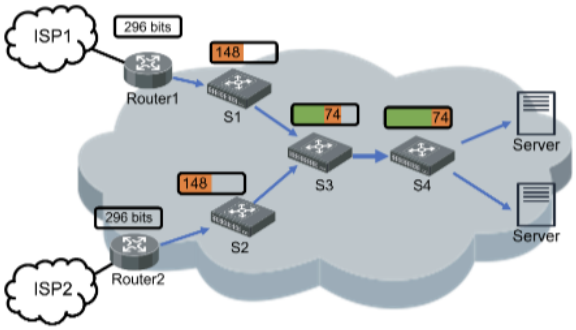


Fig. 14. BCLB Multi-edge Topology (Yang, Wu, Cui 2024)

4.3.5 BCLB Implementation. After their formulation, the researchers created a BCLB prototype which was implemented using Behavioral Model Version 2 (BMv2), an open-source software for simulating switches. The implementation like in the architecture divided the control plane and the data plane. The control plane was responsible for performing each respective topology algorithm (Linear-balance and Multiroot-balance Algorithms) and a data plane with the trie, protocol, and pointer tables. In Figure 15 shows the diagram of their simulated BCLB prototype implementation

4.3.6 BCLB Evaluation. BCLB was compared with a simple rule-based scheme using real-world traffic datasets and data center topologies. The researcher's focus was to see will their hardware and

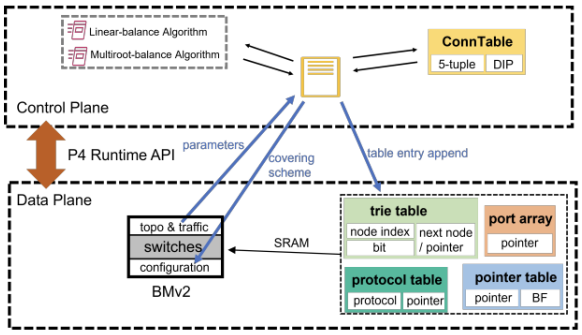


Fig. 15. Implementation framework of BCLB (Yang, Wu, Cui 2024)

network optimizations and software architecture reduce both CPU utilization and response. The results show that using the of the Linear-balance Algorithm show that the min-max load decreases as the number of switches increases, outperforming the rule-based scheme. The CPU utilization in BCLB is considerably lower than in the simple rule-based scheme as shown in Figure 16 and there is more stable forwarding latency.

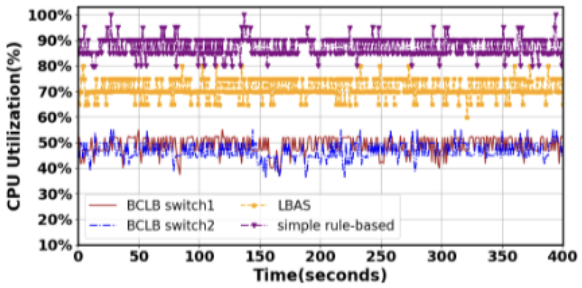


Fig. 16. Comparison of CPU utilization across different schemes (Yang, Wu, Cui 2024)

Looking at the big picture, the hardware and network optimizations like creating and implementing BCLBs are very promising. The network flow into the data center will only continue to increase and it is important to view the load balancing optimization from not only an algorithmic perspective but also one that takes into account the end-to-end hardware and software architecture. The BCLB is one of the many solutions that focus on this type of optimization but what makes the paper unique is the wide variety of practical considerations discussed that can be translated into the modern data center realm.

4.4 AI-Based Load Balancing

Over the last few years, artificial intelligence has emerged has a hot new problem-solving mechanism and industry professionals and researchers are continuously looking to apply the AI-based approaches to various domains including load balancing. Due to the complex nature of load balancing as seen throughout the several different load balancing approaches discussed so far, rudimentary AI/ML algorithms do a poor job of distributing workloads in a dynamically changing environment. On the other hand, deep learning and reinforcement learning are often applied to complex problems like load balancing because they can learn how to adapt to the dynamic data flow of the world.

4.4.1 Why Deep Reinforcement Learning. In the paper entitled *Load Balancing for Communication Networks via Data-Efficient Deep Reinforcement Learning*, Wu et al. proposed a deep reinforcement learning approach for load balancing in communication networks like wireless and cellular. The researchers discuss how Load balancing has become a core problem in modern 4G and 5G communication networks where current-day load balancing needs to provide more quality service and network performance. The load-balancing algorithms that exist today are based on manually predefined rules and non-dynamic environments. Using predefined rules is too simple and ineffective for an environment as complex as today's communication networks. Integrating reinforcement learning (RL) and especially deep RL (DRL) approaches into load balancing will allow load balancers to understand the states of the environment better to make decisions that do not reduce the performance as constantly.

4.4.2 Solving the Data Problem. Most deep learning or reinforcement learning approaches always face a common problem: The need for more training data. Acquiring sufficient data to train deep reinforcement learning models can be challenging and time-consuming. To combat this, instead of model learning from a zero-knowledge level, the researchers would use transfer learning. This is where the knowledge gained while solving one problem is applied to a different but related problem. In the case of load balancing, similar environments can share common characteristics in terms of network traffic flow and user behavior. One can use previously solved load-balancing scenarios to enhance the learning process of new situations that come across.

4.4.3 Transfer Deep RL-Based Load Balancing. This paper specifically focuses on an algorithm called Transfer Deep RL-Based Load Balancing (TRLLB) that uses the core aspects of deep, reinforcement and transfer learning applied to optimizing cellular network traffic. This model focuses on idle mode users (IMLB)

In applying TRLLB to load balancing, Wu et al. modeled load balancing as a Markov Decision Process (MDP). As part of the MDP, there are three main components:

- **State Space**: Each state consists of three variables which are the number of active UEs (User Equipment), the cell physical resource block (PRB) ratio, and the average IP throughput of each cell
- **Action Space**: Each action focuses on adjusting the cell reselection parameters and guiding which cell IMLBs should use when they become active.
- **Reward Function**: The reward is based on the average IP throughput per UE per cell and the model focuses on minimizing this

TRLLB focuses on two core ideas that are repeated continuously.

- **Creating Policies From Actions**: When the model performs a chosen action, it sends the results of that action to the

policy back, producing a policy. A policy contains a learning experience from a previously solved load-balancing problem.

- **Using Learned Policies to Produce New Actions**: The model uses the learned policies in the policy bank to create a new policy for a different but related load balancing problem, which influences the model on the next action it should take.

Figure 17 depicts the steps of TRLLB from choosing an action, storing the results of the action, and learning from it.

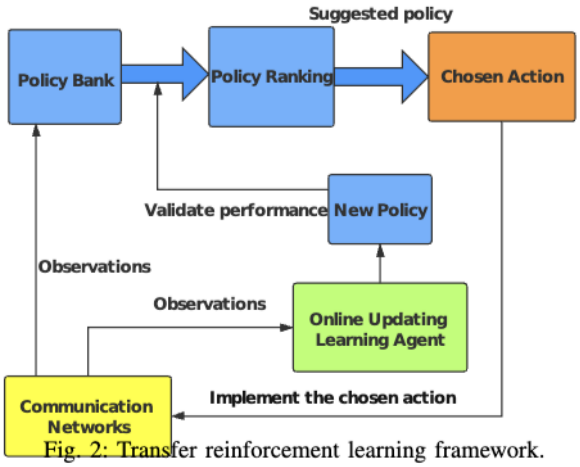


Fig. 17. TRLLB Workflow (Wu et al. 2021)

Through storing the results of the actions as policies in the policy bank and reusing the policies from similarly related tasks, TRLLB reduces the amount of training data needed to be effective. TRLLB also improves average throughput compared to traditional rule-based methods and other deep learning and reinforcement learning approaches.

4.4.4 Future of AI in Load Balancing. With Transfer Deep RL-Based Load Balancing, we saw how different an AI-based approach was than non-AI based approaches in solving the load balancing problem like most other problems. AI-based approaches allow the load balancers to better adapt to the dynamic network traffic environment and show improvement in many performance metrics. Even with the transfer learning, there is still a potential cap on how well an AI model can perform due to the data problem. In addition, there are questions about how the AI approaches can transfer to real-world communication systems. With AI just emerging as the main problem-solving technique, there is still lots of room for it to optimize load balancing in the future.

4.5 Load Balancing in the Industry

Much of the recent research in load balancing from stochastic coordination to deep reinforcement learning are impressive as they have shown in theory or in a simulated environment that they improve the key system performance metrics the researchers hope to improve. However, these approaches have not be applied to industry-level load balancing systems just yet so it raises questions on how it will scale and deal with real-world challenges. At the industry-level, load-balancing solutions cannot just focus on optimization performance metrics but need to be able to scale very well with heterogeneous infrastructures. There are also many key considerations like reliability, security, and fault tolerance that also needs to be address beyond just performance. Still, the traditional load balancing methods are widespread throughout the industry due to their simplicity and scalability.

4.5.1 Load Balancing at Large-Scale. Researchers Bartek Wydrowski, Robert Kleinberg, Stephen M. Rumble, and Aaron Archer at Google Research wrote a paper called *Load is not what you should balance: Introducing Prequal*. Prequal is the load balancer that Google's Youtube, which means it operates not only in the real-world but at very large scale.

Specifically, Prequal (Probing to Reduce Queuing and Latency) focuses on load balancing for distributed multi-tenant systems. For large-scale services like YouTube, servers are constantly running many distributed jobs and are communicating through RPC. Since many server replicas handle these jobs, a load balancing policy is necessary to distribute the workload properly. This means that balancing CPU utilization across replica servers is key. However, instead of just balancing the CPU loads, Prequal focuses on reducing request latency.

Here are the system design features of Prequal that help reduce request latency:

- **Probing**: Prequal continuously probes two key variables of the server replicas through sampling. Prequal specifically uses asynchronous probing so that the load balancer is not blocked and does not add latency to the server's request handling.
  (1) **Requests-In-Fight(RIF)**: Number of requests that are currently being processed by a server
  (2) **Latency Estimates**: how long a request is likely to take
- **Power of D Choices**: Similar to the power of two choices discussed earlier, Prequal uses the power of d choices where it will sample d servers such that d >= 2. The load balancer will evaluate each server's RIF and latency estimate and send the load to the server with the lowest estimated latency
- **Hot-Cold Lexicographic (HCL) Rule**: The HCL Rule allows Prequal to categorize servers based on the RIF and latency estimates. The two categories of servers are are "hot" (heavily loaded) and "cold" (lightly loaded).

4.5.2 Evaluation of Prequal. When Youtube started to switch many of their key services to use Prequal, there was great improvement in performances metrics where server replica error spikes of 0.01-0.1% were down to nearly zero and latency was reduced by 10-20% at the median and 40-50% at the tail. As the metrics show, Prequal is extremely great at reducing tail latency compared to other load balancing solutions. This means that the worst requests (the ones with a very large latency) in Prequal are much better than those in traditional load balancing algorithms. This divide is further exacerbated as the load is increased as shown in Figure 18.

Prequal is unique in the way that solves load balancing compared to other industry-level solutions. As in the title of the paper, the load is not directly balanced, but the key performance metrics (RIF

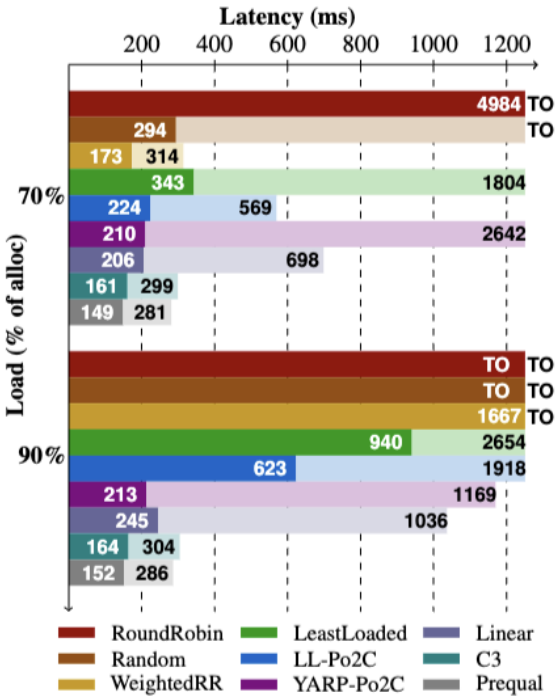


Fig. 18. Latency Comparison Across Different Load Balancing Solutions (Wydrowski, Kleinberg, Rumble, and Archer 2024)

and latency estimates) for each server are. Prequal is operated like other modern balancing solution using some variant of the power of 2 choices. As great as Prequal is, one of the big limitations is that it can only be applied to a hyper-scale system that Google has. It is not very adaptable for company's that want to build their own load-balancing solution but do not have the sheer infrastructure size of Google.

Since there is a constant need for new load-balancing solutions, the industry will continue to adapt to produce new and more optimal load-balancing solutions whether that is drawing inspiration from academia or companies building their own internal system. It will be interesting to see if and when the industry will shift away from using traditional load-balancing approaches and take a swing to use or develop a more modern solution.

5 Conclusion

As load-balancing algorithms and techniques continue to evolve and adapt, the development of each one has always been focused on solving the challenge of getting data to the end user in a high efficient, reliable, and secure manner. Looking back at the historical load balancing algorithms like the queuing models and consistent hashing not only shows how far developments in load balancing has come but also how rapidly distributed systems and the internet have evolved. It is important to remember that even though we can look down upon those historical approaches now, creating load-balancing algorithms has always been a response to user demand. We see the cat-and-mouse race of increasing user demand pressuring the need for industry professionals, researchers, and others to make breakthroughs in load balancing.

The transition from the historical and modern approaches is largely defined by the introduction of the Power of Two Choices. It probably remains as the greatest development in the load-balancing realm as we see numerous modern approaches across academia and the industry use the principle of the power of two choices in optimizing performance.

As distributed systems continue to scale, the need for more dynamic and smarter load-balancing strategies will grow. Today, cloud computing and data centers rely heavily on distributed architectures where load balancing ensures performance, reliability, and cost-efficiency. Whether in the industry or academia, the load-balancing problem can take up many forms and the solutions are even more diverse. The growth of data traffic and user demands will most likely never stop meaning improvements in load balancing will never stop.

References

Yuan-Chieh Chow et al. 1979. Models for dynamic load balancing in a heterogeneous multiple processor system. *IEEE Trans. Comput.* 100, 5 (1979), 354–361. https://ieeexplore.ieee.org/abstract/document/1675365

Guy Goren, Shay Vargaftik, and Yoram Moses. 2021. Stochastic coordination in heterogeneous load balancing systems. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. 403–414.

David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. 1997. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. 654–663.

Basetty Mallikarjuna and DAK Reddy. 2019. The role of load balancing algorithms in next generation of cloud computing. *Control Syst* 11 (2019), 20.

M. Mitzenmacher. 2001. The power of two choices in randomized load balancing. *IEEE Transactions on Parallel and Distributed Systems* 12, 10 (2001), 1094–1104.

Di Wu, Jikun Kang, Yi Tian Xu, Hang Li, Jimmy Li, Xi Chen, Dmitriy Rivkin, Michael Jenkin, Taeseop Lee, Intaik Park, Xue Liu, and Gregory Dudek. 2021. Load Balancing for Communication Networks via Data-Efficient Deep Reinforcement Learning. In *2021 IEEE Global Communications Conference (GLOBECOM)*. 01–07.

Bartek Wydrowski, Robert Kleinberg, Stephen M. Rumble, and Aaron Archer. 2024. Load is not what you should balance: Introducing Prequal. In *21st USENIX Symposium on Networked Systems Design and Implementation (NSDI 24)*. USENIX Association, 1285–1299.

Shu Yang, Xinze Wu, and Laizhong Cui. 2024. BCLB: A Scalable and Cooperative Layer-4 Load Balancer for Data Centers. In *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*. 993–1003.

[Chow et al. 1979] [Mallikarjuna and Reddy 2019] [Karger et al. 1997] [Goren et al. 2021] [Yang et al. 2024] [Wu et al. 2021] [Mitzenmacher 2001] [Wydrowski et al. 2024]