Emily Martins
Hal Nguyen
Yasmin Evans

CSC466
Spring 2025

Biweekly Update 1

Introduction

From February 9th to February 21st, we focused on doing an overview of three different areas: 1) server load-balancing (LB), (2) discovery channels, and (3) system architecture. We wanted to do an overview of these sections to get a better idea of LB strategies and distributed storage designs to help us determine a starting point for our project implementation. This update begins with a LB overview, followed by some information about discovery channels. We finish the update with some system architecture considerations and propose our system design we will use for our project.

Load-Balancing Overview

Server load-balancing (LB) works to distribute requests across servers [1]; a LB algorithm is used to determine which server handles a given request [1]. In a *static* LB algorithm, requests are distributed to servers based on a predetermined scheme, independent of the system's state [1]. In a *dynamic* LB algorithm, traffic is distributed taking into account the state of servers. In this section, we describe some initial research findings related to load-balancing (LB). We reviewed LB strategies from three different contexts: 1) industry, 2) open source software, and 3) academia.

LB Strategies Suggested by Google

The textbook *Site Reliability Engineering* [2] describes how Google runs production systems; the book includes several chapters on LB. While the chapters are not specifically targeted at server LB, we believed that the chapters were still a good resource. [2] describes several strategies to load-balance between data-centers for LB at the frontend: (a) DNS LB and (b) LB at the virtual IP address. For DNS LB, a simple strategy is returning a list of A records in a DNS reply then the client can select an address from the list. An issue with this approach includes the fact it may be difficult for a client to determine the closest address. This issue may be addressed by using an anycast address for the authoritative nameserver, whose DNS queries will go to the closest address [2]. That said, when a user requests a domain name, their computer sends the request to a recursive resolver which then queries the authoritative nameserver; thus, the authoritative nameserver may return an IP address close to the resolver's location, which might not be the close for the user [2]. The EDNS0 Client Subnet (ECS) extension [3] allows recursive resolvers to include part of the user's IP subnet in the DNS query sent to the authoritative nameserver which may help mitigate this issue; however, this solution is not standardized. That said, [2] explain that another level of frontend LB should follow DNS LB.

The next layer of frontend LB involves virtual IP addresses (VIPs) that are shared across multiple devices, whose implementation involves a network load balancer [2]. Network load balancers forward packets to backends for further processing. Google's network load balancer is Maglev [4]; it makes use of packet encapsulation (e.g., Generic Routing Encapsulation (GRE)). Thus, the network load balancer encapsulates an incoming packet into another IP packet with GRE. Then the backend can strip off the encapsulation and process the original inner packet to then send to the destination data center [2]. That said, the increase in packet size due to encapsulation is a limitation of this approach [2].

In *Site Reliability Engineering* [2]'s chapter on LB in the data-center, several LB policies are discussed including Simple Round Robin, Least-Loaded Round Robin, and Weighted Round Robin. [2] describes that when a stream of requests arrives at a data-center, there are server processes running on various machines to handle such requests; such processes are called *backend tasks*. For each incoming request, a *client task*, which makes requests to backend tasks, must select which backend task will handle the request. In Simple Round Robin, each client task sends requests to backend tasks in a round robin ordering. That said, this approach suffers from issues such as (a) unequal rate of requests per client, (b) unequal query costs (e.g., querying for all emails in a date range could be cheap or very costly), (c) differences in CPU power and storage on machines, etc [2]. In Least-Loaded Round Robin, each client task keeps track of the number of active requests sent to each backend task; it selects the one with the fewest active requests to query. Limitations of this scheme include the fact that (a) client tasks only know their own active requests (i.e. not including the requests of other client tasks), and (b) some backend tasks may be more efficient at processing requests [2]. In Weighted Round Robin, backend tasks share information such as query rates and CPU usage with client tasks; client tasks use this data to assign a "capability score" to each backend task, reflecting how much load it can handle [2]. Backends with a higher capability score receive more requests [2]. [2] indicate that in practice, Weighted Round Robin outperforms Simple Round Robin and Least-Loaded Round Robin.

Nginx LB

While Nginx is a web server, it can also be used as a load balancer [5]. Nginx open-source offers support for both HTTP LB and TCP/UDP LB [5].

Nginx open-source (i.e., not including the commercial subscription) supports the following LB algorithms for HTTP and TCP/UDP LB [6], [7]:

- **Round Robin:** This is the default algorithm. Requests/packets are sent to a list of servers starting going from the top of the list to the bottom, cycling back to the top (i.e. like a circular array). Server weights are also considered if present; the default server weight is 1.
- **Least Connections:** Requests/packets are sent to the server with the fewest number of active connections. Server weights are also considered if present; the default server weight is 1.
- **IP Hash:** This algorithm uses the first three octets of the client's IPv4 address (or entire IPv6 address) as a hash to determine to which server to send the client's requests/packets. In this case, the requests of a client always get sent to the same server unless the server becomes unavailable.
- **Hash:** The syntax for specifying this algorithm is `hash key [consistent]`. This algorithm creates client-server mappings via the given hash `key`. However, adding or removing servers can change the client-server mapping. If users specify the `consistent` option, then ketama [8] consistent hashing is used, which reduces the number of client-server remappings if the number of servers change.
- **Random:** In the simplest case, this LB algorithm randomly selects a server to send a request/packet to.

Strategies from Academia

For our academia example, we wanted to investigate different algorithms from those used by Google [2] and Nginx [6], [7]. We describe the Ant Colony Optimization (ACO) algorithm [9] and an algorithm based on reservation policy [10].

Assigning tasks to "unlimited computing resources" in cloud computing is an NP-hard problem [11]; thus, finding optimal solutions to such a problem is costly. Metaheuristic-based strategies find near optimal solutions in a more reasonable amount of time [11]. ACO is a metaheuristic-based algorithm [11]. [9] propose a cloud task scheduling algorithm based on ACO for assigning jobs to VMs, using makespan as their optimization metric. Makespan refers to the completion time of a user's final job; typically we seek to minimize the makespan of most users since they tend to desire fast task/workload completion [11]. ACO is based on the foraging behavior of real ants, which communicate via pheromones to find the shortest path between their nest and a food source [9]. Ants can follow the path to food sources by using the pheromones left by other aunts [9]. As the process continues, most ants choose the shortest path to food since such paths tend to have the highest concentration of pheromones [9]. [9] reduce ACO to scheduling tasks on VMs [9]. The algorithm starts with each ant randomly placed at VM. During an iteration of the algorithm, each ant will select a task t_i to run on VM v_j based on a probability function that depends on several variables such as the pheromone concentration on the path between task t_i and VM v_j, the expected execution time of task t_i on VM v_j, etc [9]. Once all ants have completed assigning their tasks to VMs, each ant updates the pheromone values on each path they used [9]. This procedure is repeated for a specified number of maximum iterations [9]. The minimum value of the objective function of all ants is selected as the best for the given iteration; the solution with the minimum of all solutions is considered the optimal solution [9]. In their experiments, [9] found that their ACO solution outperformed First Come First Serve and Round Robin algorithms. From the description of experiments, it is unclear whether the authors precomputed the optimal solution with ACO then compared the schedule returned by ACO to those of Round Robin and First Come First Serve [9]. That said, in the context of server LB, a clear limitation of ACO is that a system normally does not have all requests at once; specifically, users can make requests at their own time. Thus, it seems unrealistic that a real distributed system requiring LB would be able to run several iterations of the ACO to schedule tasks, since other incoming tasks could arrive once the ACO has already begun. While it seems like the ACO may not be useful for our project, it is always useful to understand and be aware of techniques that may not be useful for a research project.

[10] propose a server LB strategy based on a reservation policy; the goal is to distribute requests between replicated servers. In [10]'s approach, overloaded servers can reserve some capacity on other servers before redirecting requests to them. As a result of the reservation policy, overloaded servers redirect load to the closest servers that have a small load without overloading them [10]. [10] implement the policy in middleware—termed the resource broker. The resource broker is responsible for managing resource transfers between servers. Servers requiring additional capacity go through the resource broker to get additional resources from other lightly-loaded servers [10]. In their experiments, [10] compared the performance of their approach to various algorithms. Some examples with [10]'s corresponding definitions include:

- **Round Robin:** Overloaded servers rotate through a list of servers to which they redirect their traffic.

- **Round Robin with Asynchronous Alarm:** An extension of [10]'s Round Robin algorithm, where servers broadcast their load status, allowing servers to remove overloaded servers from their list of potential candidates to which they can distribute load.
- **Least Used:** Using information about the load of other servers received via broadcast, overloaded servers send their load to the least recently used server.

[10]'s results show that their LB strategy had a better average response time compared to other LB approaches. That being said, while [10]'s strategy reduced the number of dropped requests, some requests were still dropped.

Discovery Channeling

This section discusses DNS-based Service Discovery channeling, which is implemented frequently in real world applications, as outlined in the *DNS-Based Service Discovery* Standards document [12]. At the start of our research phase on February 9th, we decided it may be helpful to research this area to determine if it would be helpful for our project.

DNS-Based Service Discovery

The DNS-Based Service Discovery Protocol utilizes standard DNS queries to establish the network's services via connection details without prior configurations [12]. The document outlines DNS Service Discovery's ability to identify instances of services, remain consistent and query for instances of services [12]. It also elaborates on the mechanism's simplicity, denoting that any device simple enough to implement IP can run the mechanism [12]. A discovery protocol which utilizes internet protocol services may facilitate implementation, as no external applications would be necessary (unlike with API or Cloud based Service Discovery) while still achieving the same benefits of Service Discovery mechanisms.

It should be noted however that the DNS-Based Service Discovery mechanism does not include any additional security considerations for the services it requests access to and gains information from [12]. Security can be handled through the use of additional extensions [12] or by addressing the individual issues mentioned in the Data Accountability and System Security portion of this paper. For this same reason however privacy is also of concern in DNS-Based Service Discovery systems, user information may be available if networks are available to the public as noted in the *DNS-Based Service Discovery (DNS-SD) Privacy and Security Requirements* Standards document [13] which should be addressed if the system is to mimic a real world application of a network.

Dynamic Multiplexing

At the start of our research phase on February 9th, we thought our project would have required the use of dynamic multiplexing. Multiplexing mechanisms like TCP long connection multiplexing allows for comparable load balancing gains and a more stable system as outlined by [14] when compared to the lone use of discovery channelling making it an attractive possible addition to our project for the sake of optimized load balancing. TCP long connection multiplexing is a dynamic version of multiplexing wherein the server (node) with the best performance is selected to hold the longest connection with external nodes to reduce handshakes, limit recurring connections and circumvent IP limitations by funnelling multiple TCP requests through a single channel [14]. That being said, we quickly realized that, at this time, multiplexing is out of scope for our project.

System Overview

This section discusses some architecture inspiration for our project implementation. Based on our findings, we also propose our implementation design.

Architecture

To get some design ideas for our project implementation, the paper *An Improved Technique for Data Retrieval in Distributed Systems* outlines a generic architecture for a distributed storage system [15]. A similar pattern is observed in the InterPlanetary File System (IPFS) [16], [17]. This architecture consists of three key components: **Load Balancing Nodes**, **Query Nodes**, and **Data Clusters**, each with distinct responsibilities. We discuss each component in the following sections.

Load Balancing Nodes

Load Balancing Nodes serve as the entry point for users, the primary function is to distribute incoming requests among multiple Query Nodes [1], [18], for an evenly distributed workload and preventing any single node from becoming a bottleneck. This improves system scalability, reliability, availability, and performance [16].

Query Nodes

Query Nodes play a central role in node addressing. Their primary function is to establish the (P2P) connections between the users, and the involved Data Node [17]. Each Query node is responsible for

1. **Recursive Query Resolution**: When a request is received from the load balancer, a Query node initiates a recursive search across the data cluster to locate the required node [16], for either data retrieval or data storage.
2. **Address Forwarding**: Once the Data node in the cluster is located, a Query node will forward the address to the user, then a direct P2P connection will be established [16], minimizing the network overhead in the process of data transfer.

Data Nodes

The main function of a Data Node is to store and serve data, where each block of data is content-addressable. The secondary function, but equally important, of a Data node is to maintain a distributed hash table (DHT) for information keeping and routing [19]. In an ideal system of distributed storage, redundancy and consistency are implemented, however, this is out of scope for our implementation since our main focus is measuring the performance of various LB techniques. Under the normal condition (no failure), a Data node's operations can be defined as follow:

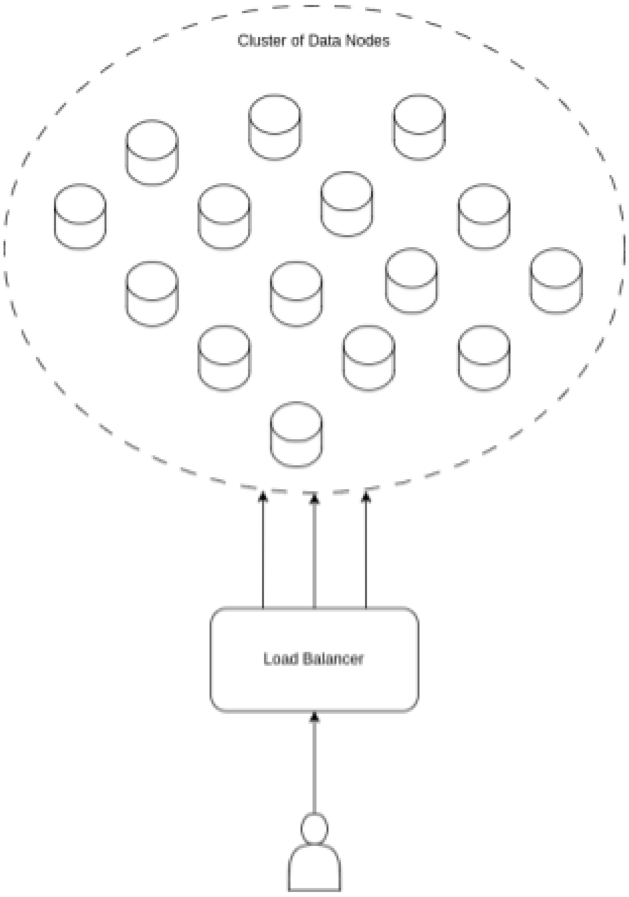
1. Upon receiving a query request from a Query Node, the Data Node checks whether it possesses the requested data block. If the data is available, the Data Node sends a confirmation message to the Query Node and waits for the user to establish a P2P connection for data transfer. If the Data

Node does not have the requested data, it forwards the query to the next node in the network [15], [16].

2. When a P2P connection is established with the user, the Data Node performs different actions based on the type of request:
    a. **Data retrieval request**: The node first verifies the proof of origin to ensure the user is authorized to access the data [20]. After successful validation, the file transfer process begins.
    b. **Data storage request**: For privacy purposes, ensuring that future access is restricted to authorized users only [21].

Project Implementation

For our implementation, we will take inspiration from IPFS [17] to derive our own design of a simplified system with two components: a **Load Balancing Node** and a **Data Cluster**. We also make the following assumptions: (a) all Data Nodes in the cluster maintain identical data and (b) we will only support data retrieval. This simplifies our scope and eliminates the need to implement data redundancy and consistency, allowing us to focus more on LB algorithms. The following figure is a visualization of our system architecture.



Implementation System Architecture

Data Accountability and System Security

The focus of our implementation is to gather quantitative data regarding the performance of the LB. That said, security is also important. At this time, we plan to assign data encryption as a responsibility of Data Nodes, incorporating this workload for performance evaluation, and assume the system is perfectly secured.

Data in the system is content-addressable, meaning each block of data undergoes an irreversible hashing function to produce a 256-bit digest, which serves as the unique addressing key for that block. The (optional) keyed hashing function BLAKE3 is chosen for its strong collision resistance property, performance, and the ability to generate arbitrary size digests [22].

To ensure privacy within the distributed system, the node encrypts the received data using a derived secret key. Additionally, it uses information unique to the data owner to cryptographically sign the data before storing it in its data storage [21]. The ChaCha20-Poly1305 Authenticated Encryption with Additional Data (AEAD) scheme meets this criteria, is fast in software, and will be selected [23].

That being said, since our implementation is focused on LB algorithms, we plan to focus on getting the LB portion of our project implemented prior to implementing—if time permits— the above security mechanisms.

Project Requirements

For our demonstration, we will be building three binaries: *user-interface*, *load-balancer*, and *data-node*. The requirements for each binary are listed below.

1. user-interface: send arbitrary amount of request, this parameter is a configurable setting.
2. load-balancing:
    a. Support multiple LB algorithms. We plan to start with Round Robin then select some dynamic LB algorithm such as Least Connections.
    b. Manage its internal states, as required by some LB algorithms.
3. data-node:
    a. Utilizes SQLite with a defined schema.
    b. All nodes in a cluster maintain identical data.
    c. P2P data transfer with the users.
    d. Configurable for optional encryption
    e. Periodically updates the LB with node state changes.
    f. Enable additional workload simulation using the `sleep` function.
    g. Upon leaving the cluster, the node communicates with the LB.

Conclusion

In summary, for the first two weeks, we did an overview of LB algorithms, discovery channels, and system architecture. We now have a basic design plan for our implementation. In the coming weeks, we plan to continue researching various LB algorithms, start our implementation, and research any other topics to support our project (e.g., implications of NAT).

References

[1] “What is load balancing? | How load balancers work.” Accessed: Feb. 19, 2025. [Online]. Available: https://www.cloudflare.com/learning/performance/what-is-load-balancing/

[2] J. P. Betsy Beyer Chris Jones, Niall Richard Murphy, *Site Reliability Engineering*. O’Reilly Media, Inc., 2024.

[3] C. Contavalli, W. van der Gaast, S. Leach, and E. P. Lewis, “Client Subnet in DNS Requests,” Internet Engineering Task Force, Internet-Draft draft-vandergaast-edns-client-subnet-02, Jul. 2013. [Online]. Available: https://datatracker.ietf.org/doc/draft-vandergaast-edns-client-subnet/02/

[4] D. E. Eisenbud *et al.*, “Maglev: A Fast and Reliable Software Network Load Balancer,” in *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, Santa Clara, CA, 2016, pp. 523–535.

[5] “nginx.” Accessed: Feb. 17, 2025. [Online]. Available: https://nginx.org/en/

[6] “Module ngx_stream_upstream_module.” Accessed: Feb. 17, 2025. [Online]. Available: https://nginx.org/en/docs/stream/ngx_stream_upstream_module.html

[7] “Module ngx_http_upstream_module.” Accessed: Feb. 17, 2025. [Online]. Available: https://nginx.org/en/docs/http/ngx_http_upstream_module.html

[8] “libketama: Consistent Hashing library for memcached clients,” Richard Jones. Accessed: Feb. 20, 2025. [Online]. Available: https://www.metabrew.com/article/libketama-consistent-hashing-algo-memcached-clients

[9] M. A. Tawfeek, A. El-Sisi, A. E. Keshk, and F. A. Torkey, “Cloud task scheduling based on ant colony optimization,” in *2013 8th International Conference on Computer Engineering & Systems (ICCES)*, Nov. 2013, pp. 64–69. doi: 10.1109/ICCES.2013.6707172.

[10] A. Nakai, E. Madeira, and L. E. Buzato, “On the Use of Resource Reservation for Web Services Load Balancing,” *J. Netw. Syst. Manag.*, vol. 23, no. 3, pp. 502–538, Jul. 2015, doi: 10.1007/s10922-014-9303-y.

[11] M. Kalra and S. Singh, “A review of metaheuristic scheduling techniques in cloud computing,” *Egypt. Inform. J.*, vol. 16, no. 3, pp. 275–295, Nov. 2015, doi: 10.1016/j.eij.2015.07.001.

[12] S. Cheshire and M. Krochmal, “DNS-Based Service Discovery.” Internet Engineering Task Force (IETF), Feb. 2013. Accessed: Feb. 20, 2025. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6763

[13] C. Huitema and D. Kaiser, “DNS-Based Service Discovery (DNS-SD) Privacy and Security Requirements.” Internet Engineering Steering Group (IESG), Sep. 2020. Accessed: Feb. 20, 2025. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8882.html#name-authors-addresses

[14] W. Li, J. Liang, X. Ma, B. Qin, and B. Liu, “A Dynamic Load Balancing Strategy Based on HAProxy and TCP Long Connection Multiplexing Technology,” in *Proceedings of the Fifth Euro-China Conference on Intelligent Data Analysis and Applications*, in Advances in Intelligent Systems and Computing, vol. 891. Springer, Cham, Dec. 2018, pp. 36–43. doi: https://doi.org/10.1007/978-3-030-03766-6_5.

[15] Ahmed Mateen, Qingsheng Zhu, Salman Afsar, and Javaria Maqsood, “An Improved Technique for Data Retrieval in Distributed Systems,” in *Proceedings of the International MultiConference of Engineers and Computer Scientists 2019*, Hong Kong, Mar. 2019, pp. 188–195. Accessed: Feb. 16, 2025. [Online]. Available: https://www.iaeng.org/publication/IMECS2019/IMECS2019_pp188-195.pdf

[16] D. Trautwein *et al.*, “Design and Evaluation of IPFS: A Storage Layer for the Decentralized Web,” in *Proceedings of the ACM SIGCOMM 2022 Conference*, ACM, Aug. 2022, pp. 739–752. doi: 10.1145/3544216.3544232.

[17] J. Benet, “IPFS - Content Addressed, Versioned, P2P File System,” 2014. [Online]. Available: https://arxiv.org/abs/1407.3561

[18] “Load balancing (computing).” IPFS. [Online]. Available: https://ipfs.io/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Load_bala

ncing_(computing).html

[19] S.Rajalakshmi, S.Balaji, and T.Godwin Selva Raja, “Routing Protocols of Distributed Hash Table Based Peer to Peer Networks,” *IOSR J. Comput. Eng.*, pp. 70–74, Jan. 2014.

[20] “Vulnerabilities and Threats in Distributed Systems,” geeksforgeeks.org. [Online]. Available: https://www.geeksforgeeks.org/vulnerabilities-and-threats-in-distributed-systems/?utm_source=chatgpt.com

[21] “Security in Distributed System,” geeksforgeeks.org. [Online]. Available: https://www.geeksforgeeks.org/security-in-distributed-system/?utm_source=chatgpt.com

[22] Jack O’Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O’Hearn, *BLAKE3, one function, fast everywhere*. (2021). Rust, C. [Online]. Available: https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf

[23] Y. Nir, “ChaCha20 and Poly1305 for IETF Protocols.” Internet Research Task Force, May 2015. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7539