Documentation

NexaLoak resolves issues in Virtual Private Networks (VPNs) related to server crashes and IP address bans by implementing the Raft consensus algorithm, which is designed for fault tolerance and performance. Raft achieves consensus through leader election, with a periodic process that selects a leader and followers in a distributed system. The algorithm ensures continuous communication checks among nodes, and in the event of a leader crash, a new leader is elected through a timed process. NexaLoak aims to enhance VPN services by decentralizing control and adapting communication processes between leaders and followers for managerial purposes. It's important to note that Raft is not a Byzantine fault-tolerant algorithm, relying on trust in the elected leader, necessitating server additions by authorized personnel.

Chapter 1: Definitions

Raft Algorithm

In the Raft consensus algorithm, which is used to manage distributed systems and ensure fault tolerance, there are three key roles:

Follower

- A follower is one of the possible states of a node in the Raft algorithm.
- Followers are passive nodes that simply listen to the leader and replicate its log entries.
- In normal operation, a majority of nodes in the Raft cluster are followers.

Candidate

- A candidate is a transitional state in the Raft algorithm.
- When a follower's election timeout elapses without receiving a valid heartbeat from the leader, it transitions to the candidate state and starts a new election.

- In this state, a node requests votes from other nodes in the cluster to become the new leader.

Leader

- The leader is a crucial role in the Raft algorithm.
- The leader is responsible for managing the replication of log entries to other nodes (followers) in the cluster.
- During normal operation, the leader sends heartbeat messages to followers to maintain its authority.

Virtual Private Network (VPN)

Briefly defined, A Virtual Private Network or VPN creates a secure connection between a computing device and a computer network. The benefits of a VPN include security, reduced costs for dedicated communication lines, and greater flexibility for remote workers.
To see more

Chapter 2: Consensus

Soon...

Chapter 3: System Architecture

To explain how a client will connect to our service, pay attention to Figure 1. First, the client tries to connect to the given URL. The default format of the URL is like `<scheme>://<user>:<password>@<host>:<port>/<path>;<params>?<query>#<frag>`. So, the user's authentication section of the URL will be before `@`, and the rest of the URL will be an address to which clients will connect.

Usually, hosts are domains that have an "A record" pointing to their IP address. This is the part that NexaLoak utilizes to prevent the user from connecting to a crashed or banned node. By the way, to get the IP address of the host, clients will query a DNS Resolver, and consequently, the resolver will respond with a list of IP addresses, which can be ordered randomly or in a meaningful way. Now, I'll clarify how NexaLoak works through several examples, each demonstrating the features and properties of the system.

Example 1: Sustainability

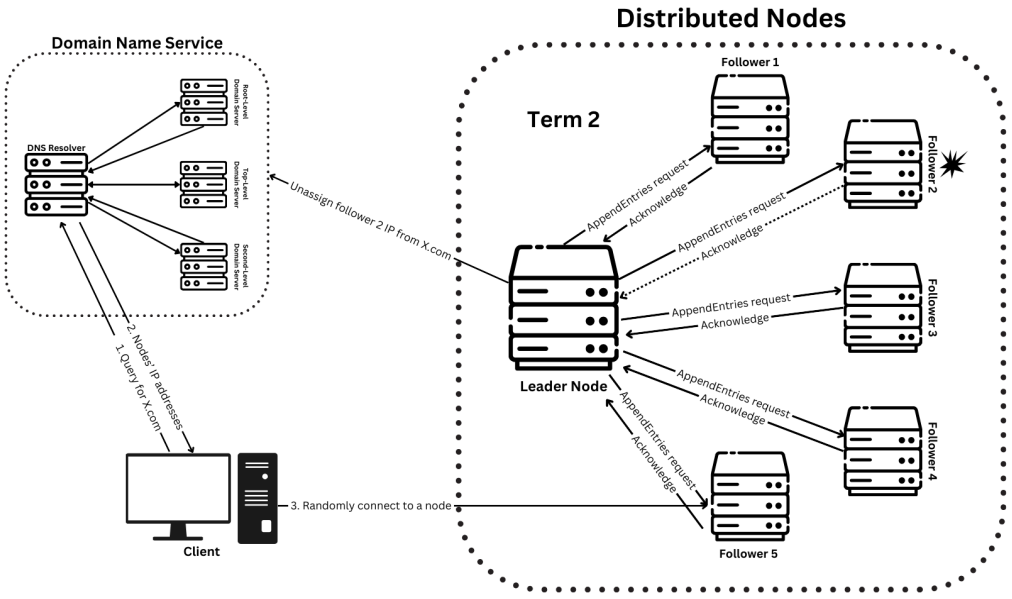

Figure 1

Suppose follower 2 crashes. The leader node discovers that follower 2 does not respond to consecutive messages. Therefore, it will send a request to the Domain Name Service and instruct it to "Unassign follower 2's IP address from our domain." Consequently, when a client connects to the service, the DNS Resolver will provide the client with 4 IPs out of the 5 available. The result will be a stable system where each node communicates with the service as effectively as possible.

This approach will be same for the problem of banning IP address. The only difference remains is the way leader figures out either a node is banned or not. In this case, the leader discovers this by calling an API that establishes a connection between the source country and the IP address. If the connection is successful, the API returns 1 as a response; otherwise, it returns 0. In this way, the leader detects anomalies and remove them by sending a request to the Domain Name Service, as I mentioned.

Example 2: Lacks of Resources

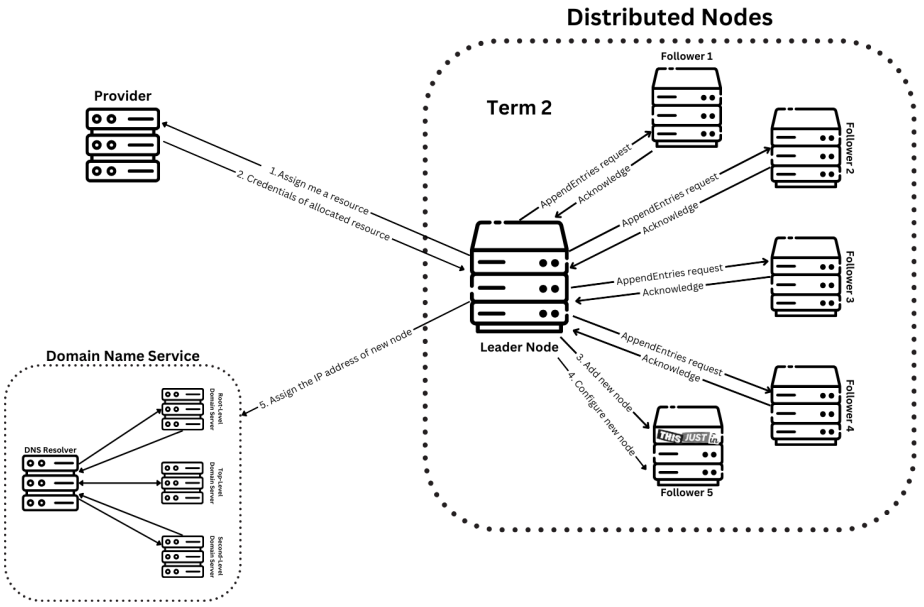

Figure 2

Removing nodes continuously faces the deficiency of resources in the future. To solve this problem, the leader must be able to add nodes when deciding to remove a resource due to failures or being filtered.

One of the advantages of the Raft algorithm is executing leader election. To clarify, the leader can add a new resource by invoking an API from the provider and configuring the new server using a bash script to serve the same as other nodes. Then, assign node's IP address to the domain. The figures below illustrate the flow of adding a new resource to our network. Checking for the quality of the IP address is the most important part of this process due to the filtering problem.

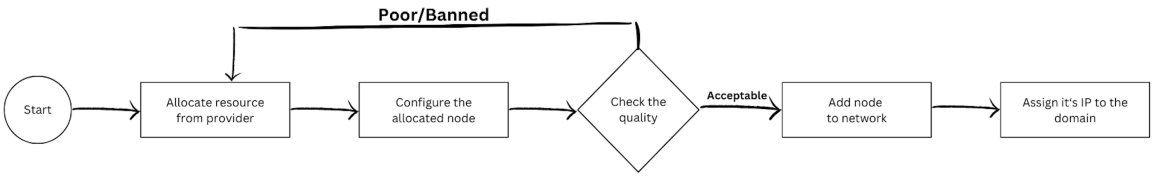

Figure 3

Example 3: Full Control

In order to obtain metrics that provide some insights, the leader offers an API that includes:

Metrics

- General status of the system, such as the current term, leader, and logs of available nodes
- Health condition of nodes (measured by the response time of each node)
- CPU and memory usage of each node
- Number of users connected simultaneously
- The traffic passed through the system in the past 30 days, 7 days, and 24 hours
- Number of banned IPs or crashed nodes during the past month

To provide and visualize these metrics, an interface needs to be integrated, which can be a Discord or Telegram bot. Maintaining the bot is one of the responsibilities of the leader. Therefore, the leader broadcasts the required information for maintenance, so if the leader dies, all nodes have the data needed for maintaining the bot, and the new leader will continuously support the bot.

Additionally, when the system serves a massive number of users, it should remain stable and persistent in terms of speed. The leader measures the throughput traffic of each server, and then the decision of adding a new resource can be made easily.

Chapter 4: Technologies

Programming Language

The programming language chosen for this project is GoLang, which has been developed by Google. The reason is GoLang is known for its simplicity and speed, making it easier for our team to develop and maintain the project. With its strong support for concurrency and clean syntax, GoLang aligns well with our goals of building a reliable and high-performance system.

Reliable In-memory Storage

Redis is popular for its reliability as an in-memory database, which is why it has been chosen for this project. It excels in providing fast and efficient data storage and retrieval, making it an ideal solution for scenarios where speed and responsiveness are crucial.

Remote Procedure Call

gRPC is used as the RPC (Remote Procedure Call) framework for this project, facilitating communication between nodes. The reason is that gRPC offers a streamlined and efficient way for nodes to exchange information. Its use of Protocol Buffers ensures a compact and fast data serialization, reducing the amount of data transferred over the network.

Top-level Proxy Protocol

V2Ray/VLESS is the proxy protocol of this project because V2Ray's focus on security and encryption ensures that our network communications remain private and secure. The ease of configuration and the active community

support further contribute to its selection as the proxy protocol, allowing for efficient and reliable proxying within our project infrastructure.

Chapter 5: Implementation

Soon...