

Tìm hiểu về Load Balancer - Bộ cân bằng tải

Nguyễn Thanh Hà

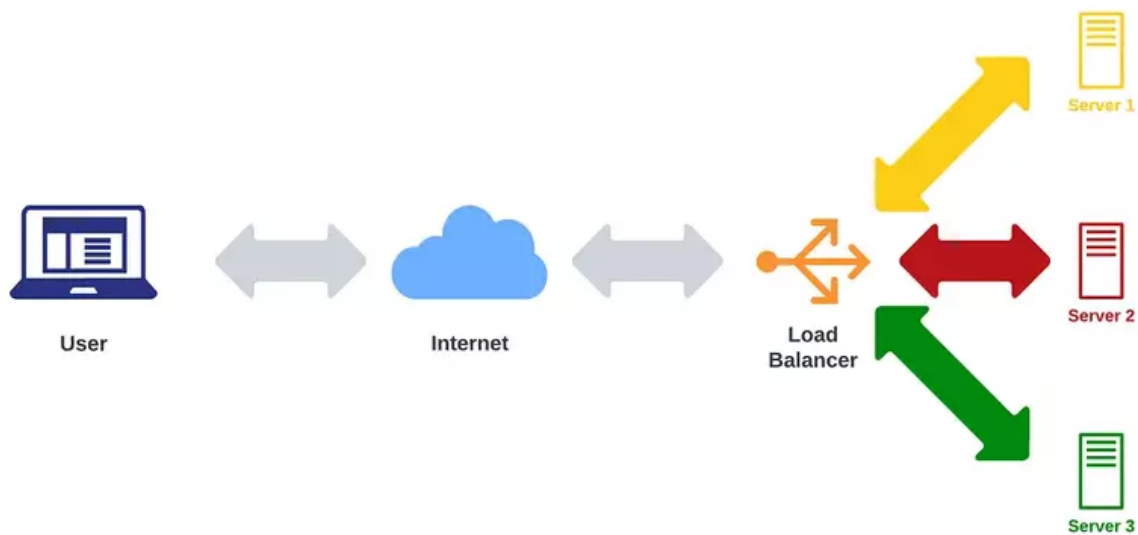
Mục lục

1.	<i>Giới thiệu về Load Balancer</i>	1
2.	<i>Các loại Load Balancer dựa trên mô hình OSI</i>	2
2.1.	Network Load Balancer (Load Balancer 4):	3
2.2.	Application Load Balancer (Load Balancer 7):	3
3.	<i>Các thuật toán Load Balancing</i>	4
3.1.	Round Robin	4
3.2.	Weight Round Robin	5
3.3.	IP Hash.....	6
3.4.	Least Connection Algorithm	7
3.5.	Least Response Time Algorithm.....	8
4.	<i>Xây dựng Load Balancer đơn giản với Go sử dụng giải thuật Round Robin</i>	10
4.1.	ReverseProxy	11
4.2.	Load Balancers	11
4.3.	Distribute Traffic	14
4.3.	Kiểm thử kết quả	17

1. Giới thiệu về Load Balancer

Load Balancer là một cân bằng tải đóng vai trò thực hiện điều hướng yêu cầu của người dùng tới một trong các máy chủ phía sau nó.

Trong một hệ thống website lớn với hàng triệu người dùng thì chỉ một máy chủ không thể nào xử lý được toàn bộ yêu cầu của người dùng, do đó ta cần phải chạy nhiều máy chủ cùng một lúc, và load balancers sẽ đứng ở đằng trước các máy chủ này để hứng yêu cầu của người dùng và điều hướng yêu cầu đó tới các máy chủ phía sau nó.



Nhờ vậy, Bộ cân bằng tải sẽ giúp đạt được các mục đích:

- Tính hiệu quả nhờ khả năng giảm lưu lượng trên mỗi máy chủ. Giảm thiểu thời gian phản hồi của server, tình trạng chậm và các thông báo lỗi khó chịu, đảm bảo sự trải nghiệm mượt mà cho người dùng.
- Khả năng mở rộng do bộ cân bằng tải có thể thay đổi quy mô cơ sở hạ tầng theo yêu cầu mà không ảnh hưởng đến dịch vụ.

Nếu tại một thời điểm, một máy chủ ngừng hoạt động do xảy ra lỗi thì những máy chủ khác có thể thay nó xử lý những yêu cầu để không làm gián đoạn tới dịch vụ.

2. Các loại Load Balancer dựa trên mô hình OSI



Bộ cân bằng tải được phân loại dựa trên tầng của mô hình OSI mà chúng hoạt động. Nó được phân ra thành các loại như sau:

2.1. Network Load Balancer (Load Balancer 4):

Bộ cân bằng tải Tầng 4 hoạt động ở tầng vận chuyển (Tầng 4) của mô hình OSI. Nó đưa ra quyết định dựa trên thông tin như địa chỉ IP nguồn và đích, cũng như số cổng của các yêu cầu đến. Mục tiêu chính của Bộ cân bằng tải Tầng 4 là phân phối lưu lượng mạng hiệu quả trên nhiều máy chủ.

- Cách hoạt động:

Khi người dùng gửi yêu cầu để truy cập một trang web hoặc ứng dụng, Bộ cân bằng tải Tầng 4 nhận yêu cầu. Sau đó, nó xem xét dữ liệu tầng vận chuyển (địa chỉ IP và port) để xác định máy chủ nào nên xử lý yêu cầu. Bộ cân bằng tải sử dụng các thuật toán khác nhau (ví dụ: round-robin, least connections) để quyết định máy chủ tốt nhất để chuyển tiếp yêu cầu. Điều này đảm bảo lưu lượng được phân phối đều giữa các máy chủ, cải thiện hiệu suất và tránh quá tải bất kỳ máy chủ nào.

- Ưu điểm:

1. Hiệu suất cao và độ trễ thấp do quyết định ở tầng mạng.
2. Thích hợp để phân phối lưu lượng TCP và UDP một cách hiệu quả.
3. Lý tưởng cho các tình huống không cần quyết định dựa trên nội dung.

- Nhược điểm: Hạn chế trong việc đưa ra quyết định cụ thể cho ứng dụng

2.2. Application Load Balancer (Load Balancer 7):

Bộ cân bằng tải Tầng 7 hoạt động ở tầng ứng dụng (Tầng 7) của mô hình OSI. Nó có thể đưa ra quyết định thông minh hơn dựa trên dữ liệu cụ thể của ứng dụng, như HTTP Headers, cookie và URLs. Bộ cân bằng tải Tầng 7 hiểu các giao thức ứng dụng, cho phép chúng tối ưu hóa phân phối lưu lượng cho các ứng dụng hoặc dịch vụ cụ thể.

- Cách hoạt động:

Khi người dùng gửi yêu cầu, Bộ cân bằng tải Tầng 7 phân tích nội dung của yêu cầu để hiểu rõ hơn về ứng dụng đang được truy cập. Ví dụ, nó có thể xác định loại dịch vụ (ví dụ: HTTP, HTTPS, FTP) hoặc URL cụ thể đang được yêu cầu. Sử dụng thông tin này, bộ cân bằng tải có thể đưa ra quyết định thông minh về máy chủ nào phù hợp nhất để xử lý yêu cầu. Điều này cho phép các chiến lược cân bằng

tải tiên tiến hơn, chẳng hạn như gửi các yêu cầu nhất định đến các máy chủ chuyên biệt có thể xử lý các tác vụ ứng dụng cụ thể.

- Ưu điểm:

1. Nhận biết ứng dụng và có thể tối ưu hóa lưu lượng dựa trên yêu cầu cụ thể của ứng dụng.
2. Cho phép định tuyến dựa trên nội dung và thuật toán cân bằng tải tiên tiến.
3. Thích hợp cho các ứng dụng phức tạp yêu cầu các phản hồi máy chủ khác nhau dựa trên loại yêu cầu.

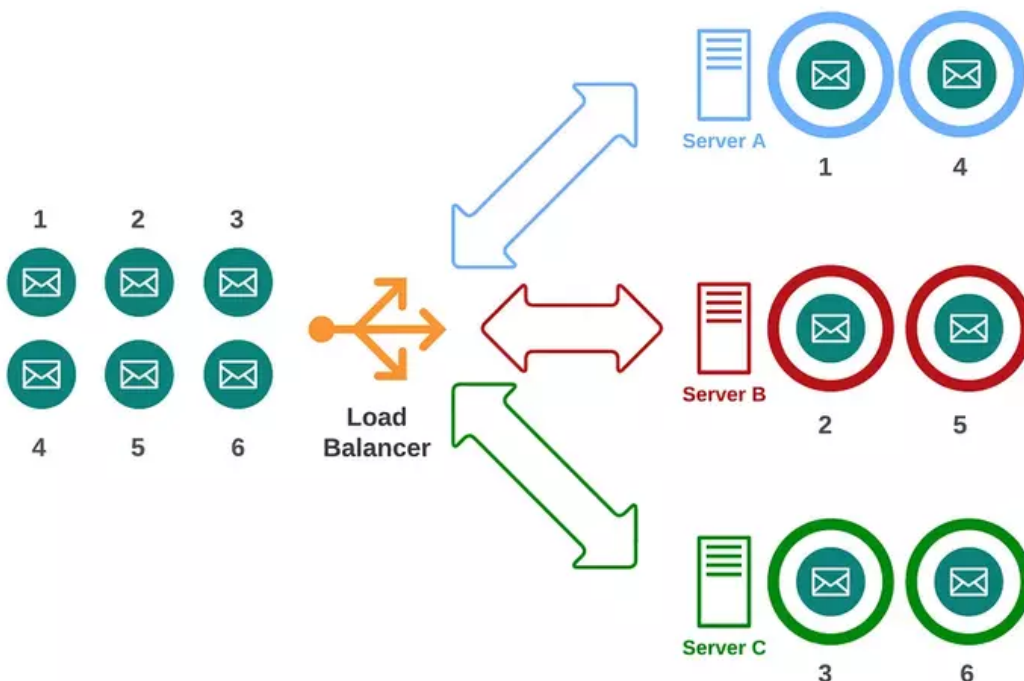
- Nhược điểm: Overhead xử lý cao hơn so với Bộ cân bằng tải Tầng 4 do phải phân tích nội dung.

3. Các thuật toán Load Balancing

3.1. Round Robin

- Khái niệm:

Thuật toán cân bằng tải Round Robin là một trong những phương pháp đơn giản và trực quan nhất được sử dụng để phân phối lưu lượng mạng đến từ nhiều máy chủ. Nó hoạt động ở tầng vận chuyển (Tầng 4) của mô hình OSI và thường được sử dụng trong nhiều bộ cân bằng tải.

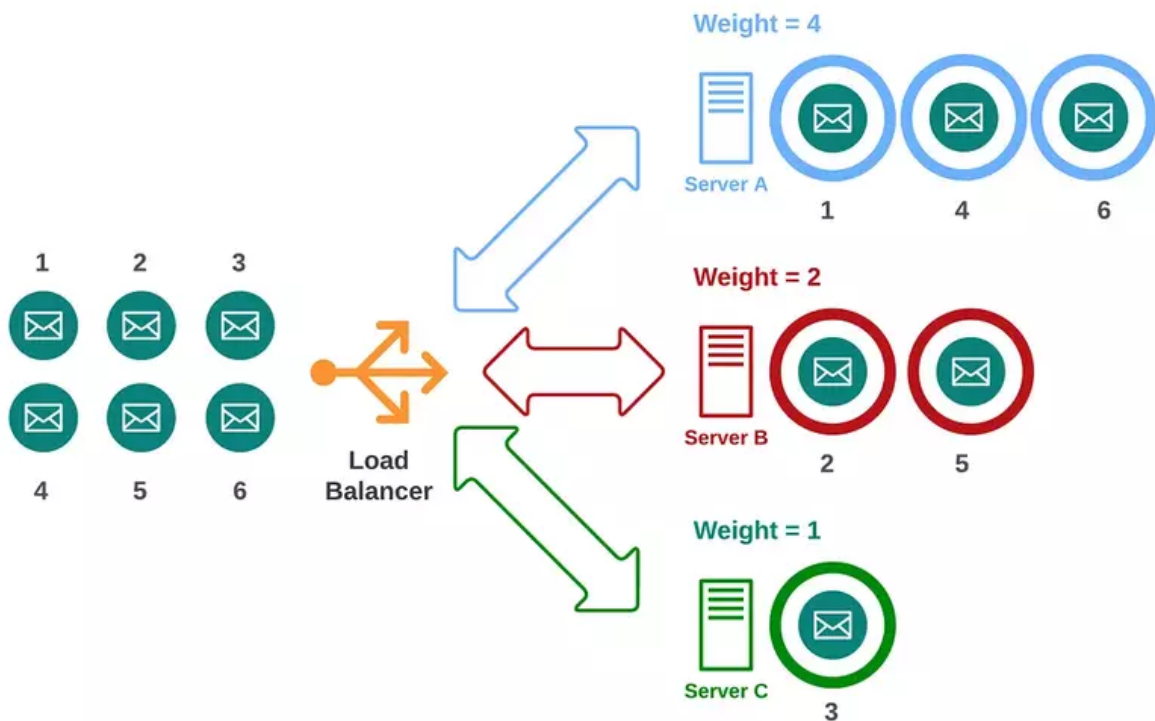


- Ưu điểm:
 1. Đơn giản: Round Robin là một trong những thuật toán cân bằng tải đơn giản nhất để triển khai. Nó không yêu cầu các phép tính phức tạp hay giám sát máy chủ sâu sắc.
 2. Công việc đều: Thuật toán đảm bảo phân phối đều các yêu cầu đến từ tất cả các máy chủ trong nhóm. Mỗi máy chủ nhận được một phần tải đều, thúc đẩy việc sử dụng tài nguyên công bằng.
- Nhược điểm:
 1. Công suất máy chủ không bằng nhau: Round Robin xem xét tất cả các máy chủ là như nhau, bất kể công suất hoặc khả năng hiệu suất của chúng. Điều này có thể dẫn đến việc một số máy chủ bị quá tải nếu chúng kém mạnh mẽ hơn các máy chủ khác trong nhóm.
 2. Không có giám sát tình trạng: Thuật toán thiếu thông minh để giám sát tình trạng hoặc khả năng phản hồi của máy chủ. Nếu một máy chủ trở nên không khả dụng hoặc gặp vấn đề về hiệu suất, Round Robin vẫn tiếp tục chuyển tiếp yêu cầu đến máy chủ đó.

3.2. Weight Round Robin

- Khái niệm:

Thuật toán cân bằng tải Round Robin có trọng số là một sự mở rộng của thuật toán Round Robin cơ bản. Nó nhằm giải quyết vấn đề về công suất máy chủ không đồng đều bằng cách gán các trọng số khác nhau cho mỗi máy chủ trong nhóm. Những trọng số này đại diện cho công suất hoặc hiệu suất tương đối của mỗi máy chủ. Máy chủ có trọng số cao hơn nhận được một phần lớn hơn của lưu lượng đến, trong khi máy chủ có trọng số thấp hơn nhận được một phần nhỏ hơn.



- Ưu điểm:

1. Phân phối dựa trên công suất: Round Robin có trọng số tính đến các công suất khác nhau của máy chủ, đảm bảo rằng các máy chủ mạnh mẽ hơn xử lý nhiều hơn.
2. Cân bằng tinh chỉnh: Khả năng đặt các trọng số cụ thể cho phép điều chỉnh chính xác sự phân phối tải, đáp ứng các máy chủ có khả năng khác nhau.

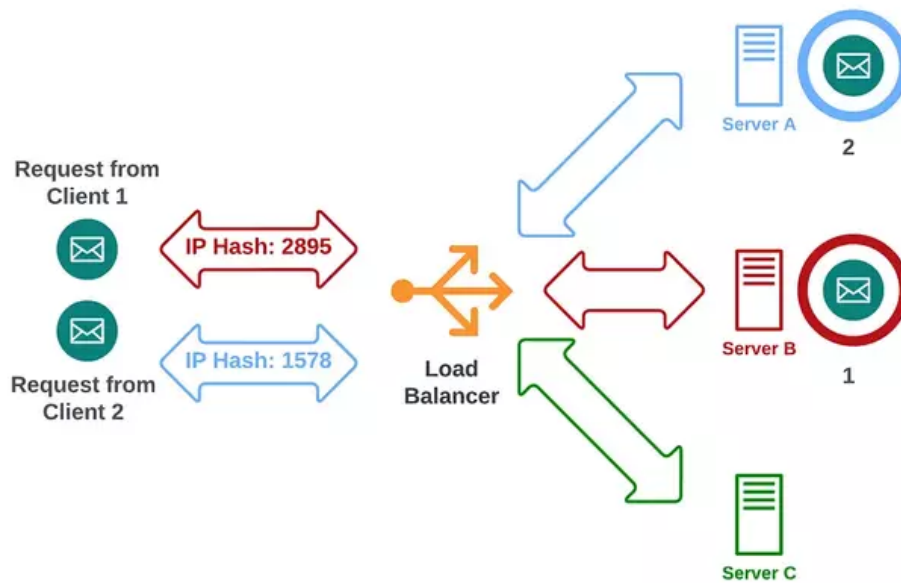
- Nhược điểm:

1. Cấu hình tĩnh: Yêu cầu cấu hình trọng số máy chủ theo cách thủ công. Nó có thể không thích ứng một cách linh hoạt với các thay đổi về hiệu suất hoặc công suất máy chủ.
2. Phức tạp: Thuật toán đặt ra một số sự phức tạp so với Round Robin cơ bản, vì người quản trị cần phải gán và quản lý các trọng số một cách phù hợp.

3.3. IP Hash

- Khái niệm:

Thuật toán cân bằng tải IP Hash là một phương pháp được sử dụng để phân phối lưu lượng mạng đến từ nhiều máy chủ dựa trên địa chỉ IP nguồn của khách hàng. Thuật toán này hoạt động ở tầng ứng dụng (Tầng 7) của mô hình OSI.



- Ưu điểm:

1. Duy trì Session: IP Hash cung cấp sự liên kết session, đảm bảo rằng các yêu cầu tiếp theo từ cùng một khách hàng luôn được chuyển hướng đến cùng một máy chủ. Điều này rất quan trọng đối với các ứng dụng phụ thuộc vào việc duy trì session người dùng hoặc trạng thái.
2. Tải Đều cho mỗi Khách hàng: Các yêu cầu của mỗi khách hàng đều được gửi đến cùng một máy chủ một cách nhất quán, đạt được sự phân phối tải cân đối cho từng khách hàng.

- Nhược điểm:

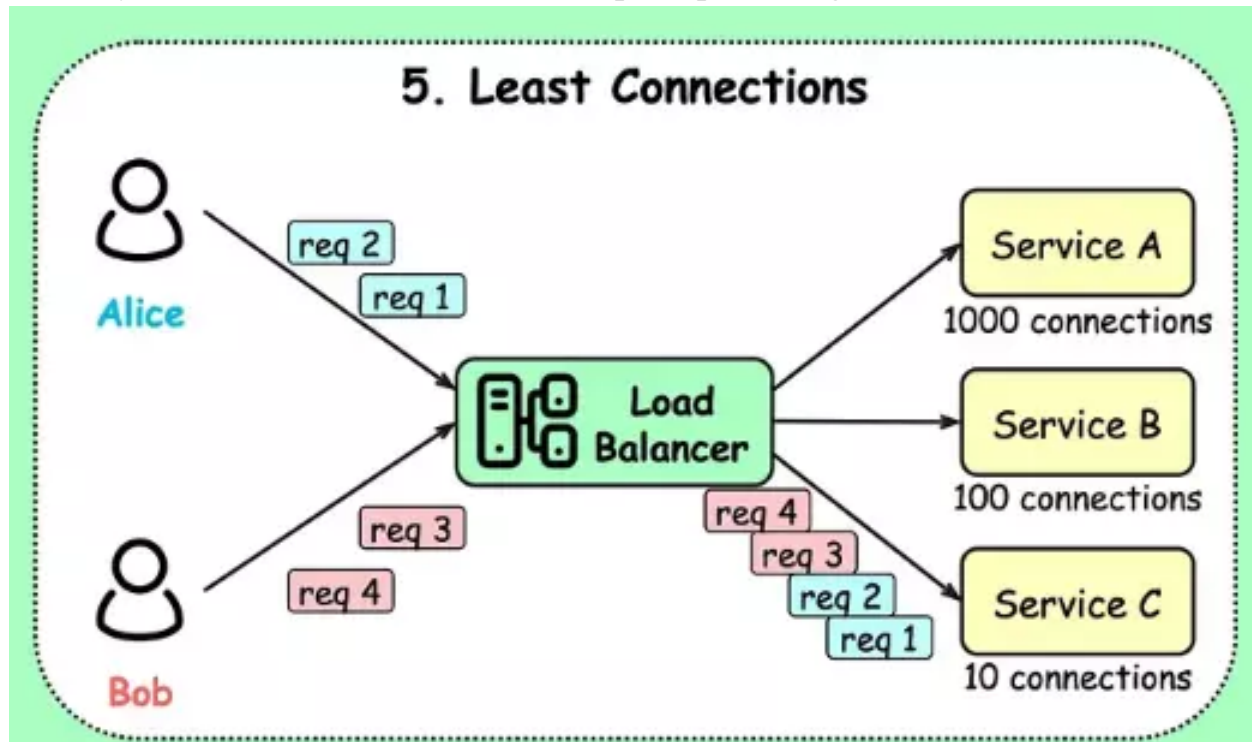
1. Tải công việc không đồng đều: Nếu một số khách hàng tạo ra một lượng lưu lượng lớn không tỷ lệ, các máy chủ tương ứng có thể gặp phải tải công việc cao hơn, có thể gây ra mất cân đối.
2. Phụ thuộc vào Khách hàng: IP Hash phụ thuộc nặng nề vào địa chỉ IP của khách hàng để ánh xạ, khiến nó kém hiệu quả khi khách hàng sử dụng nhiều địa chỉ IP.

3.4. Least Connection Algorithm

- Khái niệm:

Thuật toán cân bằng tải Least Connection được thiết kế để phân phối lưu lượng mạng đến từ nhiều máy chủ dựa trên số lượng kết nối đang hoạt động mà

mỗi máy chủ hiện có. Mục tiêu chính của thuật toán này là hướng các yêu cầu mới đến máy chủ có ít kết nối nhất, đảm bảo phân phối công việc cân đối hơn.



- Ưu điểm:

1. Phân phối tải động: Thuật toán Least Connection thích ứng theo thời gian thực với tải công việc thực tế trên mỗi máy chủ, đảm bảo rằng các yêu cầu mới được hướng đến máy chủ ít bận rộn nhất tại thời điểm đó.
2. Sử dụng tài nguyên cân đối: Bằng cách phân phối các kết nối đều qua các máy chủ, Least Connection giúp ngăn chặn bất kỳ máy chủ nào trở nên quá tải, do đó tối ưu hóa việc sử dụng tài nguyên.

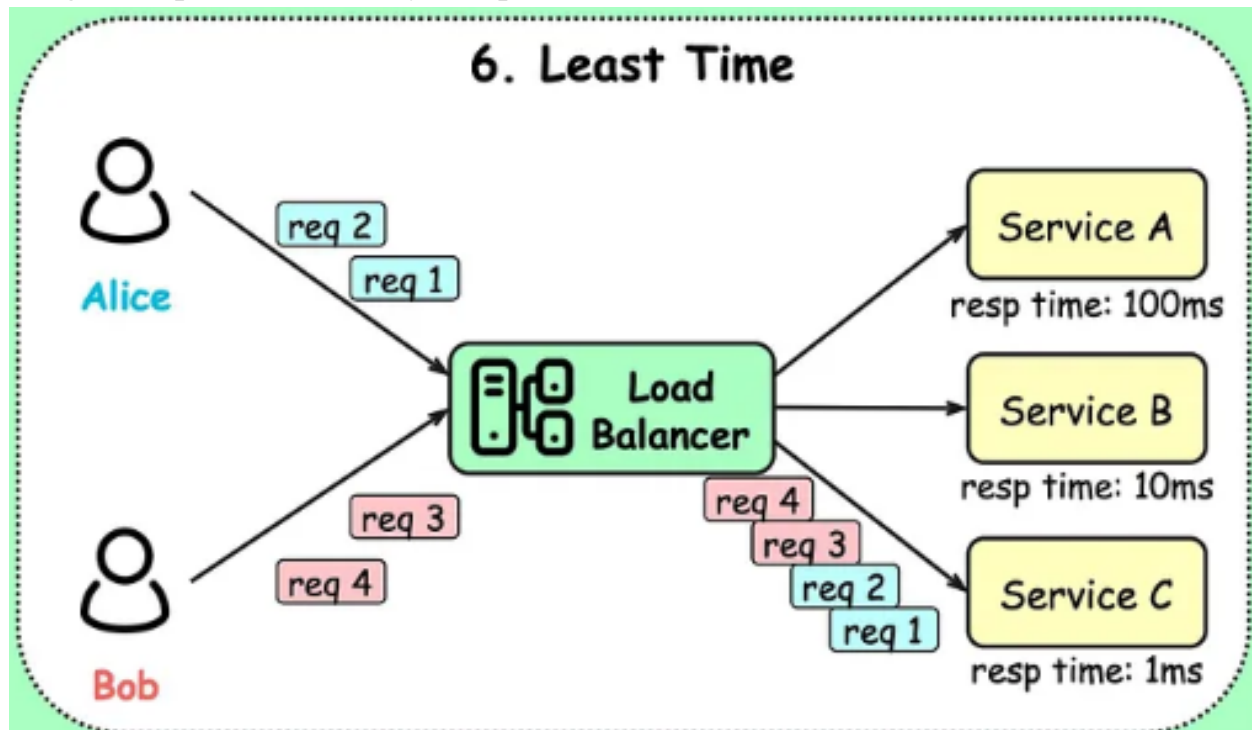
- Nhược điểm:

1. Biến động kết nối: Thuật toán Least Connection có thể gây ra biến động thường xuyên trong các kết nối cho một số máy chủ.
2. Xử lý đột biến: Trong những đợt tăng đột biến lưu lượng, các máy chủ ban đầu có ít kết nối hơn có thể nhận được một lượng lớn yêu cầu mới, có thể gây ra vấn đề hiệu suất tạm thời.

3.5. Least Response Time Algorithm

- Khái niệm:

Thuật toán cân bằng tải Least Response Time là một phương pháp được sử dụng để phân phối lưu lượng mạng đến từ nhiều máy chủ dựa trên thời gian phản hồi của chúng đối với các yêu cầu trước đó. Mục tiêu chính của thuật toán này là hướng các yêu cầu mới đến máy chủ có thời gian phản hồi ngắn nhất, đảm bảo rằng khách hàng được phục vụ bởi máy chủ phản hồi nhanh nhất có sẵn.



- Ưu điểm:

1. Trải nghiệm người dùng tối ưu: Thuật toán Least Response Time đảm bảo rằng khách hàng được hướng đến máy chủ có thể phản hồi nhanh nhất, dẫn đến trải nghiệm người dùng mượt mà.
2. Thích ứng động: Bộ cân bằng tải liên tục đánh giá thời gian phản hồi máy chủ và điều chỉnh quyết định định tuyến của mình phù hợp, làm cho nó phù hợp với môi trường động với hiệu suất máy chủ thay đổi.

- Nhược điểm:

1. Biến động thời gian phản hồi: Nếu thời gian phản hồi máy chủ biến đổi thường xuyên, bộ cân bằng tải có thể chuyển khách hàng giữa các máy chủ thường xuyên, có thể gây ra sự không nhất quán trong trải nghiệm người dùng.

2. Dễ bị tác động bởi ngoại lệ: Một đợt tăng đột biến trong thời gian phản hồi trên một máy chủ có thể dẫn đến việc tạm thời định tuyến nhiều lưu lượng hơn đến một máy chủ chậm hơn, ảnh hưởng đến hiệu suất tổng thể.
3. Phụ thuộc vào giám sát: Thuật toán Least Response Time yêu cầu giám sát liên tục thời gian phản hồi máy chủ.

4. Xây dựng Load Balancer đơn giản với Go sử dụng giải thuật Round Robin

Tạo 1 file **main.go** với đoạn code như sau:

```
1  package main
2
3  import (
4      "net/http/httputil"
5      "net/url"
6  )
7
8  type Backend struct {
9      URL          *url.URL
10     Alive        bool
11     ReverseProxy *httputil.ReverseProxy
12 }
13
14 type ServerPool struct {
15     backends []*Backend
16     current  uint64
17 }
18
19 func main() {
20
21 }
```

Ta khai báo hai kiểu dữ liệu struct là Backend và ServerPool.

Backend struct dùng để định nghĩa các server của ta, bao gồm ba thuộc tính:

- URL để định nghĩa địa chỉ của server, ví dụ localhost:8080.
- Alive để đánh dấu server còn sống hay không.
- ReverseProxy.

ServerPool struct dùng để lưu trữ các server mà Load Balancer sẽ điều hướng tới, bao gồm hai thuộc tính backends dùng để lưu server và current dùng để định nghĩa thứ tự server mà LB sẽ gửi request tới.

4.1. ReverseProxy

Định nghĩa của Go:

“ReverseProxy is an HTTP Handler that takes an incoming request and sends it to another server, proxying the response back to the client.”

Dịch ra tiếng việt đơn giản thì ReverseProxy sẽ nhận request của người dùng và gửi nó tới một server khác, sau đó nó sẽ điều hướng kết quả trả từ server đó về cho người dùng, ví dụ:

```
1 func main() {
2     u, _ := url.Parse("http://localhost:8080")
3     rp := httputil.NewSingleHostReverseProxy(u)
4
5     // initialize your server and add this as handler
6     http.ListenAndServe(":3000", http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
7         rp.ServeHTTP(w, r)
8     }))
9 }
```

Ở đoạn code trên ta chạy một server ở port 3000, và khi ta gọi vào localhost:3000 thì request sẽ được dẫn tới localhost:8080

4.2. Load Balancers

Quay lại file **main.go**, ta cập nhật thêm hàm để thêm server vào trong LB.



```
1 // AddBackend to the server pool
2 func (s *ServerPool) AddBackend(backend *Backend) {
3     s.backends = append(s.backends, backend)
4 }
```

Hàm `AddBackend()` đơn giản ta chỉ cần dùng hàm `append` của Go để thêm một server vào thuộc tính `backends` của `ServerPool`.

Tiếp theo ta thêm vào đoạn code sau ở `main()` để ta có thể chọn những server mà ta muốn LB sẽ điều hướng request tới.



```
1 func main() {
2     var serverList string
3     var port int
4     flag.StringVar(&serverList, "backends", "", "Load balanced backends, use commas to separate")
5     flag.IntVar(&port, "port", 3000, "Port to serve")
6     flag.Parse()
7
8     if len(serverList) == 0 {
9         log.Fatal("Please provide one or more backends to load balance")
10    }
11
12    servers := strings.Split(serverList, ",")
13 }
```

Ta dùng `flag.StringVar` để đọc các biến truyền vào từ terminal khi ta chạy chương trình Go và gán nó vào biến `serverList`, sau đó ta dùng hàm `strings.Split()` để tách biến `serverList` từ chuỗi thành một mảng các server, ví dụ khi ta chạy chương trình như sau.



```
1 go run main.go --backends=http://localhost:3031,http://localhost:3032,http://localhost:3033
```

Biến `serverList` sẽ là

<http://localhost:3031,http://localhost:3032,http://localhost:3033>

Chuyển thành mảng các servers

[http://localhost:3031, http://localhost:3032, http://localhost:3033]

```
PROBLEMS OUTPUT TERMINAL PORTS GITLENS DEBUG CONSOLE
hant@hant lb % go run main.go --backends=http://localhost:3031,http://localhost:3032,http://localhost:3033
2024/05/21 17:04:33 Load balancing to servers: [http://localhost:3031 http://localhost:3032 http://localhost:3033]
hant@hant lb %
```

Tiếp theo ta thêm các server này vào ServerPool

```
1 func main() {
2     var serverList string
3     var port int
4     flag.StringVar(&serverList, "backends", "", "Load balanced backends, use commas to separate")
5     flag.IntVar(&port, "port", 3000, "Port to serve")
6     flag.Parse()
7
8     if len(serverList) == 0 {
9         log.Fatal("Please provide one or more backends to load balance")
10    }
11
12    servers := strings.Split(serverList, ",")
13    // log.Printf("Load balancing to servers: %v", servers)
14
15    serverPool := ServerPool{current: -1}
16    for _, s := range servers {
17        serverUrl, err := url.Parse(s)
18
19        if err != nil {
20            log.Fatal(err)
21        }
22
23        proxy := httputil.NewSingleHostReverseProxy(serverUrl)
24        serverPool.AddBackend(&Backend{
25            URL:         serverUrl,
26            Alive:       true,
27            ReverseProxy: proxy,
28        })
29    }
30 }
```

Lúc này thì ta đã thêm được các server vào load balancers, bây giờ ta cần phải thực hiện gửi request lần lượt tới các server theo thứ tự.

4.3. Distribute Traffic

Để gửi request lần lượt tới từng server, ta cần có hàm lấy được server hiện tại và gửi request tới nó.

```
1 func (s *ServerPool) AddBackend(backend *Backend) {
2     s.backends = append(s.backends, backend)
3 }
4
5 func (s *ServerPool) NextIndex() int64 {
6     s.current++
7     return s.current % int64(len(s.backends))
8 }
9
10 func (s *ServerPool) GetNextBackend() *Backend {
11     next := s.NextIndex()
12     return s.backends[next]
13 }
```

Ta dùng hàm `GetNextBackend()` để lấy ra server mà ta muốn LB gửi request tới nó, ở trong hàm `GetNextBackend` ta sẽ dùng `s.NextIndex()` để lấy ra thứ tự của server tiếp theo và trả về server với thứ tự tương ứng nằm trong thuộc tính `backends`.

Ở hàm `NextIndex()` thì để lấy được thứ tự của server thì đầu tiên là sẽ tăng thuộc tính `current` lên 1 và tiếp đó ta sẽ lấy kết quả của phép chia dư `s.current % int64(len(s.backends))`.

Ở trên ta đã nói là thuật toán round robin sẽ lần lượt gửi request tới từng server và sau đó quay lại từ đầu, ta có thể thực hiện việc đó với phép chia dư, ví dụ ở trên thuộc tính `backends` có 3 server.

<http://localhost:3031>, <http://localhost:3032>, <http://localhost:3033>

Và ta khai báo `serverPool` với giá trị `current` là -1.

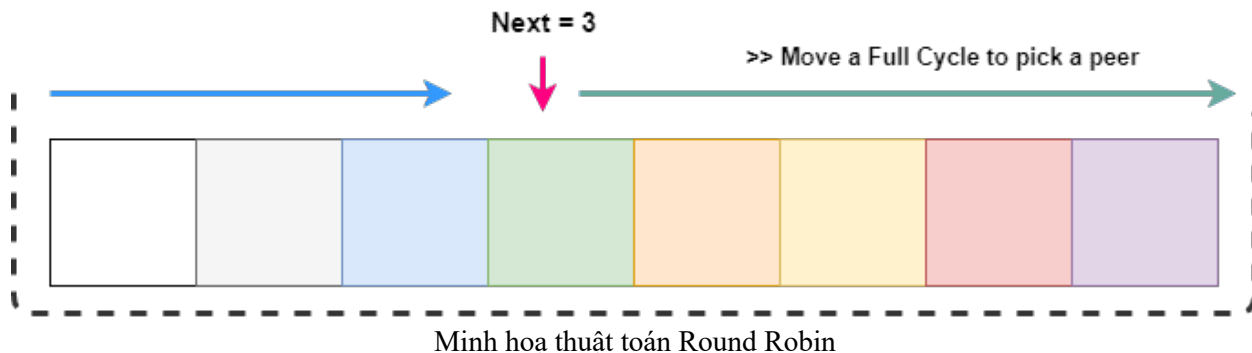


```
1 serverPool := ServerPool{current: -1}
```

Khi ta gọi hàm NextIndex sẽ như sau



```
1 // len(s.backends) is 3
2
3 s.current++ // current is 0
4 current % len(s.backends) // 0
5
6 s.current++ // current is 1
7 current % len(s.backends) // 1
8
9 s.current++ // current is 2
10 current % len(s.backends) // 2
11
12 s.current++ // current is 3
13 current % len(s.backends) // 0
14
15 s.current++ // current is 4
16 current % len(s.backends) // 1
17
18 s.current++ // current is 5
19 current % len(s.backends) // 2
20
21 s.current++ // current is 6
22 current % len(s.backends) // 0
```



Với chia lấy phần dư thì kết quả thứ tự của ta luôn đi từ 0 tới 2 sau đó quay lại 0, ta sẽ thực hiện round robin bằng cách chia lấy phần dư như trên.

Sau khi có được hàm lấy được thứ tự của server mà LB sẽ gửi request tới, ta cập nhật lại hàm **main()** như sau.

```

1  server := http.Server{
2      Addr: fmt.Sprintf(":%d", port),
3      Handler: http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
4          peer := serverPool.GetNextBackend()
5
6          if peer != nil {
7              peer.ReverseProxy.ServeHTTP(w, r)
8              return
9          }
10
11          http.Error(w, "Service not available", http.StatusServiceUnavailable)
12      }),
13  }
14
15  log.Printf("Load Balancer started at :%d\n", port)
16  if err := server.ListenAndServe(); err != nil {
17      log.Fatal(err)
18  }

```

Ở hàm để xử lý request, thì ta sẽ dùng `serverPool.GetNextBackend()` để lấy ra server ta muốn gửi request tới, sau đó ta sẽ dùng `peer.ReverseProxy.ServeHTTP(w, r)` để điều hướng request từ người dùng tới server của ta.

4.3. Kiểm thử kết quả

Giả sử ta có 3 server tại các địa chỉ sau: <https://server1.com>, <http://server2.com>, <http://server3.com>. Nhiệm vụ của Load Balancer là điều phối các request từ địa chỉ localhost:3000 đến lần lượt các địa chỉ trên theo đúng thuật toán Round Robin.

go run main.go --

backends=https://server1.com,http://server2.com,http://server3.com

```
PROBLEMS  OUTPUT  TERMINAL  PORTS  GITLENS  DEBUG CONSOLE
○ hant@hant lb % go run main.go --backends=https://server1.com,http://server2.com,http://server3.com
2024/05/21 18:04:55 Added backend: https://server1.com
2024/05/21 18:04:55 Added backend: http://server2.com
2024/05/21 18:04:55 Added backend: http://server3.com
2024/05/21 18:04:55 Load Balancer started at :3000
2024/05/21 18:04:55 Load balancing to servers: [https://server1.com http://server2.com http://server3.com]
```

Gửi requests liên tiếp đến http://localhost:3000

Gửi request lần đầu: Điều phối đến http://server2.com

```
PROBLEMS  OUTPUT  TERMINAL  PORTS  GITLENS  DEBUG CONSOLE
○ hant@hant lb % go run main.go --backends=https://server1.com,http://server2.com,http://server3.com
2024/05/21 18:04:55 Added backend: https://server1.com
2024/05/21 18:04:55 Added backend: http://server2.com
2024/05/21 18:04:55 Added backend: http://server3.com
2024/05/21 18:04:55 Load Balancer started at :3000
2024/05/21 18:04:55 Load balancing to servers: [https://server1.com http://server2.com http://server3.com]
2024/05/21 18:05:36 Forwarding request to: http://server2.com
○ hant@hant lb % curl http://localhost:3000
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
○ hant@hant lb %
```

Gửi request lần hai: Điều phối đến http://server3.com

```
PROBLEMS  OUTPUT  TERMINAL  PORTS  GITLENS  DEBUG CONSOLE
○ hant@hant lb % go run main.go --backends=https://server1.com,http://server2.com,http://server3.com
2024/05/21 18:04:55 Added backend: https://server1.com
2024/05/21 18:04:55 Added backend: http://server2.com
2024/05/21 18:04:55 Added backend: http://server3.com
2024/05/21 18:04:55 Load Balancer started at :3000
2024/05/21 18:04:55 Load balancing to servers: [https://server1.com http://server2.com http://server3.com]
2024/05/21 18:05:36 Forwarding request to: http://server2.com
2024/05/21 18:06:16 Forwarding request to: http://server3.com
○ hant@hant lb % curl http://localhost:3000
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
○ hant@hant lb % curl http://localhost:3000
```

Gửi request lần ba: Điều phối đến http://server1.com

```
PROBLEMS  OUTPUT  TERMINAL  PORTS  GITLENS  DEBUG CONSOLE
○ hant@hant lb % go run main.go --backends=https://server1.com,http://server2.com,http://server3.com
2024/05/21 18:04:55 Added backend: https://server1.com
2024/05/21 18:04:55 Added backend: http://server2.com
2024/05/21 18:04:55 Added backend: http://server3.com
2024/05/21 18:04:55 Load Balancer started at :3000
2024/05/21 18:04:55 Load balancing to servers: [https://server1.com http://server2.com http://server3.com]
2024/05/21 18:05:36 Forwarding request to: http://server2.com
2024/05/21 18:06:16 Forwarding request to: http://server3.com
2024/05/21 18:06:21 http: proxy error: dial tcp: lookup server3.com: no such host
2024/05/21 18:06:35 Forwarding request to: https://server1.com
○ hant@hant lb % curl http://localhost:3000
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
○ hant@hant lb % curl http://localhost:3000
○ hant@hant lb % curl http://localhost:3000
```

Gửi request lần bốn: Điều phối đến http://server2.com

```
PROBLEMS OUTPUT TERMINAL PORTS GITLENS DEBUG CONSOLE
○ hant@hant lb % go run main.go --backends=https://server1.com,http://server2.com,http://server3.com
2024/05/21 18:04:55 Added backend: https://server1.com
2024/05/21 18:04:55 Added backend: http://server2.com
2024/05/21 18:04:55 Added backend: http://server3.com
2024/05/21 18:04:55 Load Balancer started at :3000
2024/05/21 18:04:55 Load balancing to servers: [https://server1.com http://server2.com http://server3.com]
2024/05/21 18:05:36 Forwarding request to: http://server2.com
2024/05/21 18:06:16 Forwarding request to: http://server3.com
2024/05/21 18:06:21 http: proxy error: dial tcp: lookup server3.com: no such host
2024/05/21 18:06:35 Forwarding request to: https://server1.com
2024/05/21 18:06:55 Forwarding request to: http://server2.com
○ hant@hant lb % curl http://localhost:3000
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
○ hant@hant lb % curl http://localhost:3000
<html><head><title>localhost</title></head><body><h1>localhost</h1><p>Coming soon.</p></body></html>
○ hant@hant lb % curl http://localhost:3000
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
○ hant@hant lb %
```


Ta thấy các request được điều phối đến các servers backend theo đúng thuật toán Round Robin. Tuy nhiên nó rất đơn giản và còn cần rất nhiều thứ cần cải thiện như là:

- Nếu một server đã chết thì ta sẽ không gửi request tới nó.
- Thực hiện kiểm tra health check cho server và đánh dấu server là unhealthy để load balancers không gửi request tới đó.

Dockerfile

```
1 FROM golang:1.16.3-alpine3.13 as builder
2 WORKDIR /app
3 COPY main.go go.mod ./
4 RUN CGO_ENABLED=0 GOOS=linux go build -o lb .
5
6 FROM alpine:3.13
7 RUN apk add --no-cache ca-certificates
8 WORKDIR /root
9 COPY --from=builder /app/lb ./
10 ENTRYPOINT [ "/root/lb" ]
```

docker-compose.yml



```
1  version: '1.0'
2  services:
3    frontend:
4      build: .
5      container_name: load-balancer-simple
6      ports:
7        - "3000:3000"
8      command: --backends "http://web1:80,http://web2:80,http://web3:80"
9    web1:
10     image: strm/helloworld-http
11    web2:
12     image: strm/helloworld-http
13    web3:
14     image: strm/helloworld-http
```

Link mã nguồn: <https://github.com/hantbk/lb/tree/simple>