分散式技術解析與論述: Envoy Proxy

林君華，蔡桂毓，包徠成，& 莊又齊 (2021)

國立政治大學資訊科學系

Author Note

分散式系統 (Distributed Systems) 期中報告，授課老師: 廖峻峰

僅作為學術用途

目錄

問題目的與背景

Envoy Proxy 是一個開源、佔用資源少的邊緣服務代理。它最初是由 Lyft 所編寫和部署，現在已擁有許多開發者活躍於此，以及是 Cloud Native Computing Foundation（CNCF）官方開發項目。

圖 1. Envoy Proxy

核心概念

Envoy Proxy 的誕生是為了在網路及其應用出現問題時，網路對於應用應該要是透明易讀的，使得用戶更加容易地確定問題的根源所在。

儘管 Envoy Proxy 擁有不俗的時間表現（根據其使用增量時間方法來優化其中部分快速路徑），但是其程式碼卻以易於測試且易於理解的方式編寫，而不是為了獲得最佳性能表現。這樣的編寫將使時間更有效地被利用，且 Envoy Proxy 通常與某些特定的電腦語言一起部署，其運行時間比其自身慢幾倍，並且具有更高的內存佔用量，致使其時間表現不是最重要的部分。

為何使用 Envoy Proxy?

Envoy Proxy 擁有非常好的擴充性，因此我們可以輕鬆地將其發展至 Istio 等全新的應用實例。即使 Envoy Proxy 版本時常更新，它也是一個非常穩定（LTE）的軟體，並在 CI 的單元和集合實驗中得到 98％以上的程式碼穩固保證。最重要的是，Envoy Proxy 得到了社會及民間大小企業（例如 Google，Pivotal，Red Hat 和 IBM）的大力支持，而這也是幾個常見服務代理項目背後的能成功的因素，使不同產業中的上下關係具有真正的合作精神。

同樣地，大家都知道 Envoy Proxy 是試圖在網路及其應用出現問題時，網路對於應用應該要是透明易讀的，並且它還提供了一種更好的方法來收集應用回傳的資訊，從而使追蹤及偵錯變得更加容易，而造成部署 Envoy Proxy 的最常見方法是將其和其他應用於發佈時部署於應用之中。

基於對 NGINX、HAProxy 等硬體負載平衡和雲端負載平衡解決方案的了解，Envoy Proxy 如先前所述，與應用同時運作，並通過與平台獨立的方式提供通用功能來抽象化網路。而當所有服務流量都通過 Envoy Proxy 所流動時，通過其所具備的可觀察性，調整整體性能以及在特定單位增減其功能，即可輕鬆解決其中可視覺化之問題區域。

範例: 使用 Envoy Proxy 來進行 gRPC Service 負載平衡

Envoy Proxy 主要是來擷取和處理各項服務之間的輸入及輸出流量其功能包括：動態查找服務、負載平衡、外部 TLS 支援、HTTP/2、gRPC 代理及短路器。

儘管 gRPC Service 具有負載均衡功能，但該功能主要是由 kube-proxy 所提供。由於 gRPC 的特性，使得它無法很好地處理負載平衡，更簡單的說就是 kube-proxy 無法理解 HTTP，而 gRPC 則是基於 HTTP/2 的 RPC 協議。它的主要功能之一即是不需要每次發出請求時就建立 TCP 連線，就像 HTTP 一樣，是重用已建立的連線。 因此，這導致 kube-proxy 僅在建立連線後才能成功執行負載平衡，並且每個後續的 RPC 請求則都將使用此連線。在這種情況下，每個後續的 RPC 請求都將發送到同一位置，而這不是我們所期望達到的負載平衡。我們希望每個 RPC 請求都將根據我們所想要的方式轉發到正確的目的地，而這也就是為什麼我們更喜歡使用 Istio 中的 Envoy Proxy 來做到這一點。

架構解析

Envoy Proxy 可以做為 Sidecar Proxy 隨著應用程序（Service）部署在一起，形成一個微服務（Microservice），通過修改 IP Table 實現對服務的進出口流量的攔截，並進一步實現對進出口流量的管理。通常一個微服務會部署多份副本，這些副本加在一起，就形成了 Service Cluster。Envoy Sidecar Proxy 在此處作為獨立的進程，代理 Service 的進口流量以及出口流量，實現 Load Balancing、Service Discovery、Health Check 等功能。同時，Envoy Proxy 也可以做為 Front Proxy 充當網絡的反向代理。

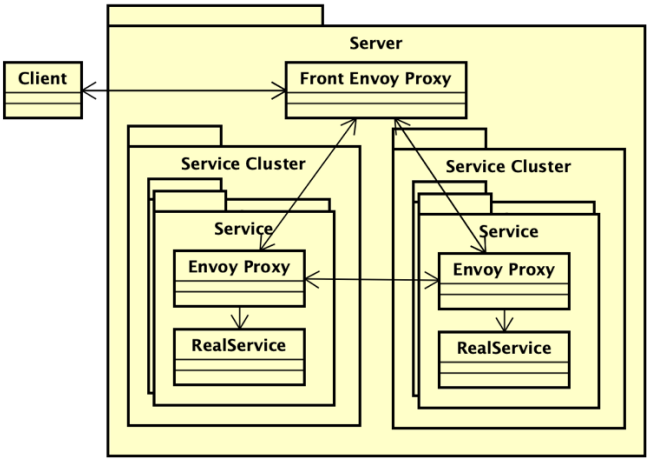

圖 2. Envoy Proxy 整體架構

當 Client 向 Server 送出要求時，此時流量先經過 Front Envoy Proxy，此處除了做中心認證、健康檢查等，Service Discovery 動態獲取每個 Service 的 IP，即可做指定的 Load Balancing (e.g. Round Robin)將流量轉發給後端的 Service Cluster 中的 Service，再由 Sidecar Proxy 處理為服務間的通信。

單獨解析 Envoy Proxy 基本架構:

- Downstream：下游主機連接到 Envoy，發送請求並接收響應，即發送請求的 Host。
- Upstream：上游主機接收來自 Envoy 的連接和請求，並返迴響應，即接受請求的 Host。
- Listener：Downstream 可以連接這些監聽器。 Envoy 暴露一個或者多個監聽器給 Downstream，每個 Listener 中可配置一條 Filter。Listener 可以監聽 Downstream，也可以接收來自其他 Listener 的資料。

- Cluster：集群是指 Envoy 連接的一組邏輯相同的 Upstream。Envoy 通過 Service Discovery 來發現 Cluster 的成員。可以選擇通過 Health Check 來確定集群成員的健康狀態。Envoy 通過 Load Balancing 決定將請求路由到集群的哪個成員。
- xDS：Envoy 動態發現資源所對應的 Discovery Service 及其相應的 API。
  - Route Discovery Service：獲取並添加、修改、刪除 HTTP connection manager 的 route configuration。
  - Cluster Discovery Service：獲取並添加、修改、刪除 cluster manager 的成員。
  - Endpoint Discovery Service：獲取 Upstream cluster。
  - Listener Discovery Service：獲取並添加、修改、刪除 Listeners。
  - Health Discovery Service：支持 Management server 對其管理的 Envoy 進行 endpoint health check。
  - Aggregated Discovery Service：通過適當得排序 xDS 可以無中斷的更新 Envoy 的配置。
  - Secret Discovery Service：支持 Envoy 發現 cryptographic secrets (certificate plus private key，TLS session ticket keys)給 Listeners。

當 Listeners 監聽到 Downstream，會通過 Filter 確認相對應的流量，並分發到正確 Cluster 的 Upstream。全程 xDS 使 Envoy 可以實現配置的完全動態化，配置實時更新而無需重啟 Envoy 或者影響服務，其中包括 Listener、Router、Cluster 和 Host (Upstream)。

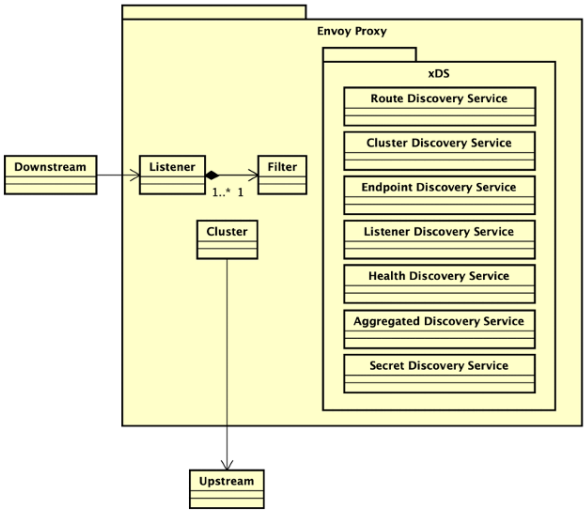

圖 3. Envoy Proxy 內部架構

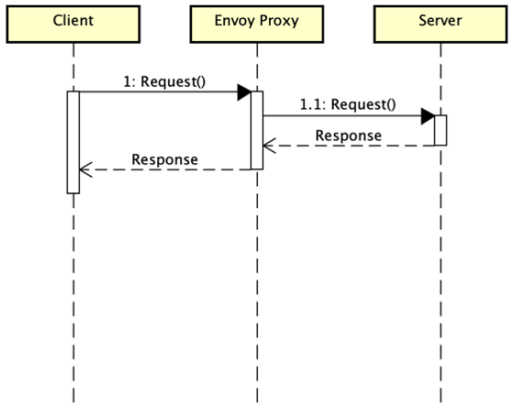

圖 4. Envoy Proxy 在 Sequence 中扮演分流、管理的角色，將 Client 請求分配給正確的 Service。

技術實作與心得

1. Install Envoy

使用 Envoy 官方提供的 official Docker image 進行安裝:

```
docker pull envoyproxy/envoy-dev:3b9610658f63d2cc01f8fd9e6ad65f423b3a716d
```

Envoy 跑起來，使用預設的 configuration:

```
docker run --rm -it \
      -p 9901:9901 \
      -p 10000:10000 \
      envoyproxy/envoy-dev:3b9610658f63d2cc01f8fd9e6ad65f423b3a716d
```

預設 configuration 讓 Envoy proxying 在 http://localhost:10000，可以透過以下指令確認:

```
curl -v localhost:10000
```

除此之外預設 configuration 也將 admin server 跑在 http://localhost:9901 ， 如下:

Command	Description
certs	print certs on machine
clusters	upstream cluster status
config_dump	dump current Envoy configs (experimental)
contention	dump current Envoy mutex contention stats (if enabled)
cpuprofiler	enable/disable the CPU profiler
drain_listeners	drain listeners
healthcheck/fail	cause the server to fail health checks
healthcheck/ok	cause the server to pass health checks
heapprofiler	enable/disable the heap profiler
help	print out list of admin commands
hot_restart_version	print the hot restart compatibility version
init_dump	dump current Envoy init manager information (experimental)
listeners	print listener info
logging	query/change logging levels
memory	print current allocation/heap usage
quitquitquit	exit the server
ready	print server state, return 200 if LIVE, otherwise return 503
reopen_logs	reopen access logs
reset_counters	reset all counters to zero
runtime	print runtime values
runtime_modify	modify runtime values

2. Install and run main application/ service

確認 Envoy 安裝成功、可以正常運行後，我們以 Spring Boot 上提供的一個範例(Hello-World Restful web service)作為我們要部署 Envoy 的 service。

(此範例須使用 JDK 1.8 後的版本及 Maven 3.2+，下載、執行方式請參考上方 Sprint Boot 連結。)

此 web service 開放 port 8080 做為呼叫地址，如下:

```
curl http://localhost:8080/greeting
```

執行結果:

```
{"id":1,"content":"Hello, World!"}%
```

3. Deploy Envoy as a sidecar on the main service

要將 Envoy 部署在 service 上，需要編寫 Envoy 的 configuration 檔，因此我們新增一份 simple.yaml，大致可以將結構分成三部分，(1) listeners, (2) clusters, (3) admin，其分別對應的功能描述於前面“架構解析”，並且在以下檔案以註解輔助說明。

simple.yaml 內容如下:

```
static_resources:
  listeners:
  - address:                # Listener will bind itself to
      socket_address:       # port 10000 at IPv4 address 0.0.0.0
        address: 0.0.0.0    # Traffic that comes through any other port,
        port_value: 10000   # Envoy won't have any knowledge for.


    filter_chains:          # Describes how the requests should be
                            # filtered and routed once it enters Envoy.
    - filters:
      - name: envoy.filters.network.http_connection_manager
        typed_config:
          "@type":
type.googleapis.com/envoy.extensions.filters.network.http_connection_manager.v3.HttpCo
nnectionManager
          codec_type: auto
          stat_prefix: ingress_http
          route_config:
            name: local_route
            virtual_hosts:
            - name: "http-route"
              domains:
              - "*"
              routes:         # Telling the chain to route traffic according
                              # to the prefix and the cluster that it matches.

                              # Try to match any request with "/greeting" request and
              - match:        # forwards that to greetingservice cluster.
                  prefix: "/greeting"
                route:
                  cluster: greetingservice
          http_filters:
            -
              name: envoy.filters.http.router

clusters:
- name: greetingservice
  connect_timeout: 0.25s
  type: strict_dns
  lb_policy: round_robin
  load_assignment:
    cluster_name: greetingservice
    endpoints:
    - lb_endpoints:
      - endpoint:               # An endpoint which redirects
          address:              # the requests to 0.0.0.0:8080
            socket_address:
              address: 0.0.0.0
              port_value: 8080

admin:                          # Runs the Envoy admin service on port 9901
  access_log_path: "/dev/null"
  address:
    socket_address:
      address: 0.0.0.0
      port_value: 9901
```

使用自己定義的 configuration 將 Envoy 跑起來:

```
docker run --rm -it \
      -v $(pwd)/simple.yaml:/simple.yaml \
      -p 9901:9901 \
      -p 10000:10000 \
      envoyproxy/envoy-dev:3b9610658f63d2cc01f8fd9e6ad65f423b3a716d \
          -c /simple.yaml
```

Envoy is proxying on port:10000, and redirect the request to port:8080:

```
curl http://localhost:10000/greeting
```

執行結果:

```
{"id":2,"content":"Hello, World!"}%
```

4. 連至 admin server (http://localhost:9901) 查看各種 states data

e.g. upstream cluster status:

```
service_envoyproxy_io::observability_name::service_envoyproxy_io
service_envoyproxy_io::default_priority::max_connections::1024
service_envoyproxy_io::default_priority::max_pending_requests::1024
service_envoyproxy_io::default_priority::max_requests::1024
service_envoyproxy_io::default_priority::max_retries::3
service_envoyproxy_io::high_priority::max_connections::1024
service_envoyproxy_io::high_priority::max_pending_requests::1024
service_envoyproxy_io::high_priority::max_requests::1024
service_envoyproxy_io::high_priority::max_retries::3
service_envoyproxy_io::added_via_api::false
service_envoyproxy_io::52.220.193.16:443::cx_active::0
service_envoyproxy_io::52.220.193.16:443::cx_connect_fail::0
service_envoyproxy_io::52.220.193.16:443::cx_total::1
service_envoyproxy_io::52.220.193.16:443::rq_active::0
service_envoyproxy_io::52.220.193.16:443::rq_error::0
service_envoyproxy_io::52.220.193.16:443::rq_success::2
service_envoyproxy_io::52.220.193.16:443::rq_timeout::0
service_envoyproxy_io::52.220.193.16:443::rq_total::2
service_envoyproxy_io::52.220.193.16:443::hostname::www.envoyproxy.io
service_envoyproxy_io::52.220.193.16:443::health_flags::healthy
service_envoyproxy_io::52.220.193.16:443::weight::1
service_envoyproxy_io::52.220.193.16:443::region::
service_envoyproxy_io::52.220.193.16:443::zone::
service_envoyproxy_io::52.220.193.16:443::sub_zone::
service_envoyproxy_io::52.220.193.16:443::canary::false
service_envoyproxy_io::52.220.193.16:443::priority::0
service_envoyproxy_io::52.220.193.16:443::success_rate::-1.0
service_envoyproxy_io::52.220.193.16:443::local_origin_success_rate::-1.0
```

5. 心得

透過 Docker 及官方提供的 docker image，部署 Envoy 變得十分容易快速，但欲將其部署在自己的 service 上時，必須編寫 configuration，然而其中資訊複雜且須先對 Envoy 架構及功能有十足掌握才能理解，因此花了許多時間在查找 configuration 相關資料，另外也因為對 Docker 掌握度不足，在 debug 時多了一些不確定因素讓 debug 的效率低落。

透過 admin server 及其 GUI interface 讓觀察相關狀態十分容易，但也因資料量巨大，需要再更了解其內容意義，才能找出對我們及後續專案重要、有意義的內容，也要再想想如何有意義地將這些資料應用在我們的專案上面。

評論

我們認為 Envoy Proxy 之所以符合分散式系統的概念，是因為 Envoy Proxy 符合「CAP 定理（CAP theorem）」，而一般的分散式系統皆符合 CAP 定理（CAP theorem）。CAP 定理，又被稱作布魯爾定理（Brewer's theorem），它指出對於一個分散式計算系統來說，不可能同時滿足以下三點：

1. 一致性（Consistency）: 所有資料庫客戶端同時更新，用同樣的查詢，一定取得相同的值（等同於所有節點訪問同一份最新的數據副本）。

2. 可用性（Availability）: 總是可以讀寫，不會因為資料失效，而遇到無法讀取之類的狀況（每次請求都能獲取到非錯的響應，但是不保證獲取的數據為最新數據）。

3. 分區容錯性（Partition tolerance）: 數個節點之間的網路或溝通出現問題，還是可以對外提供服務（以實際效果而言，分區相當於對通信的時限要求。系統如果不能在時限內達成數據一致性，就意味著發生了分區的情況，必須就當前操作在一致性和可用性之間做出選擇）。

在 Envoy Proxy 中存在著 Bootstrap、Listeners（監聽器配置）、Cluster（集群配置）、Route（路由配置）等四個主要的部分。當 Envoy Proxy 運作時，以 Listeners 監聽到 Downstream 的資料時為例，這時的 Listeners 在監聽到新的資料時所產生的狀態更新將會導致與其他節點的數據不一致，即喪失了 C 性質（Consistency）。而此時如果為了保證數據一致性，將 Listeners 設置為不可用，如此一來又會喪失了 A 性質（Availability）。除非能夠保證幾個節點間能夠互相隨時保持通信，才能夠既保證維持 C 性質又能保證維持 A 性質。然而這麼一來又會導致失去 P 性質（Partition tolerance）。這個「三點中僅能滿足兩點」的特性，即符合 CAP 定理（CAP theorem）。

結語

「Envoy 是專為大型現代 SOA (Service-Oriented Architecture) 架構設計的 L7 代理和通信總線。」

“Envoy is an L7 proxy and communication bus designed for large modern service-oriented architectures”

Envoy Proxy 最主要的理念，就是網絡對應用程序來說應該是透明的。當網絡和應用程序出現問題時，應該很容易確定問題的根源，這使得 Envoy 能在大型面向服務架構做到快速的部署及升級，不僅如此，Envoy 可以使用任何程式語言框架實現。除此之外，Envoy 支援 gRPC 的負載均衡，有別於 kube-proxy 無法理解 HTTP 而做不到，Envoy 被 Istio 的納入架構中，成為預設的 proxy，更好的管控 Kubernetes Cluster 的 Expose 的服務。

Envoy Proxy 理念的實現並不簡單，因此 Envoy Proxy 提供了非常多高級的功能，包括: L3/L4 filter 架構、HTTP L7 filter 架構、頂級 HTTP/2 支持、gRPC 支持、前端/邊緣代理支持等與 xDS。換句話說，Envoy 是模塊化和易於測試的方式來編寫的，而不是最大限度實現本身的最佳性能，Envoy 通常會和比它自身慢數倍、內存佔用高數倍的語言和運行時部署在一起，因此 Envoy 是為了更有效在與其部署的架構利用時間，而不是單就 Proxy 的角度做最優化。

實際操作 Envoy，讓我們對於分散式系統有了更深刻的認識。基於分散式系統的概念分析 Envoy，也讓我們看到了在分散式系統的趨勢中，遇到的各種問題以及其解決的方案，Envoy 為其中之一，我們認為 Envoy 的確成功了解決 Proxy 更高級的實現，也支援了基於 gRPC 的通信。