Scalabilité

L'infrastructure systèmes et réseaux est là pour supporter des "objets" de plus haut niveau: des applications ou des services (web ou réseaux). Il est essentiel de penser son infrastructure et son architecture applicative en termes de scalabilité.

Ma définition de la scalabilité : **"Aptitude d'un système informatique à accroitre linéairement sa capacité en fonction de la charge et en gardant les mêmes performances"**.

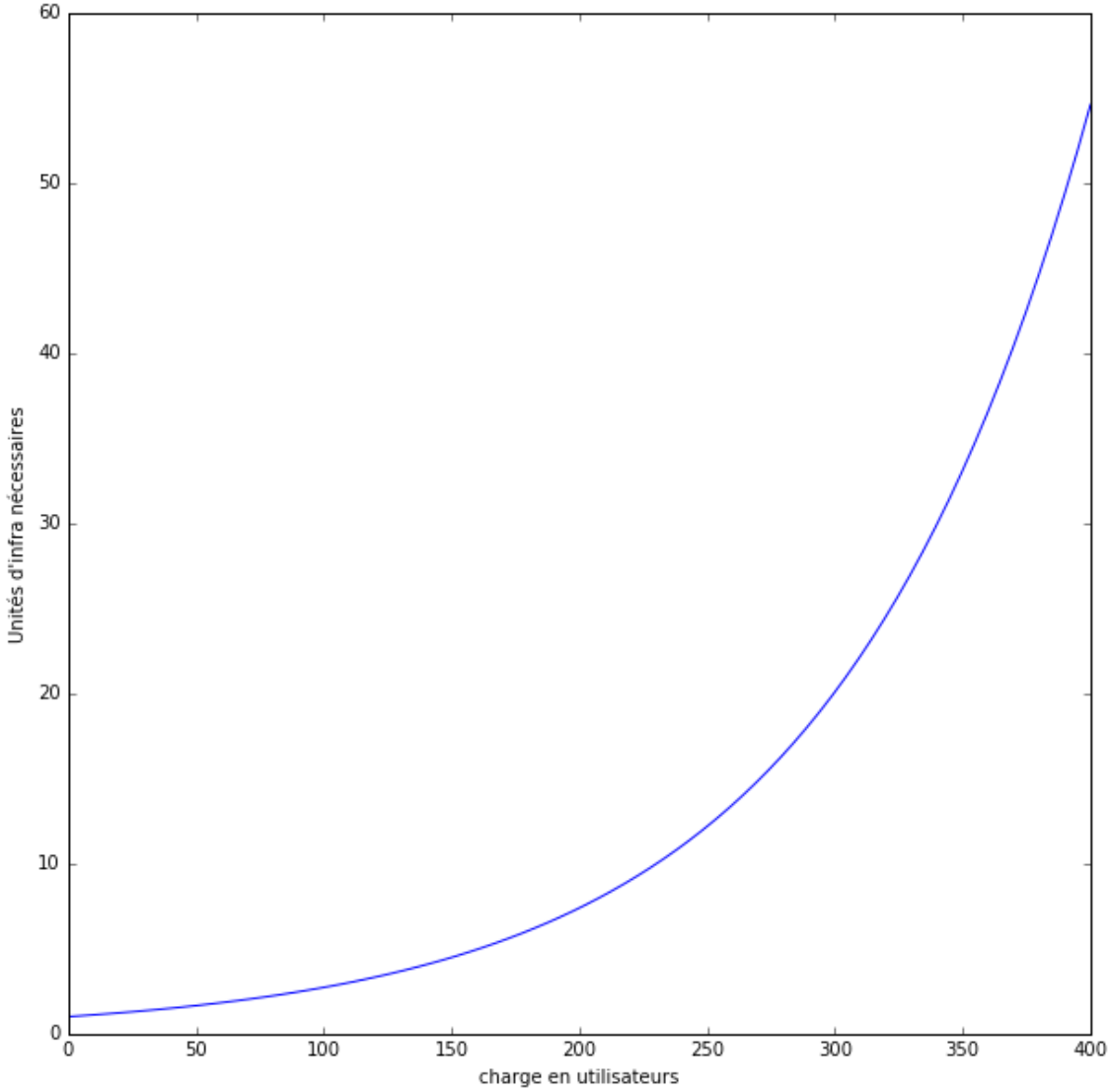
Autrement dit tout ajout de 100 utilisateurs simultanés va nécessiter x unités d'infra supplémentaires , en étant indépendant du nombre d'utilisateurs déjà connectés. Si on rajoute une charge de 200 utilisateurs alors on aura 2x unités d'infra supplémentaires.

C'est une définition idéale , tôt ou tard une analyse marginale va prévaloir et il faudra plus de ressources pour arriver à intégrer le même nombre d'utilisateurs. Néanmoins ca reste un leitmotiv: la variation du nombre d'utilisateurs au cours d'une même journée , la charge accrue pendant des périodes de soldes ,les pannes sont déjà en soi de bonnes raisons pour penser "scalable" et "agile" dès le départ.

> En fait d'après ma propre expérience , la scalabilité est limitée par la scalabilité de chaque briques (exemple CPU pour la compression HTTP, IO sur les baies de stockages) le comportement observé est plutôt celui là:

In [1]:

```
%matplotlib inline
%run expon.py
```



La scalabilité dépends de l'infrastructure mais aussi du logiciel. Les langages de scripts supportent la charge jusqu'à un certain seuil mais parfois il faut passer à autre chose (Twitter a démarré avec Ruby et est passé à JAVA, Facebook a démarré avec PHP et a changé pour passer sur du code compilable PHP->>C++-->EXE. Les bases de données, les stockages centralisés sont particulièrement délicats à scaler.

La mise en place de caches applicatifs, de caches de génération de pages, de caches d'éléments statiques et l'optimisation des sessions en termes de mémoire (JVM) sont des éléments qui participent à rendre une application performante et peuvent aider à augmenter la performance et gérer une charge supplémentaire. Les choix architecturaux doivent être cohérents et stables et sont à traiter parallèlement avec la recherche de scalabilité.

Une infra ne doit pas être dimensionnée à la moyenne mais en tenant compte d'un écart type calculé sur l'activité. Ensuite c'est un choix qui n'est pas technique et qui doit être pris par l'entreprise: quels moyens pour quels niveaux de services ? Passer de 100Mb/s à 1Gb/s pour être scalable est aussi un choix financier.

En sens inverse il est contre-productif de mettre des ressources là ou il n'y a pas de besoins. On cherche à avoir une infra **élastique** . Certains paliers demandent des changements lourds et une remise en cause profonde de l'infra et du dev.

Sharding ou partitionnement Horizontal

On va spécialiser des briques: Par exemple on fragmentera les tables d'utilisateurs par ordre alphabétique et on dédie un serveur à une catégorie d'utilisateurs.

Partitionnement Vertical

On va multiplier le nombre de brique nécessaires pour répondre à la charge (x LDAP, x DNS, x Serveurs WEB, x Baie de stockage...).

Il ne suffit pas de mettre des serveurs en ligne , il faut aussi répartir le flux web issu de la demande des utilisateurs vers les serveurs. Ce dispatcheur est appelé LOAD BALANCER ou équilibreur de charges. En 2015 les constructeurs qualifient cette fonction d'*applications switching*. Cette fonction d'équilibrage va permettre de tirer partie du partitionnement vertical.

Les solutions d'équilibrage de charge: RR DNS

Le **Round Robin DNS**: On affecte plusieurs IP à un seul nom dans le DNS. Pour un même FQDN les réponses DNS A type seront alternées entre ces IP. On équilibre ainsi la charge entre tous les serveurs, mais en cas de perte d'une machine, une réponse sur X requêtes ne renverra plus rien.

On peut envisager de modifier le pool d'IP du DNS à la volée mais du fait du mécanisme de cache DNS il faut abaisser le TTL ce qui n'est pas efficient en termes réseau.

Un autre mécanisme complémentaire peut être de basculer l'IP du serveur défaillant sur un autre serveur. C'est une VIP flottante. **Keepalived** sait produire des VIP flottantes. En cas d'arrêt d'un serveur , un autre serveur du pool détecte cet arrêt via **VRRP** et reprend à son compte la VIP assurant ainsi la haute disponibilité.

Les solutions d'équilibrage de charge: Le Load Balancer niveau 3 ou 4.

Une solution comme LVS (Linux Virtual Server) permet de répartir la charge en TCP/UDP sur différents serveurs. Elles est très performante du fait de son implémentation en kernel space et supporte des charges élevées. Néammoins ce type de solution ne peut pas être aussi "riche" qu'un équilibreur de charge fonctionnant en niveau 7.

Les solutions d'équilibrage de charge: Le Load Balancer niveau 7.

Ce type de solution permet de manipuler des flux HTTP au niveau 7. Comme elle nécessite d'ouvrir plus profondément les paquets réseaux , elle sera moins performante.

Un **LB** peut être hardware ou logiciel. Côté hardware on peut citer Citrix Netscaler, Riverbed,F5, A10... Des Machines virtuelles sont parfois fournies comme LB.

> Ces boitiers programmables disposent d'ASICs leurs permettant de traiter la charge. Il va s'agir de faire de la compression HTTP (gzip) et du SSL, de rediriger le flux Web vers un serveur au plus prêt géographiquement du client, ou encore de manipuler les headers.

Ces boitiers ont des fonctionnalités avancés. On peut créer des instances virtuelles de LB dédiés à des environnements et indépendants les uns des autres. Ils restent néanmoins très cher et sont à réserver pour des environnements sous très forte charge. Ils disposent de port 10Gb/s et de cartes de compression/SSL. Ils travaillent sur les couches 3 à 7.

Ces boitiers sont capables de réutiliser les connexions TCP ouvertes vers les serveurs, de détecter qu'un service est arrêté via des scripts de check, de rerouter le flux vers un serveur disponible.

Problématiques liées à l'équilibrage de charge.

HTTP est un protocole sans état, il faut donc un lien entre chaque requête HTTP pour que le serveur et le client aient de la mémoire . C'est le rôle du cookie de session présent dans toutes les applications web. En cas d'équilibrage de charge un serveur voyant arriver une requête avec un cookie de session généré par son jumeau ne sera pas quoi faire car il ne le reconnaitra pas.

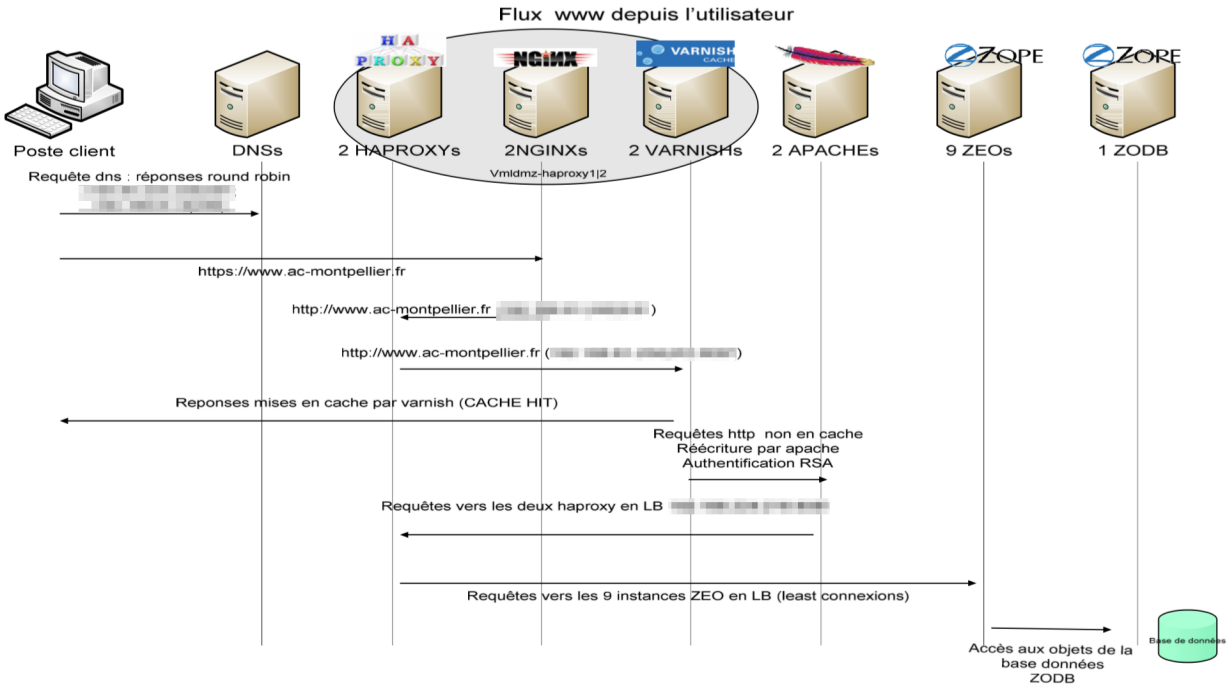
Solutions à cette problématique de l'équilibrages de charge.

- **Affinity**: On utilise un élément sous le niveau applicatif (ex IP source) pour que le client se connecte au même serveur.
- **Persistence**: On utilise un élément au niveau applicatif pour que le client se connecte au même serveur.
- **sticky session**: C'est une session maintenu par persistence. La persistence est plus sure mais a un cout important (mémoire, ouverture et modification des paquets réseaux en couche sept), mais l'affinité par exemple basée sur l'IP source pose le problème du changement côté client dont on n'est pas maitre.

Solutions logicielles sous Linux.

Néanmoins il existe des solutions logicielles, intégrées ou non dans le Kernel Linux qui sont efficaces dans des cas normaux. On peut améliorer les performances en rajoutant des cartes dédiées (GPU) pour le SSL ou la compression HTTP. LVS qui est un module intégré au Kernel et qui travaille en couche 3 et 4 est aussi un LB très performant.

Un exemple d'architecture scalable

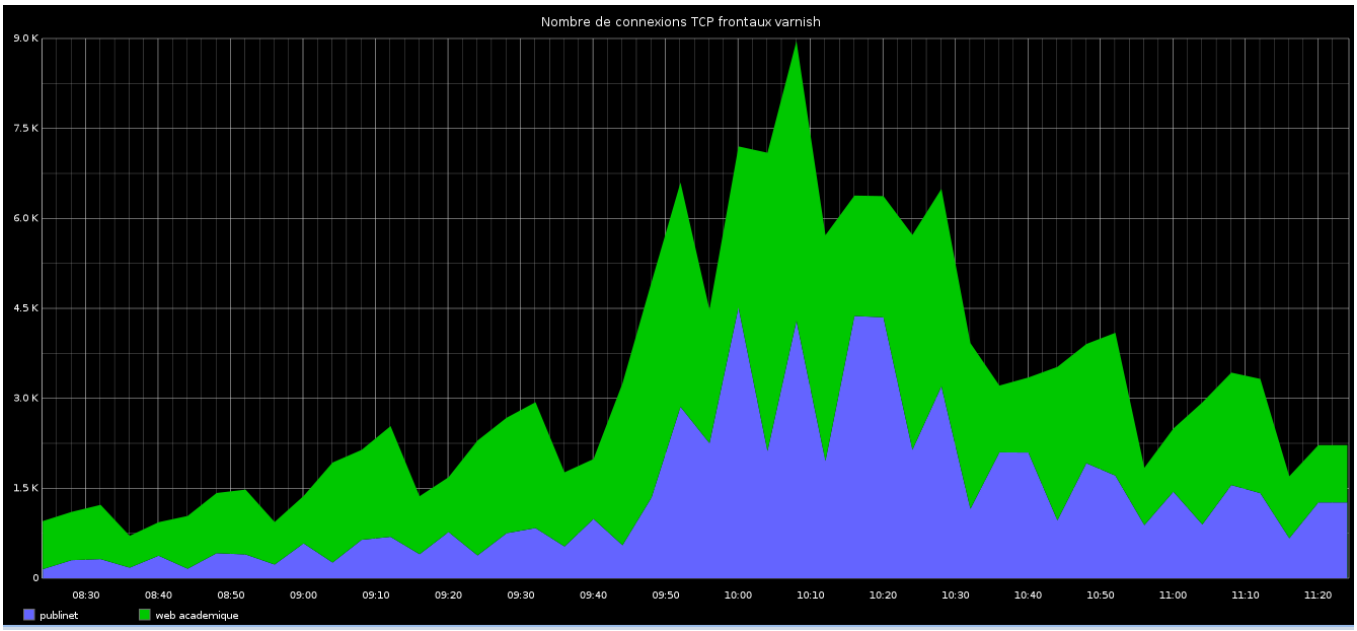


- Le DNS fait du RoudRobin répartissant la charge sur les deux serveurs frontaux HAPROXY.
- NGINX sert de reverse proxy et se charge de la connexion SSL. (Fonction qui peut être réalisée par haproxy depuis la version 1.5).
- Varnish est un accélérateur HTTP qui va mettre en cache spécifiquement certaines pages pour les délivrer plus rapidement (CSS, images, Texte....), pré-fetcher des requêtes avant que le cache n'expire, assurer la compression gzip...
- HA proxy est l'équilibreur de charge logiciel qui travaille en User Land. Il va diriger les requêtes HTTP vers les serveurs Web.
- Apache sert à authentifier le flux.
- CPS est un Content Management System bâtit sur Zope.

Ces briques sont toutes capables de garder la mémoire de l'utilisateur et de **"sticker"** l'utilisateur sur le bon serveur. On peut "sticker" en fonction de l'adresse source, d'un élément du header HTTP existant (JSESSIONID par ex), ou en rajoutant un cookie maison (ex SRV1ID)

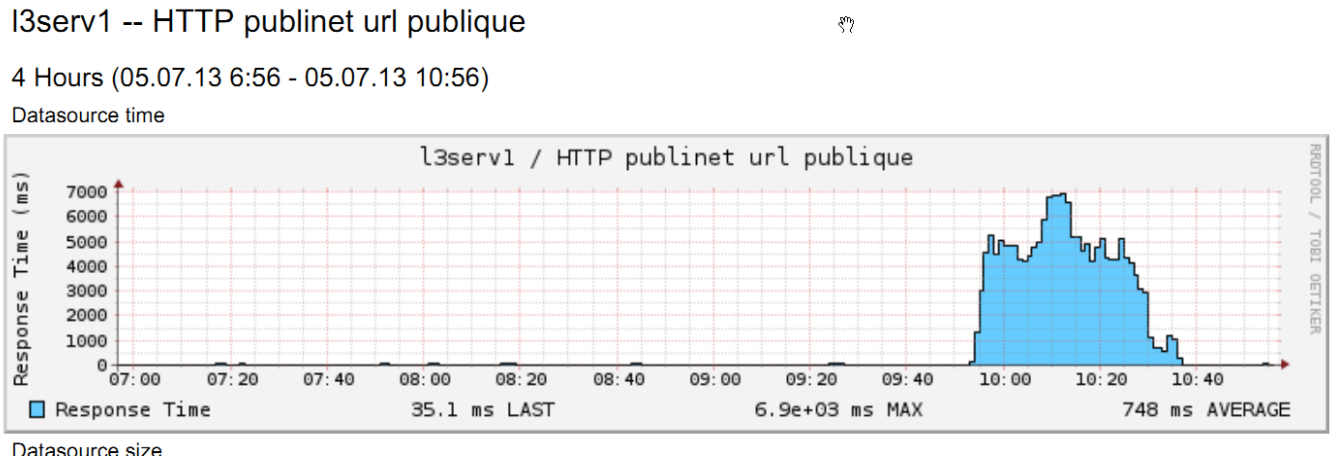
Cette architecture a supporté 120 Mb/s de flux lors de la consultation des résultats du Bac. A chaque niveau on peut ajouter des briques supplémentaires pour éponger la charge.

Charge en nombre de connexions

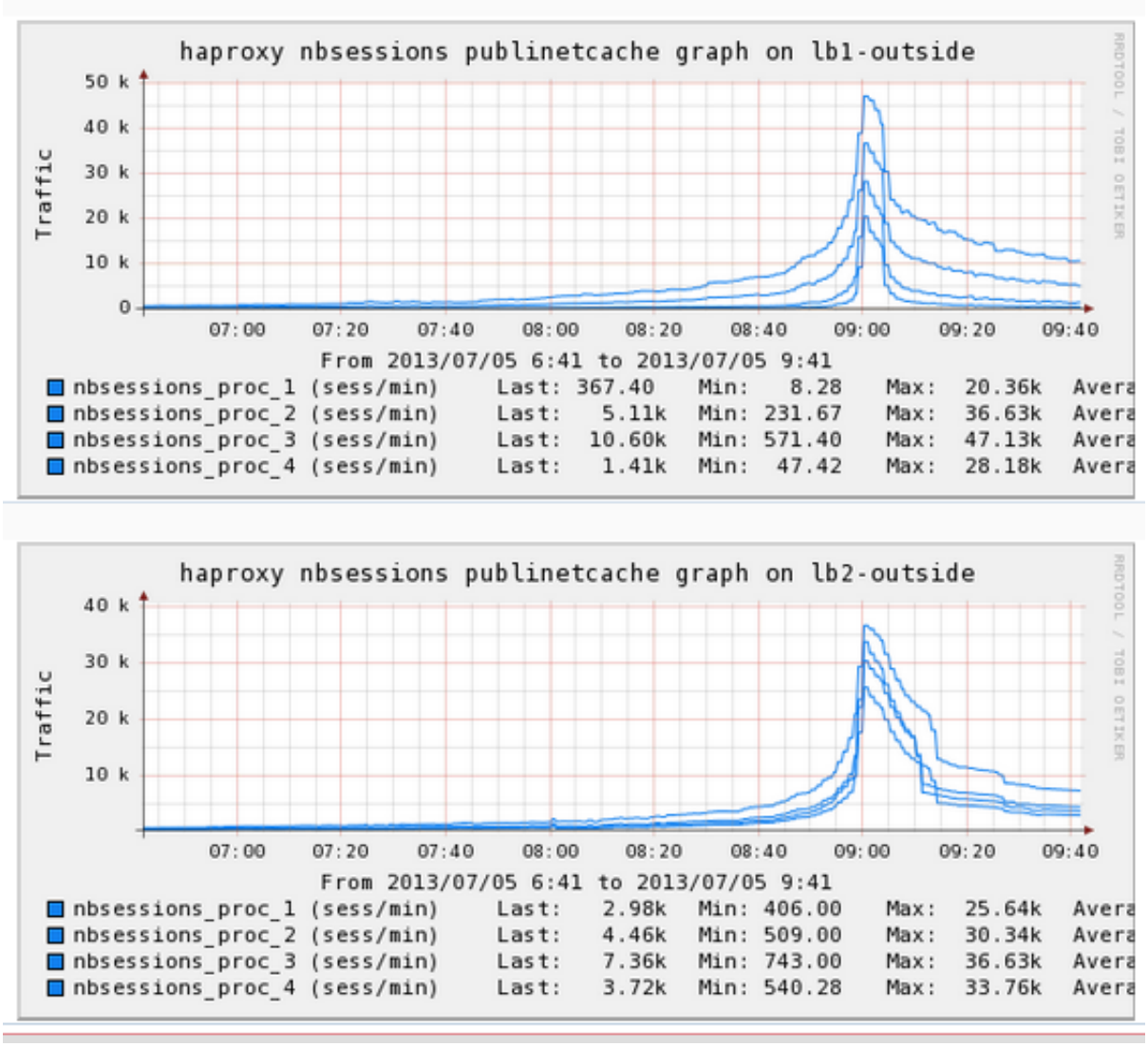


On le paye tout de même en Response Time en ms

L'architecture tient la charge au prix de temps de réponse long durant 40 mn. Il faut toujours se poser la question de l'investissement pour des pointes de traffic aussi courte.



Oui c'est scalable en utilisant des varnish *4



HAPROXY est un équilibreur de charge opensource. Créé par un Francais (Willy Tarreau) , Assez simple, très économe en ressources il est capable de load balancer du flux HTTP ou TCP (couche 3 à 7) mais pas UDP.

Haproxy

Selon la documentation officielle « HAProxy est un relais TCP/HTTP (il fonctionne donc aux niveaux 4 et 7) offrant des facilités d'intégration en environnement hautement disponible. Il est capable :

- D'effectuer un aiguillage statique défini par des cookies ;
- D'effectuer une répartition de charge avec création de cookies pour assurer la persistance de session ;
- De fournir une visibilité externe de son état de santé ;
- De s'arrêter en douceur sans perte brutale de service ;
- De modifier/ajouter/supprimer des en-têtes dans la requête et la réponse ;
- D'interdire des requêtes qui vérifient certaines conditions ;
- D'utiliser des serveurs de secours lorsque les serveurs principaux sont hors d'usage ;
- De maintenir des clients sur le bon serveur d'application en fonction de cookies applicatifs ;
- De fournir des rapports d'état en HTML à des utilisateurs authentifiés.

Linux est un fantastique OS, néanmoins y mettre un LB n'est pas une opération neutre sans effet et il convient de modifier quelques valeurs afin de l'optimiser. En cas de flux réseaux importants il convient de surveiller le nombre d'interruptions qui devient très élevé et qui va augmenter le Load Average du serveur en provoquant des attentes du CPU. Les options suivantes de /etc/sysctl.conf permettent d'optimiser les paramètres réseaux.

```
net.ipv4.ip_nonlocal_bind = 1 # permet au programme de se binder sur
ip non configurée
                              # indispensable afin de permettre une
haute dispo en actif actif

net.ipv4.ip_forward = 1       # activation du routage
fs.file-max = 131070          # Nombre de fichiers ouverts

# paramètres tcp recommandés sur le web
# increase TCP max buffer size setable using setsockopt()
# 16 MB with a few parallel streams is recommended for most 10G path
s
# 32 MB might be needed for some very long end-to-end 10G or 40G pat
hs
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216

# increase Linux autotuning TCP buffer limits
# min, default, and max number of bytes to use
# (only change the 3rd value, and make it 16 MB or more)
net.ipv4.tcp_rmem = 4096 87380 16777216
net.ipv4.tcp_wmem = 4096 65536 16777216
# protection antidos dos
net.core.netdev_max_backlog = 5000
net.ipv4.ip_local_port_range = 1000 65535 # On donne plus de connexi
ons possibles par IP
# Il faut de la performance et éviter le SWAP en priviligiant l'util
isation de la mémoire vive
vm.swappiness = 0
```

Netfilter est aussi un goulot d'étranglement et il convient d'augmenter la table de suivi des connexions via /etc/modprobe.confLe nombre d'interruptions du Kernel du aux grands nombre de paquets réseaux est aussi une caractéristique des serveurs servant d'équilibreurs de charges.

```
options ip_conntrack hashsize=24576
```

et dans /etc/security/limits.conf il faut aussi modifier le nombre de fichiers ouverts par process dans /etc/security/limits.conf :

```
www-data hard nofile 65536 ( nginx) haproxy hard nofile 65536 (hap
roxy)
```

Chiffrage des flux via SSL

l'utilisation d'un open SSL récent , sur un processeur récent permet d'avoir des performances tout à fait intéressante et en accroissement constant. voir bench [Intel] (https://software.intel.com/en-us/articles/accelerating-ssl-load-balancers-with-intel-xeon-v3-processors (https://software.intel.com/en-us/articles/accelerating-ssl-load-balancers-with-intel-xeon-v3-processors)) Sur une VM il est facile d'atteindre 1000 connexions secondes. Il est à noter qu'une fois la connexion SSL établie le cout d'une nouvelle connexion sera bien moindre. Le choix du CYPHER est aussi stratégique en termes de performances.

Architecture d'un LB.

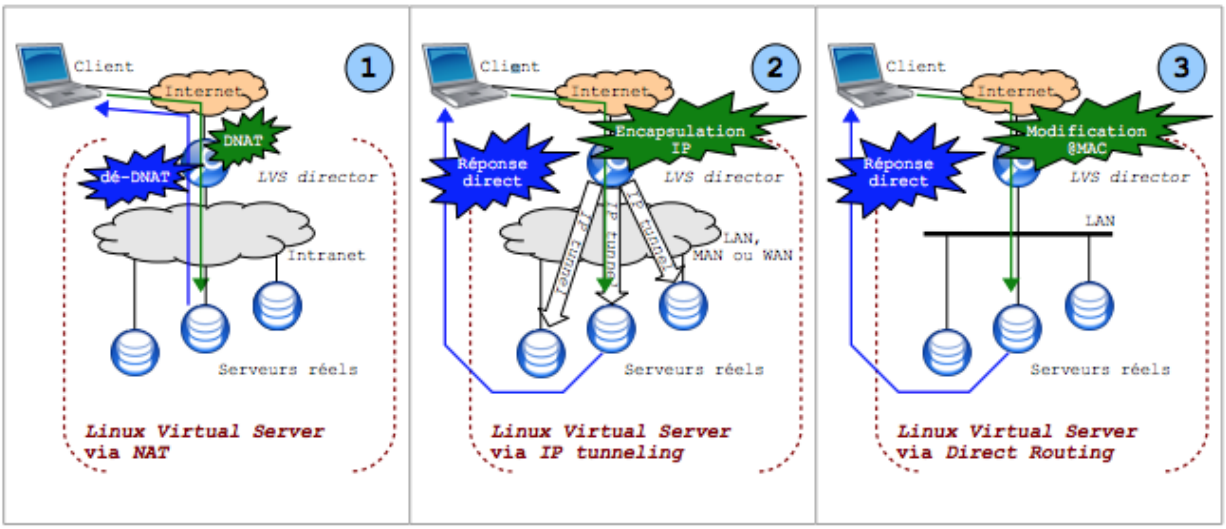
La mise en place d'un équilibreur de charge peut être structurante pour l'architecture réseau. Deux choix centraux s'offrent:

- Le fait d'avoir des équilibreurs dans des VLANs séparés des serveurs et d'utiliser le routage ou un tunnelpour communiquer entre le LB et les serveurs à équilibrer.
- Donner à l'équilibreur un carte réseau dans chaque VLANs des serveurs à équilibrer.

Une des problématiques centrales est de garder l'adresse IP source du client. Le rajout d'un header X-Forwaded-For ou mieux l'utilisation du proxy protocol (http://www.haproxy.org/download/1.5/doc/proxy-protocol.txt). Sans çà vous serez incapable de répondre à des demandes de stats ou à des réquisitions judiciaires.

Trois modes de configuration avec LVS

Schéma Source JRES 2013

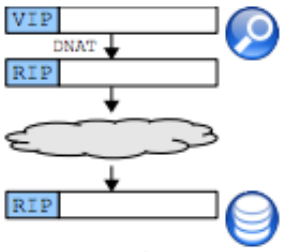


En fait il y a en a 4: Le mode FULL-NAT est le petit dernier. On fait dans ce cas du SOURCE NAT.

Le mode NAT est le plus intuitif et celui que j'ai le plus personnellement utilisé. Dans ce mode la translation a pour effet de remplacer l'adresse VIP du service par les adresses RIP des serveurs réels. Les serveurs peuvent etre adressés avec des adresses IP publiques ou privées mais doivent impérativement utiliser le répartiteur comme passerelle par défaut pour les paquets retour.

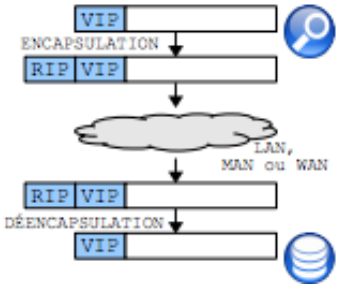
En imposant le LB comme le point de passage obligatoire on impacte très fortement l'architecture. Néanmoins en termes de configuration avec une passerelle par VLAN on reste proche d'une structure réseau "classique".

Schéma Source JRES 2013 A.SIMON.



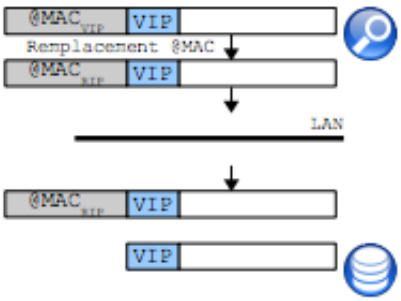
Le mode TUNNEL permet de supprimer l'adhérence du LB avec le VLAN des serveurs load-balancés. Néanmoins le rajout de tunnel n'est pas si simple et les perfomances seront moindre.

Schéma Source JRES 2013



La technique (3) de routage direct (Direct Routing) manipule les adresses MAC des trames Ethernet entre le répartiteur et les serveurs réels. Avec cette technique, le répartiteur va substituer l'adresse MAC de la trame d'origine par l'adresse MAC du RIP à utiliser, après quoi il retransmet la trame modifiée sur le lien physique. La trame modifiée est nativement acheminée par le réseau de niveau 2 au serveur visé. Cette modification d'adresse MAC est très simple a mettre en œuvre par le répartiteur et peu coûteuse, ce qui apporte a cette solution les performances les **plus élevées** par rapport aux deux autres modes présentés. La seule contrainte du routage direct est que le répartiteur et tous les serveurs réels doivent avoir un lien commun sur le meme segment de réseau de niveau 2 (répartiteur et serveurs avec une interface réseau sur le meme VLAN).

Source JRES A.SIMON



In [ ]:

```
%%bash
#ipvadm permet de configurer LVS:

# rajoute un service sur la VIP (Virtual Ip Address) 192.168.100.100

ipvsadm -A -t 192.168.100.100:80 -s rr

# Rajoute deux RIP (Real IP Address)
ipvsadm -a -t 192.168.100.100:80 -r 192.168.100.2:80 -g -w 1
ipvsadm -a -t 192.168.100.100:80 -r 192.168.100.3:80 -g -w 1

# Visualisation:
ipvsadm -Ln
```

In [ ]:

```
%%bash


ipvsadm -A -t 192.168.100.100:80 -s rr

# Rajoute deux RIP (Real IP Address)
ipvsadm -a -t 192.168.100.100:80 -r 192.168.100.2:80 -g -w 1
ipvsadm -a -t 192.168.100.100:80 -r 192.168.100.3:80 -g -w 1
```

Persistence des sessions:

C'est une des problématiques essentielle de l'équilibrage de charge et plus généralement de la scalabilité. Les protocoles sans état comme http nécessite un lien entre les différentes requêtes faites par le client. Si lors de la requête le client est redirigé vers un nouveau serveur, ce serveur ne reconnaitra pas le client et demandera une nouvelle authentification.

Une solution est d'assurer la persistence de sessions via un cluster de machines. Dans le monde Java on peut citer weblogic qui permet d'assurer une haute disponibilité applicative en recopiant les objets sur différents serveurs en utilisant le multicast et les sockets.

Une autre solution est de partager les sessions. Php est l'exemple type de mode de fonctionnement. A chaque utilisateur va correspondre des informations stockées par PHP. Ces informations sont liées normalement à un serveur ce qui n'est pas compatible avec du Load Balancing. Des outils comme REDIS peuvent stocker les sessions PHP et les rendre accessible à tous les serveurs PHP. Redis stocke des clefs/valeurs qui peuvent être des sessions PHP mais aussi des Datas , du cache de requêtes SQL...

Une autre solution est d'utiliser des cookies de SESSION qui vont être reconnus par le LOAD BALANCER. Un JSESSIONID générée par l'application va permettre à ce dernier de ce souvenir à quel serveur est rattaché quel JSESSIONID et de redirigé toujours la requête vers le même serveur. Si l'application ne génère pas de SESSIONID , le Load balancer est en capacité de de créer un cookie de session et de l'intégrer dans le header HTTP. Néanmoins cette manoeuvre qui nécessité d'ouvrir un paquet en Layer 7 est couteuse en temps et donc en performance.

On peut aussi stocker les sessions dans une base de données, sur des systèmes de fichiers partagés mais ces solutions sont peu efficaces vis à vis d'un stockage en mémoire et n'ont plus court sauf pour du cache de documents. On peut aussi sticker un client en fonction de son adresse IP. C'est une solution simple mais le NAT peut très bien fausser l'équilibrage de charge car on ne peut connaitre le nombre de client différents derrière l'adresse de NAT.

TLS et load balancing.

TLS permet de chiffrer les connexions entre le client et le serveur. TLS est bati sur TCP et la latence qui en résulte lors de la poignée de main TCP peut impacter sensiblement les performances. Il y a d'abord un TLS handshake qui va permettre au client et au serveur de s'accorder sur les protocoles de chiffrage. Une fois ce handshake terminé, l'échange de données peut commencer. A chaque nouvelle connexion d'un même client le handshake n'est pas nécessaire : TLS est capable de se garder en cache les informations ce qui améliore le nombre de requêtes SSL traitées. Bien sur la condition est que la requête arrive sur le même serveur...

Gestion de SSL avec le load balancer

Deux modes existent:

- SSL terminaison: La connexion est chiffrée du client au load balancer. Le traffic passe en clair du LB au serveur. Ce mode permet de voir le traffic et donc de manipuler les headers http.
- SSL passthru: La connexion est chiffrée du client au server, le LB traite le flux en tant que flux TCP sur le port 443 sans le décrypter.Ce mode ne permet pas au LB de manipuler les headers HTTP.

Compression http

Les pages de textes, les feuilles de style , le javascript peut être compresser dans le format gzip avant envoi vers le client. Le navigateur decompresse ce flux avant de l'afficher ce qui permet de gagner de la bande passante sur le réseau et d'avoir un effet booster visible par le end-user: les CPU sont rapides et décompressent le flux plutôt que d'attendre les paquets sur le réseau.

Mise en cache

La mise en cache des éléments statiques (images , texte, css..) via Varnish ou NGINX permet de scaler une application en déchargeant le serveur final qui met du temps à générer dynamiquement les pages web. Cette génération dynamique n'a pas d'intérêt si le contenu est statique (par exemple les listes des résultats du bac qui ne changent pas ou très peu). Le LB agit au niveau 7 afin de rediriger les requêtes HTTP vers le cache (accélérateur) http.

HAPROXY: un équilibreur de charge opensource de niveau 7.

Linux Virtual Server est un outil travaillant en "Kernel Land" et capable de traiter des flux importants en UDP ou TCP. Néanmoins la plupart des applications actuelles sont des application web travaillant au niveau 7. Elles nécessitent souvent des opérations plus sophistiquées qu'un "simple mapping de port" ou un dnat. Haproxy est l'outil libre qui s'est imposé comme l'équilibreur de charge logiciel de niveau 7. Asynchrone , des performances restent néamoins très importantes même en userland. De très haut niveau il est en capacité de manipuler de façon complexe les requêtes http, de partager de l'information pour faire du master-master. Il peut aussi travailler au nivau 4 en TCP mais ne supporte pas UDP.

Configuration simple de haproxy

Haproxy va publier des VIP (Virtual Ip Address) à destination des clients web au travers de la directive frontend

```
frontend f-http
    bind 10.0.0.1:80
    default_backend b_http
```

La VIP sera 10.0.0.1 et écoutera sur le port 80, les requêtes seront redirigées vers les serveurs référencés par le backend b_http.

```
backend b_http
    balance roundrobin
    option httpchk HEAD /index.html HTTP/1.0
    server s_http1 172.20.0.1:80 check
    server s_http2 172.20.0.2:80 check
```

Les deux serveurs traiteront à tour de rôle les requêtes web.

Affinité d'un client pour un serveur

Haproxy permet est capable de rediriger un client sur le même serveur, en se servant d'un cookie existant ou en insérant un nouveau cookie.

```
backend b_http
    cookie QUELSERVEUR insert indirect
    option httpchk HEAD /index.html HTTP/1.0
    server s_http1 172.20.0.1:80 cookie A check
    server s_http2 172.20.0.2:80 cookie B check
    server s_http3 172.20.0.3:80 cookie C check
```

Utilisation d'ACL pour rediriger vers un backend spécifique.

```
frontend f_http
    bind 10.0.0.1:80
    default_backend b_http
    acl acl_prof path_beg /prof
    use_backend b_prof if  acl_prof
    default_backend b_http

backend b_prof
        option httpchk HEAD / HTTP/1.0
        server web3 192.168.100.100:80 check
```

Configuration en coupure SSL (SSL terminaison)

Haproxy est capable de dialoguer en SSL avec le client

```
frontend www-https
   bind haproxy_www_public_IP:443 ssl crt /etc/ssl/private/example.com.pem
   reqadd X-Forwarded-Proto:\ https
   default_backend b_http
```