

## Répartition de charge sur une plate-forme Web

Propriétés	Description
<b>Type de publication</b>	Côté Labo
<b>Intitulé court</b>	<b>Répartition de charge sur une plate-forme Web</b>
<b>Intitulé long</b>	<b>Répartition de charge sur une plate-forme Web</b>
<b>Module</b>	BTS SIO2 – <b>SISR3 – Exploitation des services</b>
<b>Transversalité</b>	<b>SI7 :</b> <ul style="list-style-type: none"> <li>Stratégies et techniques associées à la continuité de service</li> <li>Stratégies et techniques de répartition et de réplication</li> </ul> <b>SISR4 :</b> <ul style="list-style-type: none"> <li>Contrôler et améliorer les performances d'un système</li> </ul>
<b>Présentation</b>	L'objectif de ce Côté Labo (mis en œuvre en module) est d'optimiser l'utilisation des deux serveurs Web en configurant une répartition de charge.
<b>Activités</b>	<b>D1.3 - Mise en production d'un service</b> <ul style="list-style-type: none"> <li>A1.3.2 Définition des éléments nécessaires à la continuité d'un service</li> </ul> <b>D2.1 - Exploitation des services</b> <ul style="list-style-type: none"> <li>A2.1.2 Évaluation et maintien de la qualité de service</li> </ul> <b>D3.2 - Installation d'une solution d'infrastructure</b> <b>D3.3 - Administration et supervision d'une infrastructure</b> <ul style="list-style-type: none"> <li>A3.3.1 Administration sur site ou à distance des éléments d'un réseau, de serveurs, de services et d'équipements terminaux</li> </ul>
<b>Savoir-faire principaux</b>	<b>En SISR3 :</b> <ul style="list-style-type: none"> <li>Caractériser les éléments nécessaires à la qualité, à la continuité et à la sécurité d'un service</li> <li>Installer et configurer les éléments nécessaires à la qualité et à la continuité du service</li> <li>Contrôler et améliorer les performances d'un service</li> <li>Valider et documenter la qualité, la continuité et la sécurité d'un service</li> </ul>
<b>Outils</b>	<b>SE :</b> Linux Ubuntu <b>Serveurs/services :</b> Apache2 installé et configuré à l'identique sur deux serveurs, HAProxy. <b>Clients :</b> navigateur sur STA Linux, Windows ou autre système (Lynx pour le navigateur en console)  <b>Site officiel de HAProxy:</b> <a href="http://haproxy.1wt.eu/">http://haproxy.1wt.eu/</a> <b>Documentation en ligne :</b> <a href="http://haproxy.1wt.eu/#docs">http://haproxy.1wt.eu/#docs</a>  <b>Sources :</b> <ul style="list-style-type: none"> <li><a href="http://1wt.eu/articles/2006_lb/">http://1wt.eu/articles/2006_lb/</a> : article sur la répartition de charges par le développeur d'HAProxy</li> <li><a href="http://architectures-web.smile.fr/Repartition-de-charge/Principe-de-repartition-de-charge">http://architectures-web.smile.fr/Repartition-de-charge/Principe-de-repartition-de-charge</a></li> <li>Article de Fabien Germain dans Linux Magazine Hors-série n°45</li> </ul>
<b>Mots-clés</b>	Disponibilité, Répartition de charge, HAProxy
<b>Durée</b>	2 heures
<b>Auteur(es)</b>	Apollonie Raffalli et David Duron avec la relecture attentive de Gaëlle Castel  Pour ce document, extraits choisis par Marie-pascale Delamare (pour la publication complète (avec réplication de données et cluster web voir la publication sur <a href="http://www.reseaucerta.org">http://www.reseaucerta.org</a> )).

## SISR3 : Cluster et répartition de charge

Tout serveur a une capacité de traitement limitée. Lors de périodes de pointe, cette capacité peut s'avérer insuffisante. Il est alors nécessaire d'ajouter un ou plusieurs serveurs afin de répartir le travail (la charge) entre eux.

La répartition de charge (load balancing) est « un ensemble de techniques permettant de distribuer une charge de travail entre différents ordinateurs d'un groupe. Ces techniques permettent à la fois de répondre à une charge trop importante d'un service en la répartissant sur plusieurs serveurs, et de réduire l'indisponibilité potentielle de ce service que pourrait provoquer la panne logicielle ou matérielle d'un unique serveur » (source [http://fr.wikipedia.org/wiki/R%C3%A9partition\\_de\\_charge](http://fr.wikipedia.org/wiki/R%C3%A9partition_de_charge)).

La répartition de charge est donc une des technologies qui participe à la haute disponibilité. Elle s'entend le plus souvent au niveau des serveurs HTTP (par exemple, sites à forte audience devant pouvoir gérer des centaines de milliers de requêtes par secondes), c'est-à-dire en frontal sur une plate-forme web comme nous allons le mettre en œuvre dans ce Côté Labo. Mais le même principe peut s'appliquer sur n'importe quel service aux utilisateurs ou service réseau.

### Principes de la répartition de charge

Les techniques de répartition de charge les plus utilisées sont :

- **Le DNS Round-Robin (DNS RR)** : lorsqu'un serveur DNS répond à un client, il fournit une liste d'adresses IP, dans un certain ordre, la première adresse étant celle que le client utilisera en priorité (les autres sont des adresses de secours) ; l'ordre sera évidemment différent pour un autre client (permutation circulaire en général). Le Round-Robin peut être mis en œuvre sur n'importe quel serveur DNS.
- **Le niveau TCP/IP ou niveau 4** : le client établit une connexion vers le « répartiteur » (matériel ou outil logiciel) qui redirige ensuite les paquets IP entre les serveurs selon l'algorithme choisi lors de la configuration (RR, aléatoire, en fonction de la capacité des serveurs, etc.).
- **Le niveau « applicatif » ou niveau 7 ou « répartition avec affinité de serveur »** : on analyse ici le contenu de chaque requête pour décider de la redirection. En pratique, deux choses sont recherchées et analysées :
  - les cookies, qui figurent dans l'entête HTTP ;
  - L'URI, c'est-à-dire l'URL et l'ensemble de ses paramètres.Ce niveau est parfois rendu nécessaire par certaines applications qui exigent que les requêtes d'un même utilisateur soient adressées à un même serveur.  
Cette technologie de répartition induit bien évidemment des délais supplémentaires car chaque requête HTTP doit être analysée.

### Le répartiteur de charge logiciel HAProxy

HAProxy est le logiciel libre de répartition de charge le plus utilisé. C'est une solution très complète au plan fonctionnel, extrêmement robuste et performante.

**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.

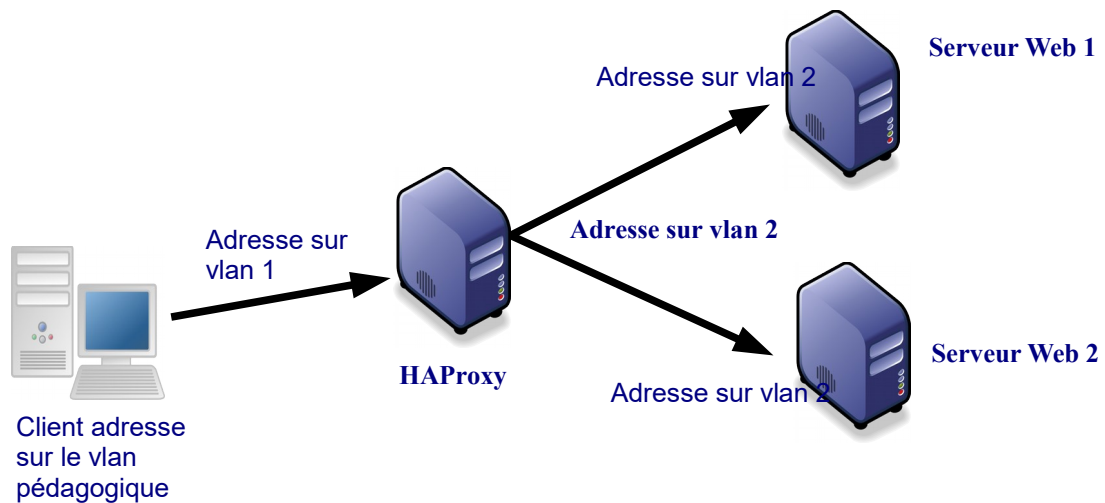
En outre, il requiert peu de ressources et son architecture événementielle mono-processus lui permet de gérer facilement plusieurs milliers de connexions simultanées sur plusieurs relais sans effondrer le système. »



HAProxy peut aider aussi à se prémunir contre les attaques DOS (voir par exemple ici : [http://publications.jbfavre.org/web/protegez\\_votre\\_serveur\\_web\\_avec\\_haproxy.fr](http://publications.jbfavre.org/web/protegez_votre_serveur_web_avec_haproxy.fr)) mais cela ne sera pas abordé dans ce Côté Labo.

## Architecture et contexte

Une configuration relativement simple va nous permettre de mettre en œuvre la répartition de charge sur deux serveurs Web préalablement installés.



La demande de connexion est adressée au serveur HAProxy (coté client) qui détermine, selon l'algorithme configuré, le serveur auquel il va affecter la connexion, parmi les serveurs disponibles (en l'espèce serveur web 1 (lamp1) et serveur web 2 (lamp2)).

Une fois la connexion TCP établie, l'équipement de répartition de charge devient pratiquement transparent : dans son rôle de base, il transfère les paquets IP du client vers le serveur sélectionné et vice versa jusqu'à fermeture de la connexion.

## Besoins :

L'équipe informatique de votre organisation souhaite que chacun de vos serveurs web soit utilisé de manière optimale, qu'il n'y en ait pas un surchargé tandis que l'autre est sous-utilisé. Les deux serveurs web sont identiques au titre près de la page d'accueil et situés sur le même vlan routé qu'OpenERP.

Il est nécessaire d'ajouter aussi un serveur sur lequel sera installé HAProxy (vous déploierez le modèle Mod\_Lubuntu\_Docker).

HAProxy nécessite 2 cartes réseaux :

- une considérée comme « publique » permettra l'accès depuis « l'extérieur » (coté client). Il faut ici utiliser un nouveau vlan pour la partie publique (soit en tout deux vlan routés pour ce TP) ;
- l'autre comme considérée comme « privée » assurera la communication avec les 2 serveurs Web.

## Vous trouverez :

- en **annexe 1**, un mode opératoire concernant l'installation et la configuration minimale de HAProxy ;
- en **annexe 2**, des indications sur une configuration plus avancée d'HAProxy ;
- en **annexe 3**, des exemples (avec commentaires) de statistiques obtenues.

## Déroulement de la séquence

### 1. Mise en place des deux serveurs web

Il vous faut deux machines virtuelles linux turnkey-lamp. Pour cela vous déploierez le modèle Mod\_SISR3Web.

- Pensez à changer le nom de vos machines (/etc/hostname, /etc/hosts).
- Configurez vos interfaces (/etc/network/interfaces) et vérifiez la connectivité entre vos deux machines.
- Modifiez une des deux pages web pour pouvoir différencier ces deux serveurs ;
- Testez l'accès à vos pages web respectives.

### 2. Mise en place de l'environnement de test, installation et première configuration d'HAProxy (aide en annexe 1)

On part ici du principe que l'on doit donc **intégrer le nouveau service HAProxy**.

- Rédigez une note argumentée listant les modifications à apporter à votre environnement afin de pouvoir tester la nouvelle solution.
- Mettez en place cette plate-forme de test.
- Installez HAProxy.
- Procédez à une première configuration d'HAProxy avec mise en place des statistiques.
- Démarrez HAProxy.
- Proposez et réalisez des tests (tests de non régression compris) permettant de vérifier que la solution est opérationnelle.

### 3. Configuration avancée d'HAProxy (aide en annexe 2)

- Procédez à une nouvelle configuration d'HAProxy en faisant l'hypothèse que le serveur initialement de secours (lamp2) est trois fois moins puissant que le serveur maître (lamp1).
- Proposez et réalisez des tests permettant de vérifier que la solution est opérationnelle.
- Revenez à une répartition de charge égalitaire.
- Vérifiez si le changement de serveur ne pose pas de problème lorsque l'utilisateur s'authentifie. Auquel cas, configurez HAProxy pour pouvoir utiliser l'application c'est à dire laisser la session se dérouler sur le même serveur.

### 3. Pour aller plus loin (aide en annexe 3)

Votre organisation dispose d'OpenERP dont tous les services sont sur un même serveur physique. Ce dernier doit être accessible via HAProxy sans toute fois être intégrée dans le mécanisme de répartition de charges.

- Intégrez à HAProxy le serveur OpenERP.

## Annexes

### Annexe 1 : installation et première configuration d'HAProxy

#### Installation et démarrage d'HAProxy

```
apt-get update & apt-get install haproxy
```

Le démon haproxy est configuré pour se lancer au démarrage de la machine. Son script de lancement se trouve dans `/etc/init.d`.

**Mais par défaut, HAProxy n'est pas activé. Pour le faire, il faut mettre la valeur « 1 » à la variable `ENABLED` dans le fichier `/etc/default/haproxy`.**



Le démon ne démarre pas, car le fichier de configuration par défaut ne contient pas toutes les directives nécessaires :

```
root@haproxyAR:~# /etc/init.d/haproxy restart
[...] Restarting haproxy: haproxy[ALERT] 236/215943 (8688) : config : no <listen>
line. Nothing to do !
[ALERT] 236/215943 (8688) : Fatal errors found in configuration.
failed!
```

Il est nécessaire de procéder à une configuration minimale.

#### Configuration d'HAProxy

Le fichier de configuration `/etc/haproxy/haproxy.cfg` se décompose en plusieurs sections repérées par des mots clés dont :

- **global** : paramètres agissant sur le processus ou sur l'ensemble des proxies ;
- **defaults** : paramétrages par défaut qui s'appliquent à tous les *frontends* et *backends*. Ces paramètres peuvent être redéfinis dans chacune des autres sections.

Ces deux sections sont déjà alimentées avec des directives et des valeurs acceptables (pour une plate-forme de test) :

Directives et valeurs	Quelques explications
<b>global</b> log /dev/log local0 log /dev/log local1 notice chroot /var/lib/haproxy user haproxy group haproxy daemon	Le paramètre <b>chroot</b> change la racine du processus une fois le programme lancé, de sorte que ni le processus, ni l'un de ses descendants ne puissent remonter de nouveau à la racine.

**Les logs d'HAProxy** sont écrits dans `/var/log/haproxy.log`. Pour une solution en production, il sera nécessaire de gérer les logs plus finement (idéalement intégrés à une solution de journalisation centralisée).

Directives et valeurs	Quelques explications
<b>defaults</b> log global mode http * option httplog option dontlognull timeout connect 5000	<b>mode http</b> : le service relaye les connexions TCP vers un ou plusieurs serveurs, une fois qu'il dispose d'assez d'informations pour en prendre la décision. Les entêtes HTTP sont analysés pour y trouver un éventuel cookie, et certains d'entre-eux peuvent être modifiés par le biais d'expressions régulières.  <b>Temps d'expiration des connexions</b> (valeur en

timeout client 50000 timeout server 50000  errorfile 400 /etc/haproxy/errors/400.http errorfile 403 /etc/haproxy/errors/403.http errorfile 408 /etc/haproxy/errors/408.http errorfile 500 /etc/haproxy/errors/500.http errorfile 502 /etc/haproxy/errors/502.http errorfile 503 /etc/haproxy/errors/503.http errorfile 504 /etc/haproxy/errors/504.http	millisecondes par défaut mais peut s'exprimer dans une autre unité de temps) <b>Timeout connect 5000</b> : temps d'attente de l'établissement d'une connexion vers un serveur (on abandonne si la connexion n'est pas établie après 5 secondes) <b>Timeout client 50000</b> : temps d'attente d'une donnée de la part du client <b>Timeout server 50000</b> : temps d'attente d'une donnée de la part du serveur  Les « errorfile » définissent les messages d'erreurs envoyés aux internautes.
--	--

\* D'autres modes existent comme « tcp » (connexions TCP génériques)

**log global** indique que l'on souhaite utiliser les paramètres de journalisation définis dans la section 'global'.

**option httplog** active la journalisation des requêtes HTTP.

**option dontlognull** : comme nous le verrons plus loin, HAProxy va se connecter régulièrement à chacun des serveurs afin de s'assurer qu'ils soient toujours vivants. Par défaut, chaque connexion va produire une ligne dans le journal qui sera ainsi pollué. Cette option permet de ne pas enregistrer de telles connexions pour lesquelles aucune donnée n'a été transférée.

*Ces options peuvent être désactivées dans chacune des autres sections avec un « no » devant.*

**La directive *listen* permet de créer un service de répartition de charge :**

Directives et valeurs	Quelques explications
<b>listen httpProxy 192.168.0.210:80</b> balance roundrobin option httpclose option httpchk HEAD / HTTP/1.0 server web1 10.22.100.212:80 check server web2 10.22.100.213:80 check	<b>listen</b> permet de demander à HAProxy d'écouter sur l'IP et le port indiqué. Si on ne précise que « :80 », le HAProxy écoute sur toutes les IPs de la machine, mais ici les requêtes ne pourront venir que de l'interface « externe » soit celle orientée vers les clients.  Voir ci-dessous pour les explications des autres directives.



**C'est HAProxy qui écoute sur le port 80** : tout service Web sur cette machine écoutant sur ce port ne sera pas possible et devra être arrêté.

**balance** permet de choisir l'algorithme de la répartition vers les frontaux.

Le **roundrobin** est le plus classique et le plus simple : il consiste à utiliser les serveurs un à un, chacun son tour. Il est possible d'affecter des poids particuliers aux ressources, par exemple pour utiliser deux fois plus souvent le frontal qui dispose d'une très grosse CPU et/ou de beaucoup de RAM.

Les autres modes possibles sont :

- **leastconn** : le serveur sélectionné sera celui ayant précédemment reçu le moins de connexions
- **source** : le serveur est sélectionné en fonction de l'IP source du client
- **uri** : le choix du serveur est fonction du début de l'URI demandée
- **url\_param** : le choix du serveur est fonction de paramètres présents dans l'URL demandée
- **hdr** : le choix du serveur est fonction d'un champ présent dans l'en-tête HTTP (Host, User-Agent, ...).

**option httpclose** force à fermer la connexion HTTP une fois la requête envoyée au client. On évite ainsi de conserver la connexion HTTP ouverte (« keep-alive ») et donc de renvoyer systématiquement cette dernière vers le même frontal tant que la connexion reste ouverte.

Conserver la connexion ouverte pourrait être le comportement recherché, mais, dans le cadre de notre Côté Labo, cela permettra de montrer facilement que l'algorithme « roundrobin » fonctionne bien et que l'on tombe sur un frontal différent à chaque rafraîchissement de la page.

**option httpchk**, suivie d'une requête HTTP, permet de vérifier qu'un frontal web est toujours en vie. HAProxy peut tester la disponibilité des serveurs en adressant une requête HTTP (ici, aucun fichier n'est précisé derrière le « / » présent après le « HEAD », HAProxy va envoyer la requête suivante à chaque serveur : `http://@IP_serveur/`).

Si un frontal venait à ne plus répondre à cette requête, il serait considéré comme hors service, et serait sorti du pool des frontaux ; aucun utilisateur ne serait redirigé vers lui.

Cette vérification est en fait activée grâce à l'option **check** de **server** (voir ci-après).

**server** déclare un serveur frontal, utilisé pour assurer le service. Chaque « server » est nommé (nom libre) et suivi de son IP/port de connexion (port qui pourrait être différent du port d'écoute de HAProxy).

L'option **check** est expliquée ci-dessus (dans **option httpchk**).



**Il est courant et parfois obligatoire de scinder la directive *listen* en deux sections *frontend* et *backend* (voir Annexe 2) :**

- **frontend** : définit les paramétrages de la partie publique. Les connexions « arriveront » donc ici.
- **backend** : définit la partie privée d'HAProxy. Apparaîtront ici le ou les serveurs Web sur lesquels portent la répartition.



## Test d'HAProxy

Pour vérifier que la syntaxe du fichier est correcte :

```
haproxy -c -f /etc/haproxy/haproxy.cfg
-c      pour vérifier (check) le fichier
-f      pour spécifier le fichier de configuration
```

```
root@haproxyAR:~# haproxy -c -f /etc/haproxy/haproxy.cfg
Configuration file is valid
```

Pour vérifier que le service fonctionne bien et écoute sur le port 80 avec la commande netstat, en n'affichant que la ligne correspondante à « haproxy » :

```
root@haproxyAR:~# netstat -tln | grep haproxy
tcp      0      0  192.168.0.210:80    0.0.0.0:*           LISTEN   8877/haproxy
```

haproxy est bien lancé et attend des requêtes sur le port 80.

Pour vérifier que le service a bien le comportement attendu, il suffit de se connecter à deux reprises à partir d'un navigateur.



**Attention au cache du navigateur** qui renvoie la page en cache au lieu de se connecter au serveur (ce qui ne permet donc pas de tester la répartition de charge) !  
Plusieurs solutions sont possibles dont :

- l'activation des touches CTRL+F5, sur « Chrome », pour recharger la page ;
- le changement de navigateur, voire de STA ;
- la possibilité d'utiliser le client « lynx » en ligne de commande :

lynx <http://192.168.0.210/appliFrais/cAccueil.php>

```
Intranet du Laboratoire Galaxy-Swiss Bourdin
Laboratoire Galaxy-Swiss Bourdin
Suivi du remboursement des frais du serveur Web 1 IntraLab
```

lynx <http://192.168.0.210/appliFrais/cAccueil.php>

```
<<< Intranet du Laboratoire Galaxy-Swiss Bourdin
Laboratoire Galaxy-Swiss Bourdin
Suivi du remboursement des frais du serveur Web 2 hdIntraLab
```

HAProxy envoie bien alternativement sur l'un et l'autre serveur Web

## Mise en place des statistiques

### Directives et valeurs

```
listen httpProxy 192.168.0.210:80
balance roundrobin
...
stats uri /statsHaproxy
stats auth apo:mdpstats
stats refresh 30s
```

### Quelques explications

**stats uri** permet d'activer la page de statistiques, en définissant l'endroit où les statistiques pourront être consultées (ici [http://@IP\\_du\\_haproxy/statsHaproxy](http://@IP_du_haproxy/statsHaproxy))

**stats auth** sécurise l'accès en le protégeant par un nom d'utilisateur et un mot de passe séparés par « : »

**stats refresh** rafraîchit la page toutes les 30s

Voir en *annexe 4* des exemples de statistiques que l'on peut obtenir.



Pour que les modifications soient prises en compte, il est bien sûr nécessaire de recharger le fichier de configuration : **service haproxy reload**.

## Annexe 2 : configuration avancée de HAProxy

### Utilisation des *frontends* et *backends*

Les sections *frontend* et *backend* remplacent la section *listen* et permettent d'affiner la configuration :

- **frontend** définit les paramétrages de la partie publique. Les connexions « arriveront » donc ici.
- **backend** : définit la partie privée de HAProxy. Apparaîtront donc ici le ou les serveurs Web qu'HAProxy sur lesquels portent la répartition.

Vous trouverez, dans la documentation officielle, les directives admissibles dans chacune des sections : <http://cbonte.github.io/haproxy-dconv/configuration-1.4.html#4.1>

Exemple :

```
frontend proxypublic
  bind 192.168.0.210:80
  default_backend fermeweb
```

```
backend fermeweb
  balance roundrobin
  option httpclose
  option httpchk HEAD / HTTP/1.0
  server web1 10.22.100.212:80 check
  server web2 10.22.100.213:80 check
  stats uri /statsHaproxy
  stats auth apo:mdpstats
  stats refresh 30s
```

Les directives *stats* peuvent être définies dans la section *defaults*, ainsi elles s'appliqueront à tous les *backend*.

**bind** définit le port d'écoute. Il y aura autant de directives *bind* que de ports d'écoute.

Ici, un seul backend est défini (*backend fermeweb*). Aussi, l'intérêt de scinder la directive *listen* en deux sections est limité mis à part une meilleure lisibilité notamment dans la lecture des statistiques.

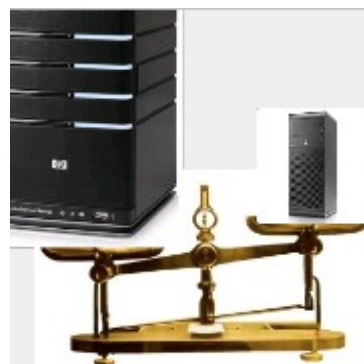
**default\_backend** définit le backend par défaut. Cette directive est obligatoire même s'il n'y a qu'un seul backend sinon HAProxy considère qu'aucun service n'est défini (erreur « 503 Service Unavailable »).

### Répartition inégale sur serveurs hétérogènes

Il est possible que le Cluster soit composé de serveurs disparates, et que l'on ne souhaite pas leur faire porter la même charge. Imaginons par exemple un processeur 2 fois moins puissant, une mémoire moins importante, etc.

Nous supposons ici que le serveur lamp2 est deux fois moins puissant que le serveur lamp1.

Il est possible de paramétrer HAProxy pour répartir la charge sur les serveurs, en fonction de leur puissance (algorithme *Weighted-least-connection*) en affectant aux serveurs une pondération différente, reflet de leur puissance : par exemple 100 pour lamp1 et 50 pour lamp2 (la valeur doit être entre 0 et 255).



Ce paramétrage est possible grâce au mot clé *weight* dans la directive *server* :

```
server web1 10.22.100.212:80 weight 100 check
server web2 10.22.100.213:80 weight 50 check
```

## La problématique des sessions

Le protocole HTTP utilise de nombreuses connexions TCP pour la session d'un même internaute. Avec la configuration élaborée, les requêtes d'un même internaute sont donc réparties entre les deux serveurs ce qui peut poser des problèmes lors de l'accès à certaines applications qui ont besoin de retrouver en mémoire des informations de *contexte*, relatives aux échanges précédents de l'internaute de la même session.

HAProxy peut effectuer une répartition de charge de manière à ce qu'un même utilisateur, dans le cadre d'une session, soit toujours redirigé vers le même frontal via un **cookie** : cookie existant dans l'application ou cookie créé par HAProxy.



Un **cookie** est une donnée (sous la forme **identifiant+valeur**) conservée sur le navigateur, à la demande du serveur et qui est retourné au serveur avec chacune des requêtes HTTP.

### Cookie ajouté par HAProxy

La configuration d'HAProxy permettant l'ajout d'un cookie dans les requêtes HTTP est la suivante :

```
backend fermeweb
    balance roundrobin
    option httpclose
    option httpchk HEAD / HTTP/1.0
    cookie QUELSEVEUR insert indirect
    server web1 10.22.100.212:80 cookie W1 check
    server web2 10.22.100.212:80 cookie W2 check
```

**QUELSEVEUR** sera  
l'identifiant du cookie  
**W1 et W2** seront les  
valeurs possibles

Le principe est très simple :

- À la première connexion de l'utilisateur sur le site, HAProxy va déterminer, selon l'algorithme sélectionné dans la configuration (ici roundrobin), vers quel serveur Web le rediriger.
- Quand HAProxy récupère auprès du serveur Web la copie de la page demandée par l'utilisateur, il la renvoie en y insérant un cookie QUELSEVEUR valant W1 ou W2 selon le serveur Web utilisé.
- Ainsi à la prochaine requête de notre utilisateur, son navigateur renverra également ce cookie avec la requête.
- HAProxy saura déterminer, en fonction de la valeur, vers quel frontal renvoyer l'utilisateur.
- Il fait aussi en sorte que des caches ne mémorisent pas ce cookie et de le cacher à l'application.
- À la fin de la session, le cookie est détruit.

**Nota** : Si l'application web gère elle-même ses propres cookies, le paramétrage est différent pour pouvoir les utiliser.

### Que se passerait-il si un des serveurs avait un problème et venait à planter ?

HAProxy sait déterminer que l'identifiant QUELSEVEUR pointe vers un serveur qui n'est plus actif dans le Cluster. Il détruit le cookie en lui réassignant une nouvelle valeur pour rediriger l'utilisateur vers un nouveau serveur web.

Certes la session qui était en cours sera alors perdue (il faudra se ré-authentifier) mais le service sera toujours rendu et le client aura toujours accès à l'application Web.

## Répartition de charge de niveau 7

Nous avons maintenant 2 serveurs se répartissant la charge, soit de manière égalitaire, soit en fonction d'un poids attribué à chacun d'eux.

Imaginons maintenant que l'on souhaite dédier un serveur particulier à la gestion d'une mailing liste, ou bien à la partie administrative de notre site, ou bien encore à une application web, le reste du site étant réparti sur les autres serveurs.

Nous pouvons mettre en place un **filtre applicatif via les « acl »** qui redirigera une URL particulière ou un port particulier sur un serveur.

Le principe est le suivant : les ACLs sont déclarées avec, au minimum, un nom, un test et une valeur valide à tester.

## Exemple d'utilisation d'ACL :

frontend proxypublic

bind 192.168.0.210:80

bind 192.168.0.210:5222

acl acces\_interface\_admin path\_beg /gestadm

use\_backend admin if acces\_interface\_admin

acl d1 dst\_port 5222

use\_backend JabberD1 if d1

default\_backend fermeweb

HAProxy vérifiera si le chemin donné par l'URL commence par /gestadm, auquel cas il utilisera le backend « admin ».

Il est possible d'ajouter d'autres chemins séparés par un espace sur la ligne de l'ACL.

backend admin

option httpchk HEAD / HTTP/1.0

server web3 192.168.100.100:80 check

backend JabberD1

server Server1ZoneD 192.168.100.45:80 check

D'innombrables autres règles peuvent être ajoutées. Les URL qui n'appartiennent à aucune règle sont envoyées sur le backend par défaut.

Attention en cas d'acl sur le port de destination, haproxy doit aussi écouter sur ce port. Il faut donc deux options « bind ».

## Annexe 3 : exemples de statistiques

### Statistics Report for pid 1918

#### > General process information

pid = 1918 (process #1, nproc = 1)  
uptime = 0d 0h02m41s  
system limits: memmax = unlimited; ulimit-n = 4014  
maxsock = 4014; maxconn = 2000; maxpipes = 0  
current conns = 1; current pipes = 0/0  
Running tasks: 1/4

active UP      backup UP  
active UP, going down      backup UP, going down  
active DOWN, going up      backup DOWN, going up  
active or backup DOWN      not checked  
active or backup DOWN for maintenance (MAINT)

Note: UP with load-balancing disabled is reported as "NOLB".

Display option:

- [Hide "DOWN" servers](#)
- [Refresh now](#)
- [CSV export](#)

External resources

- [Primary site](#)
- [Updates \(v1\)](#)
- [Online man](#)

#### publicproxy

	Queue	Session rate	Sessions	Bytes	Denied	Errors	Warnings	Server
	Cur Max Limit	Cur Max Limit	Cur Max Limit Total LbTot	In Out	Req Resp	Req Conn Resp Retr Redis	Status	LastChk Wght Act Bek Chk Dwn DwnTime
Frontend	1 0 0	0 0 0	1 1 2 000 49	9 790 31 985	0 0 0	0 0 0	0 0 0	OPEN

#### fermeweb

	Queue	Session rate	Sessions	Bytes	Denied	Errors	Warnings	Server
	Cur Max Limit	Cur Max Limit	Cur Max Limit Total LbTot	In Out	Req Resp	Req Conn Resp Retr Redis	Status	LastChk Wght Act Bek Chk Dwn DwnTime
web1	0 0 0	- 0 3	0 1 - 22 22 4 591 10 044	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	2m41s UP	L7OK/200 in 2ms 1 Y - 0 0 0s
web2	0 0 0	- 0 1	0 1 - 4 4 668 1 888	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	38s DOWN	L4CON in 0ms 1 Y - 2 1 38s
web3	0 0 0	- 0 3	0 1 - 21 21 4 239 9 888	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	2m41s UP	L7OK/200 in 1ms 1 Y - 0 0 0s
Backend	0 0 0	1 6	1 1 0 49 47 9 790 31 985	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	2m41s UP	2 2 0 0 0 0s

### On distingue deux blocs :

- Le bloc frontend (publicproxy)
- Le bloc backend (fermeweb) disposant de trois serveurs web, deux en « vert » et un en « rouge ».

### On constate ici que :

- les serveurs totalisent 49 sessions, dont 22 pour web1, 4 pour web2 et 21 pour l'hôte web3.
- le nombre de sessions faible pour web2 s'explique par le fait qu'il est tombé. Après 2 « check » infructueux, il a été déclaré down, et cela depuis 38s ;
- les frontaux web1 et web2 sont UP depuis 2 min 41s ;
- les frontaux web1, web2, et web3 ont eu respectivement 3, 1 et 3 sessions simultanées maximum ;
- on a également une indication sur les quantités de données qui ont transité vers et depuis chaque frontal.

Même une fois web2 redémarré, la trace de son inaccessibilité reste, ce qui est fort instructif pour un administrateur réseau.

Dans le tableau suivant, on voit que le frontal web2 est resté inaccessible pendant 6 minutes, et qu'il a redémarré depuis 5 minutes 36 :

# Statistics Report for pid 1918

## General process information

active UP

active UP, going down

active DOWN, going up

active or backup DOWN

active or backup DOWN for maintenance (MAINT)

backup UP

backup UP, going down

backup DOWN, going up

not checked

Note: UP with load-balancing disabled is reported as "NOLB".

Display option:

[Hide 'DOWN' servers](#)
[Refresh now](#)
[CSV export](#)

External resources:

[Primary site](#)
[Updates \(v1.4\)](#)
[Online manual](#)

pid = 1918 (process #1, nbproc = 1)

uptime = 0d 0h13m40s

system limits: memmax = unlimited; ulimit-n = 4014

maxsock = 4014; maxconn = 2000; maxpipes = 0

current conns = 1; current pipes = 0/0

running tasks: 1/4

publicproxy

	Queue		Session rate			Sessions				Bytes		Denied		Errors		Warnings		Server												
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bek	Chk	Dwn	Dwntme	Ti	
Frontend	1	6	-	1	1	2 000	58					11 543	56 202	0	0	0					OPEN									

fermeweb

	Queue		Session rate			Sessions				Bytes		Denied		Errors		Warnings		Server											
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bek	Chk	Dwn	Dwntme	Ti
web1	0	0	-	0	3	0	1	-	24	24	4 925	10 988	0	0	0	0	0	0	0	0	13m40s UP	L7OK/200 in 1ms	1	Y	-	0	0	0s	
web2	0	0	-	0	1	0	1	-	6	6	1 002	2 832	0	0	0	0	0	0	0	0	5m36s UP	L7OK/200 in 2ms	1	Y	-	2	1	6m1s	
web3	0	0	-	0	3	0	1	-	24	24	4 740	11 284	0	0	0	0	0	0	0	0	13m40s UP	L7OK/200 in 3ms	1	Y	-	0	0	0s	
Backend	0	0	-	1	6	1	1	0	58	54	11 543	56 202	0	0	0	0	0	0	0	0	13m40s UP		3	3	0	0	0s		