

Assurer la qualité

La qualité d'un site Web, en dehors de tout aspect esthétique ou de pertinence du contenu, est jugée par les clients sur la vitesse et sur la disponibilité. Une entreprise dont le site se révèle au mieux lent, au pire indisponible aux yeux des utilisateurs, passe souvent pour techniquement incompétente voire méprisante envers ses clients.

Disponnibilité

Pour un site commercial, il est bien évidemment inconcevable que celui-ci devienne inaccessible pour cause de panne (disque dur défaillant...). Certes le 0 pannes est un idéal inaccessible quelquesoit la solution envisagée. Pour autant, une solution simple et ayant fait ses preuves existe et celà quelque soit le domaine : la **redondance**.

La redondance pour un serveur Web ou un serveur intranet peut se situer à plusieurs niveaux :

- Disques RAID, Alimentations pluggables à chaud, onduleurs...
- Multiplier les serveurs

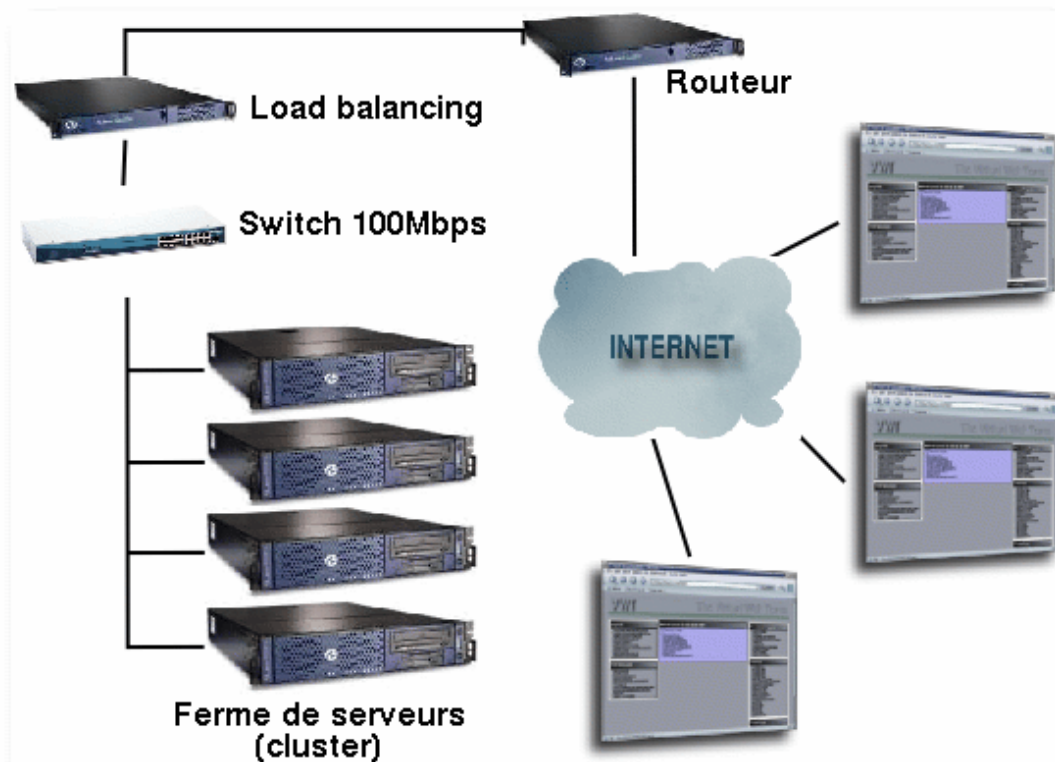
Vélocité

La temps mis à l'affichage de la première page d'un site ne doit pas dépasser ce que l'on désignera par '**temps psychologique**'. En 2001 il était estimé à 8 secondes, alors que l'ADSL ne représentait encore qu'une faible part de marché. Il va fort probablement encore diminuer (accoutumance). La qualité d'un site découle donc directement de sa **vélocité**, et plusieurs solutions permettent de l'améliorer :

- réécrire les applications
- changer de langage (asp ou coldfusion -> php)
- changer de serveur pour un serveur plus puissant
- multiplier les serveurs...

Solution ?

Que ce soit dans le cas de la **disponnibilité** ou de la **vélocité**, nous avons vu qu'une des solutions possibles était de multiplier les serveurs. La technique appelée **répartiton de charge** permet de réaliser des '**fermes de serveurs**' ou '**clusters**' vus comme un seul serveur et satisfaisant au mieux à la problématique **disponnibilité/vélocité**.



Solution classque de répartition de charge (load balancing). Un serveur pouvant être aussi chargé d'assurer un rôle de firewall, assure la répartition de charge.



Un peu de théorie

Les applications Web modernes utilisent le plus souvent un mécanisme de persistance (les fameuses sessions en PHP par exemple). HTTP étant un protocole non connecté, un répartiteur de charge doit pouvoir assurer qu'au cours d'une session limitée dans le temps, un utilisateur donné devra être reconnecté au serveur ayant effectué la transaction initiale. Plusieurs solutions possibles ont été mise en oeuvre.

Algorithmes déterministes

- **Tables de hash** : On construit une table de hash à partir de l'adresse IP du client. Cet algorithme suppose que le client ne passe pas au travers de différents proxies (adresse IP variable), sinon il n'est valable qu'au cours d'une session limitée dans le temps. N'assure pas forcément une répartition homogène
- **Redirection** : Les clients sont envoyés vers un serveur de redirection, celui-ci les redirigeant vers un serveur selon un algorithme qui lui peut ne pas être déterministe. Le problème peut alors survenir si le serveur vers lequel le client a été redirigé est indisponible. Peu fiable donc.

Algorithmes non-déterministes

Les algorithmes non-déterministes ne permettent pas la persistance à eux seuls et nécessitent un mécanisme auxiliaire pour assurer cette tâche. Par contre ils offrent la répartition de charge la plus homogène et offrent les meilleures performances sur les connexions permanentes (ftp, smtp ...)

- **Round Robin** : ou 'tourniquet' est comme son nom l'indique un algorithme de file tournante : l'équilibrage choisit à chaque instant un serveur dans la file et boucle sur celle-ci. Certainement le plus performant.
- **Least Conn** : Le serveur renvoie vers le serveur le moins chargé. Si en théorie il semble le plus adapté, en réalité dans le cadre du Web dynamique, un serveur peut être considéré comme chargé alors que les processus sont en attente d'une requête vers une base de données...
- **First Respons** : Les requêtes clients sont envoyés simultanément à tous les serveurs et le premier qui répond sera chargé de la connexion. Difficile à mettre en oeuvre et rarement employé.

Persistance

A partir du moment où l'on décide d'utiliser un algorithme non-déterministe (plus performant), se pose alors le problème de la persistance. Deux solutions s'offrent à nous.

Cookies

Le serveur cible émet un cookie contenant un identifiant vers le client. L'équilibreur de charge pourra ensuite retrouver le serveur initial en demandant au client de présenter le cookie. Si cette méthode présente l'avantage d'être simple à mettre en oeuvre, elle n'est pas fiable en pratique puisque que rien ne garantit que le client accepte les cookies... et que les règles de bonne conduite interdisent de forcer l'utilisateur à les accepter pour pouvoir accéder au site.

Tracking

Le répartiteur de charge affecte un serveur par Round Robin ou tout autre algorithme, et mémorise le lien dans une table. A la connexion suivante si le client est reconnu (par son adresse IP par exemple) le serveur le reconnecte au serveur précédemment accédé. Tout comme l'algorithme déterministe de hashage, seule une session limitée dans le temps peut être gérée. C'est la solution adoptée par **PEN**

PEN

PEN est un load balancer agissant sur les protocoles de la couche TCP (http, snmp, pop3...).

Installation



<http://siag.nu/pen/>

Le site officiel de PEN

Téléchargez le source le plus récent ici :



<ftp://siag.nu/pub/pen/>

Archives TGZ

La version téléchargée pour l'essai est la 0.9.5

La suite se déroule sans problème :

```
# ./configure
# make
# make install
```

Par défaut PEN s'installe dans /usr/local/bin. Si vous désirez le mettre ailleurs, un ./configure --prefix=chemin habituel fera l'affaire

Load balancer des serveurs Web

Bien qu'apte à répartir la charge pour différents protocoles utilisant la couche TCP (SMTP, POP3, LDAP ...) nous nous intéresserons ici qu'à la répartition de charge pour le protocole HTTP, et donc vérifier en quoi nos applications PHP peuvent profiter de cette technologie.

PEN et les connexions persistantes

PEN possède une table permettant de stocker les connexions effectuées. Cette table peut stocker par défaut 2048 connexions (différentes). Lors d'une connexion par un client, PEN vérifie dans la table si il a déjà réalisé une connexion correspondant à l'ip du client. Si c'est le cas, il appelle le serveur ayant déjà traité cette connexion assurant ainsi le maintien de la session. Sinon, il effectue un Round Robin classique et affecte un serveur à la connexion entrante.

La table des connexions est une file tournante, ce qui signifie donc que lorsque celle-ci est pleine, PEN écrase l'entrée la plus ancienne.

Mise en oeuvre

Pour lancer PEN, il est nécessaire de lui fournir quelques paramètres sur la ligne de commande (hummm un fichier pen.conf aurait été le bienvenu...). La syntaxe et les paramètres importants sont les suivants :

```
pen l bhost:80 host1:80 host2:80
```

Autres paramètres optionnels :

- -l penlog.log : fichier de log
- -i penlog.log : génère un fichier contenant le PID de PEN, utile pour les autres utilitaires livrés avec.
- -c N : Nombre maximum de clients dont l'on veut suivre la trace. Par défaut 2048.

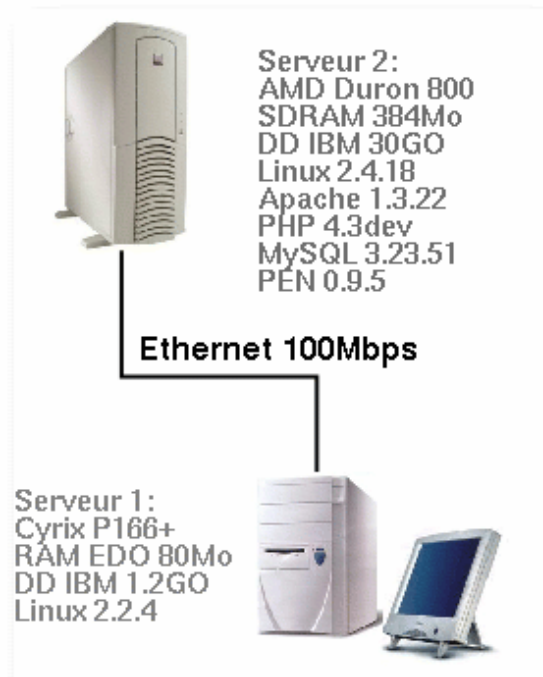
Il existe tout plein d'autres options que je vous laisserai découvrir (taper ./pen sans paramètres). Sachez que PEN permet une administration et une visualisation des statistiques internes.

FAILSAFE!

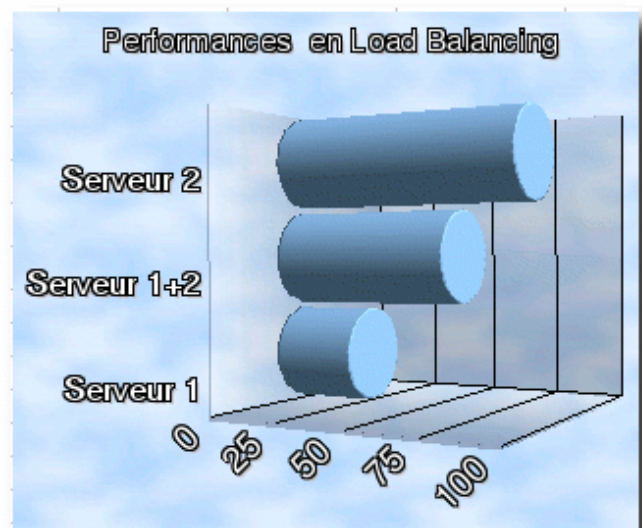
Lorsque PEN détecte qu'un serveur n'est plus actif, qu'il ne répond plus, il sélectionne automatiquement le serveur utilisé le moins récemment. Ce qui permet d'assurer une presque parfaite disponibilité... Car que se passerait-il si le serveur sur le quel tourne PEN tombe lui même en panne ? La solution existe : installer un routeur virtuel(vrpp). Voir urls en fin d'article.

Le test

Le test a été effectué sur le script d'affichage du calendrier du jour (voir article sur la classe calendar dans le même numéro). La configuration utilisée était la suivante :



Résultats



Le serveur 1, testé seul permet 29 requêtes/s, le serveur 2 testé seul, permet 90 requêtes/s. Si l'on force PEN à effectuer un Round Robin (requêtes distribuées alternativement sur les 2 serveurs) le résultat est de 64 requêtes secondes. Certes, la configuration générale est très déséquilibrée, mais dans une véritable configuration, les 2 serveurs devraient posséder une puissance sensiblement équivalente.

Résultats (suite)

On peut conclure que la répartition des requêtes sur les 2 serveurs augmente bien la puissance apparente du serveur 1 et qu'elle diminue la puissance apparente du serveur 2. Mais dans le cas où nous serions passé d'une configuration serveur 1 seul à serveur 1 + serveur 2, nous avons bien un gain en puissance, et une disponibilité accrue.

Un petit calcul

Soit P_1 le nombre de requêtes/s du serveur 1 et P_2 pour le serveur 2. En théorie le nombre de requêtes/s pouvant être servies par un système à répartition de charge devrait être dans le cas du Round Robin :

- $P_{\min} = \min(P_1, P_2)$
- $P_{\text{Tot}} = P_{\min} \times 2$
soit dans notre cas :
- $P_{\text{Tot}} = 29 \times 2 = 58$ requêtes secondes
Ce qui somme toute est assez proche du résultat obtenu.

Pour des serveurs dont la puissance est homogène, on peut donc conclure que théoriquement la puissance apportée par N serveurs est :

- $P_{\text{Tot}} = P_1 + P_2 + \dots + P_n$
soit :
- $P_{\text{Tot}} = P \times N$

Dans les faits il convient de multiplier P_{Tot} par une constante K dont la valeur sera presque toujours < 1 . En pratique une valeur tel que $0.7 < K < 0.9$

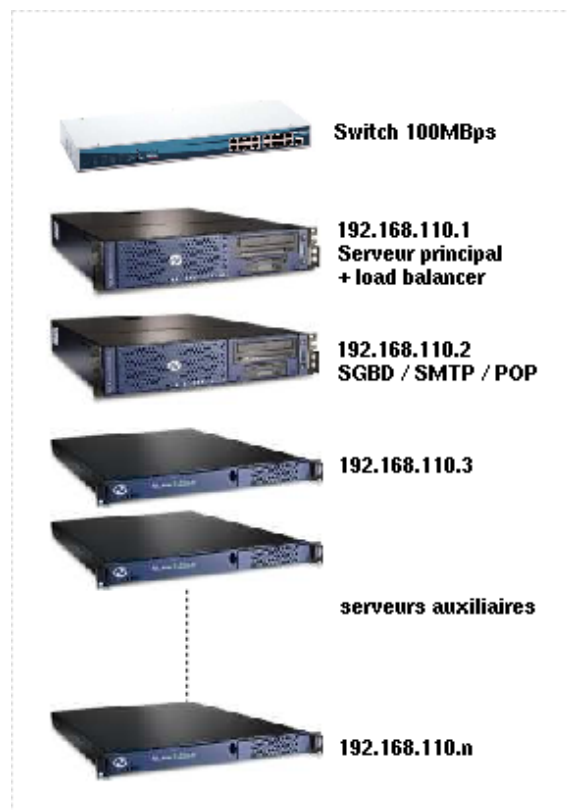
Un serveur performant

PEN ou tout autre Load Balancer adéquat permet de constituer des serveurs très performants, et ceci de façon relativement économique.

La figure ci-contre décrit une configuration typique pour intra/extranet. On notera les points suivants :

- Il est absolument nécessaire d'installer le SGBD sur un serveur séparé, voire dans le cas d'applications hautes-performances d'avoir un cluster pour le SGBD...
- Un cluster impose une politique et des outils efficaces permettant de d'assurer la synchronisation des données.
- Il est absolument nécessaire d'installer le SGBD sur un serveur séparé, voire dans le cas d'applications hautes-performances d'avoir un cluster pour le SGBD...
- Il est nécessaire d'implémenter le Load Balancer de telle façon qu'en cas de chute du serveur principal un autre serveur prenne le relais.

- Dans les cas extrêmes il serait bon que le Load Balancer puisse rediriger les requêtes vers des serveurs géographiquement éloignés, soit pour diminuer les sauts successifs au travers des DNS (cas d'un client/utilisateur indonésien ou bolivien servis chacuns par un serveur le plus proche de lui), soit en cas de défaillance critique d'un sous-système du cluster.



Conclusion

Pour un coût relativement bas, il est donc possible sous Linux de réaliser un cluster de serveurs Web permettant ainsi à la fois une meilleure disponibilité tout en accroissant la puissance. Ainsi atteindre le million de pages vues/mois est possible avec une configuration minimale répartissant la charge sur 2 serveurs à base de processeurs PIII 900 + un serveur SGBD sur BI-PIII...

Liens utiles



<http://linux-ha.org/>
La haute disponibilité sous Linux



<http://w3.arobas.net/~jetienne/vrrpd/index.html>
VRRPd

KDO kdo@zephpmag.com