# L'EVOLUTIVITE APPLICATIVE PAR LA REPARTITION DE CHARGE

Septembre 2006
Willy Tarreau
w@1wt.eu

Revision 1.0
http://1wt.eu/articles/2006_lb/

TABLE DES MATIERES


Pourquoi cet article 3
Introduction 3
Qu'est ce que la répartition de charge (load balancing) ? 3
Techniques de répartition de charge 4
le DNS 4
Réduire le nombre d'utilisateurs par serveur 5
Surveiller les serveurs 6
Choisir le meilleur serveur 6
Persistance 7
L'apprentissage de Cookie 7
Insertion de cookie 8
Limites de le persistance – le SSL 8
Une ferme SSL dédiee 10
Comment choisir entre un répartiteur matériel ou logiciel ? 11
nombre de sessions 12
Répartition de charge de niveau 3/4 13
Répartition de charge de niveau 7 14
la meilleure combinaison, pour ceux qui peuvent se l'offrir 17
Optimisation applicative indépendamment du répartiteur 18
Séparer les contenus statiques et dynamiques 18
Ce qui peut être optimisé coté serveur - Exemple avec Apache 18
Plus de lecture sur le sujet de la répartition de charge 19

## Pourquoi cet article

En tant qu'auteur du logiciel open source de répartition de charge HAProxy[1], on me pose souvent des questions au sujet des choix d'architectures ou des choix de produits de répartition de charge. Ces questions n'ayant aucune réponse simple et évidente, il est préférable de tenter d'exposer les avantages et les inconvénients de certaines solutions, mais également de mettre en garde contre certains pièges faciles à éviter. J'espère être rapidement en mesure de complèter cet article avec une analyse plus poussée du protocole HTTP et quelques exemples d'architectures. En attendant, il vous est possible de me contacter directement pour toute question ou demande d'information complémentaire.

## Introduction

Initialement, le contenu du Web était principalement fait d'objets statiques, rapidement fournis à quelques utilisateurs qui passaient le plus clair de leur temps à lire plutôt qu'à cliquer. Aujourd'hui, à l'inverse, nous voyons de vraies applications qui retiennent les utilisateurs de longues minutes voire des heures, avec peu de contenu à lire entre les « clics » et générant un lourd travail réalisé par les serveurs.

Les utilisateurs visitent souvent les mêmes sites qu'ils connaissent donc parfaitement et vont à l'essentiel, ne perdant plus de temps à lire. Ils exigent des réponses rapides en infligeant inconsciemment aux serveurs de fortes charges à chaque clic de souris.

Cette nouvelle dynamique de consommation du Web a donc engendré un nouveau besoin de haute performance et de disponibilité permanente des applications.

## Qu'est ce que la répartition de charge (load balancing) ?

Alors que la puissance de tout serveur est limitée, une application Web doit être en mesure de fonctionner sur plusieurs serveurs pour supporter une augmentation constante du nombre des utilisateurs. C'est ce qu'on appelle l'évolutivité applicative (*scalability*). La gestion de cette évolutivité n'est généralement pas un réel problème pour les applications intranet puisque le nombre d'utilisateurs est connu et a fort peu de chances d'augmenter soudainement.

En revanche, sur les portails Internet, la croissance permanente du nombre d'utilisateurs connectés, tout comme la banalisation des accès haut débit augmentent fortement la charge. Le responsable du site doit ainsi trouver des solutions pour répartir cette charge sur plusieurs serveurs, soit par le biais de mécanismes incorporés aux applications, soit en utilisant des composants externes ou bien, en réorganisant son architecture. La répartition de charge est donc la capacité d'utiliser plusieurs serveurs qui délivrent le même service et font le même travail. Mais cette augmentation du nombre de serveurs augmente le risque de dysfonctionnement , risque qui doit donc être traité comme il se doit. La capacité de garantir la disponibilité du service face a de multiples dysfonctionnements est qualifiée de haute disponibilité. Cette capacité est bien souvent indispensable dans le cadre de la répartition de charge ce qui est la raison principale pour laquelle la majorité des gens mélangent souvent ces deux concepts.

Néanmoins, certaines techniques de répartition de charge ne fournissent aucune haute disponibilité et sont donc dangereuses en production.

[1] http://haproxy.1wt.eu/

## Techniques de répartition de charge

### LE DNS

Le moyen le plus simple d'opérer une répartition de charge est de dédier des serveurs pour des groupes d'utilisateurs prédéfinis.

Cette méthode est facile à mettre en oeuvre pour un Intranet mais beaucoup plus complexe pour des serveurs Internet. Une approche commune repose sur l'utilisation du DNS en rotation (*round robin*).

Si un serveur DNS possède plusieurs adresses pour un nom de serveur particulier, il les proposera ainsi à tour de rôle en effectuant un cycle.

De cette manière, des utilisateurs différents verront des adresses différentes pour le même nom de serveur et seront ainsi capables d'atteindre des serveurs différents de manière transparente. Ceci est très fréquemment utilisé pour de la répartition de charge en mode multi sites, mais ne peut fonctionner que pour une application qui ne nécessite pas de gestion de contexte. Pour cette raison, cette méthode est généralement utilisée pour des moteurs de recherche, des serveurs POP ou pour la présentation de contenu statique.

Cette méthode ne fournit aucun moyen de disponibilité. Elle nécessite donc des moyens additionnels pour tester en permanence l'état des serveurs et pouvoir ainsi basculer l'adresse d'un serveur défectueux vers un autre.

Pour cette raison, elle est généralement considérée comme une solution complémentaire mais rarement comme une solution de répartition de charge principale.

```
$ host -t a google.com
google.com. has address 64.233.167.99
google.com. has address 64.233.187.99
google.com. has address 72.14.207.99

$ host -t a google.com
google.com. has address 72.14.207.99
google.com. has address 64.233.167.99
google.com. has address 64.233.187.99

$ host -t a mail.aol.com
mail.aol.com. has address 205.188.162.57
mail.aol.com. has address 205.188.212.249
mail.aol.com. has address 64.12.136.57
mail.aol.com. has address 64.12.136.89
mail.aol.com. has address 64.12.168.119
mail.aol.com. has address 64.12.193.249
mail.aol.com. has address 152.163.179.57
mail.aol.com. has address 152.163.211.185
mail.aol.com. has address 152.163.211.250
mail.aol.com. has address 205.188.149.57
mail.aol.com. has address 205.188.160.249
```

**Fig.1** : Rotation DNS : plusieurs addresses pour un même nom de serveur, fournies dans un ordre cyclique

**REDUIRE LE NOMBRE D'UTILISATEURS PAR SERVEUR**

Une approche plus courante consiste à répartir la population des utilisateurs sur plusieurs serveurs. Ceci implique donc la présence d'un répartiteur de charge entre les utilisateurs et les serveurs. Ce dernier peut prendre la forme d'un matériel, d'un logiciel installé sur un serveur dédié ou directement sur les serveurs d'application. Dans ce cas, il est important de noter que le déploiement de nouveaux composants augmente également le risque de dysfonctionnement. Une pratique courante consiste donc à disposer d'un second répartiteur en backup du premier.

Généralement, un répartiteur matériel fonctionnera au niveau des paquets réseau, agissant sur le routage et utilisant l'une des deux méthodes suivantes :

- **Routage direct** : Le répartiteur route la même adresse de service au travers de différents serveurs physiques locaux qui doivent être sur le même segment de réseau et doivent partager la même adresse de service. Cette méthode possède l'énorme avantage de ne nécessiter aucune modification au niveau IP, faisant en sorte que les serveurs répondent directement à l'utilisateur sans passer par le répartiteur. C'est ce qu'on appelle le « retour direct du serveur » (*Direct Server Return - DSR*) . Comme la puissance de traitement requise est minime, cette méthode est couramment utilisée sur les serveurs frontaux des sites à fort trafic. D'un autre coté, elle requiert de solides compétences du modèle TCP/IP pour obtenir une configuration correcte et optimale.

- **Tunnelling** : Cela fonctionne exactement comme le routage direct à la différence près que des serveurs distants peuvent être utilisés grâce à l'utilisation de tunnels établis entre ces derniers et le répartiteur de charge. Dans ce cas, le retour direct du serveur est également possible.

- **Translation d'adresse IP (NAT)** : L'utilisateur se connecte sur une adresse de destination virtuelle qui est ensuite convertie par le répartiteur en adresse réelle de l'un des serveurs. A première vue, cette méthode semble la plus simple à déployer puisqu'elle ne requiert aucune configuration du serveur. En revanche, elle requiert des règles de programmation applicative très strictes. Une erreur très courante est celle des serveurs qui retournent leur adresse interne dans leurs réponses. Cette méthode provoque également plus de charge sur le répartiteur qui doit convertir les adresses dans les deux sens, maintenir une table des sessions et supportera le trafic de retour. Parfois, il arrive qu'un paramétrage trop court des durées d'expiration des sessions provoque des effets de bord connus sous le nom « ACK storms ». Dans ce cas, la seule solution consiste à augmenter le délai d'expiration, en prenant le risque de saturer la table des sessions du répartiteur de charge.

A l'opposé, on trouvera les répartiteurs de charge logiciels. Le plus souvent, Ils fonctionnent comme des « reverse proxy », prétendant être le serveur et ayant ensuite pour rôle de lui transmettre le trafic. Ceci implique que les serveurs eux-mêmes ne puissent être directement joignables par les utilisateurs, mais également que certains protocoles ne seront pas gérés par le répartiteur de charge.

Ces logiciels nécessitent plus de puissance que les solutions matérielles agissant au niveau du réseau mais comme ils séparent les communications entre les utilisateurs et les serveurs, ils fournissent ainsi un premier niveau de sécurité en ne transmettant que ce qu'ils comprennent. C'est la raison pour laquelle on retrouve souvent des fonctionnalités de filtrage d'URL associées à ces produits.

#### SURVEILLER LES SERVEURS

Pour être en mesure de choisir un serveur, un répartiteur doit savoir lesquels sont disponibles. Pour cela, il leur enverra régulièrement des « ping », des tentatives de connexion, des requêtes, ou tout type de test que l'administrateur considérera comme représentatif de leur état de fonctionnement. Ces tests sont généralement appelés des « Health checks ».

Un serveur défaillant peut parfois répondre aux tests tels que le « ping » mais pas aux connexions TCP; de la même façon, il peut parfois répondre à ces connexions mais pas aux requêtes HTTP. Lorsqu'il s'agit d'un serveur d'application Web multi niveaux, certaines requêtes HTTP provoqueront une réponse immédiate alors que d'autres pourront échouer.

Ainsi, il y a un grand intérêt à choisir les tests les plus représentatifs qui seront supportés à la fois par l'application et par le répartiteur. Certains tests peuvent permettre de récupérer des données depuis des bases de données afin de s'assurer que l'ensemble de la chaîne reste valide. L'inconvénient de cette pratique sera une plus grande consommation de ressources du serveur (processeur, threads, …).

Ces tests devront donc être suffisamment espacés dans le temps pour éviter une trop grande surcharge, mais néanmoins suffisamment rapprochés pour détecter rapidement un dysfonctionnement. Cette méthode de « Health checks » est l'un des aspects les plus compliqués de la répartition de charge. Il est ainsi courant qu'après quelques tests, les développeurs de l'application décident d'implémenter une requête spéciale, dédiée au répartiteur, qui réalisera des tests internes plus représentatifs.

Dans ce domaine, les répartiteurs logiciels sont nettement les plus flexibles, fournissant généralement la possibilité de créer des scénarios automatisés qui pourront être rapidement modifiés et corrigés en très peu de temps.

#### CHOISIR LE MEILLEUR SERVEUR

Un répartiteur de charge peut distribuer la charge de multiples façons. Une idée reçue très répandue est qu'il enverra la requête au serveur qui répond le plus rapidement.

Cette pratique doit être évitée car si un serveur a de bonnes raisons de répondre plus rapidement, il risque de déséquilibrer la ferme de serveurs en récupérant la majorité des requêtes.

Une autre idée fréquemment rencontrée consiste à envoyer une requête au serveur le moins chargé. Bien que cela soit utile dans des environnements qui supposent des sessions de longue durée, ce n'est nullement adapté pour des serveurs dont la charge peut varier d'un facteur important en quelques secondes.

Pour les fermes de serveurs homogènes, la méthode cyclique (round robin) est bien souvent la meilleure.

Elle utilise chaque serveur à la suite. SI les serveurs sont de capacité inégale, l'utilisation d'une pondération (*weighted round robin*) permettra d'assigner le trafic aux serveurs en tenant compte de leur capacité relative.

Ces algorithmes présentent néanmoins un inconvénient : ils ne sont pas déterministes. Cela signifie que deux requêtes successives d'un même utilisateur auront une grande chance d'aboutir sur deux serveurs différents.

Dans le cas ou le contexte utilisateur est stocké sur le serveur d'application, il sera donc perdu entre les deux requêtes. Ainsi, lorsque des initialisations de sessions complexes sont utilisées (ex : négociation de clé SSL), celle-ci devront être systématiquement rejouées à chaque connexion.

Pour contourner ce problème, un algorithme très simple est parfois utilisé : le hachage d'adresse IP (*address hashing*).

Pour le décrire brièvement, il s'agit de diviser l'adresse IP de l'utilisateur par le nombre de serveurs, la résultat définissant le serveur devant être sélectionné pour cet utilisateur. Ceci

fonctionne tant que le nombre de serveurs ne change pas et que l'adresse IP de l'utilisateur reste stable ce qui n'est pas toujours le cas.

En cas de défaillance d'un serveur, tous les utilisateurs sont répartis à nouveau et perdent leurs sessions. Le faible pourcentage d'utilisateurs qui naviguaient à travers la ferme de proxies ne pourront alors plus utiliser l'application (env 5-10%). Cette méthode n'est pas toujours applicable ; en effet, pour assurer une distribution correcte, il est nécessaire de disposer d'un nombre important d'adresses IP source.

Ceci est vrai sur Internet, mais pas toujours dans le cas de petites sociétés ou même au sein de quelques infrastructures de fournisseurs d'accès Internet. Quoi qu'il en soit, cette solution est très efficace pour éviter un calcul trop fréquent des clés de session SSL.

## Persistance

La solution pour contourner les limites décrites ci-dessus est d'utiliser la persistance des sessions. La persistance est un moyen pour s'assurer qu'un utilisateur donné se connectera toujours sur un même serveur où son contexte est connu. Une solution économique consiste a faire en sorte que l'application redirige la demande vers l'adresse locale du serveur en utilisant une réponse HTTP de type 302.

L'inconvénient majeur est qu'en cas de défaillance du serveur, l'utilisateur ne dispose d'aucune échappatoire et essayera en vain de contacter le serveur. Tout comme avec le DNS, cette méthode n'est utile que pour les sites importants où l'adresse donnée à l'utilisateur sera toujours joignable.

Une seconde solution est de demander au répartiteur d'apprendre les associations utilisateur-serveur, la manière la plus simple pour lui étant d'enregistrer sur quel serveur s'est connecté l'adresse IP de l'utilisateur lors de la dernière connexion. Ceci résoud généralement le problème lié aux modifications du nombre de serveurs de la ferme à cause des dysfonctionnements mais ne résout pas le problème de gestion des utilisateurs disposant d'adresses IP changeantes.

Que reste-t-il ? Depuis le début, nous affirmons que nous essayons de garantir qu'un utilisateur retrouvera son contexte lors de connexions successives à un même serveur. Comment ce contexte est il identifié par le serveur ? Grâce à un « cookie ».

Les « cookies » ont été inventés spécifiquement pour cela : donner une information à l'utilisateur qui la retransmettra à son retour de façon à ce que nous sachions comment le gérer. Bien sur, cela implique que l'utilisateur supporte et accepte les cookies mais c'est généralement le cas pour des applications qui doivent gérer la persistance.

SI le répartiteur peut alors identifier le serveur en fonction du cookie présenté par l'utilisateur, cela devrait permettre de résoudre la plupart des problèmes exposés.

Il existe principalement deux approches en ce qui concerne les cookies : le répartiteur peut apprendre le cookie de session assigné par les serveurs, ou il peut insérer un cookie pour identifier le serveur.

#### L'APPRENTISSAGE DE COOKIE

Cette méthode est la moins intrusive. Le répartiteur est configuré pour apprendre le cookie de l'application (ex : « JSESSIONID »). Quand il reçoit la requête de l'utilisateur, il vérifie si elle contient ce cookie avec une valeur connue. Si tel n'est pas le cas, il dirigera la requête vers n'importe lequel des serveurs en fonction de l'algorithme de répartition appliqué. Il récupérera alors la valeur du cookie à partir de la réponse du serveur et l'ajoutera dans sa table des sessions avec l'identifiant du serveur correspondant. Quand l'utilisateur revient, le répartiteur voit le cookie, vérifie sa table de sessions et trouve le serveur associé vers lequel il redirige la requête.

Cette méthode, bien que très utilisée, présente deux inconvénients mineurs dus à l'aspect apprentissage de cette méthode :

- Le répartiteur dispose d'une mémoire limitée, il peut donc être saturé et la seule solution est donc de limiter la durée de vie d'un cookie dans la table des sessions. Ceci implique que si l'utilisateur revient après l'expiration du cookie, il aura toutes les chances d'être redirigé vers le mauvais serveur.

- Si le répartiteur primaire meurt et que son backup prend la main, ce dernier n'aura aucune association existante dans sa table et dirigera donc l'utilisateur vers le mauvais serveur. Bien sur, il n'est pas possible d'utiliser les répartiteurs en mode actif-actif. Une solution consisterait à synchroniser les sessions en temps réel, ce qui est très difficile à garantir.

Pour contourner ces inconvénients, on préfère souvent utiliser un algorithme déterministe de répartition tel que le « address hashing » de l'adresse IP de l'utilisateur quand c'est possible. De cette façon,dans le cas ou le cookie serait perdu par le répartiteur, les utilisateurs qui disposent d'une adresse fixe seront quand même correctement dirigés sur leur serveur.

#### INSERTION DE COOKIE

Si les utilisateurs supportent les cookies, pourquoi ne pas en ajouter un contenant du texte prédéfini ? Cette méthode, appelée « insertion de cookie », consiste à insérer l'identifiant du serveur dans un cookie de réponse. De cette manière, le répartiteur n'a rien à apprendre, il aura juste à lire la valeur du cookie présenté par l'utilisateur pour sélectionner le bon serveur.

Ceci permet de résoudre les deux problèmes de l'apprentissage de cookie que sont : la limitation de mémoire et le basculement entre répartiteur actif-passif. Néanmoins, cette solution nécessite un travail plus important de la part du répartiteur qui doit ainsi ouvrir le flux pour y insérer ses données. Ceci n'est pas trivial, particulièrement sur les répartiteurs matériels basés sur des ASICs qui disposent d'une connaissance limitée des protocoles TCP et HTTP. Cela nécessite également que le navigateur de l'utilisateur accepte plusieurs cookies.

Ce n'est pas toujours le cas sur les terminaux mobiles par exemple, qui sont parfois limités à un seul cookie. A partir de ce principe, il est facile de décliner quelques variations telles que la modification de cookie qui consiste à préfixer un cookie existant avec l'identifiant du serveur.

**Note**: Il est important de porter une attention particulière à la gestion du cache des réponses : les caches frontaux peuvent stocker le cookie du répartiteur et le fournir à tous les utilisateurs qui par exemple, demanderont la page d'accueil, ceci pouvant provoquer une redirection massive de tous les utilisateurs sur le même serveur.

#### LIMITES DE LA PERSISTANCE – LE SSL

Nous avons passé en revue les méthodes reposant sur l'analyse des échanges client-serveur, et parfois même leur modification.

Mais avec le nombre sans cesse croissant d'applications qui s'appuient sur le protocole SSL pour leur sécurité, le répartiteur ne sera plus systématiquement en mesure d'accéder au contenu HTTP. Pour cette raison, nous voyons apparaître de plus en plus de répartiteurs logiciels qui fournissent le support du protocole SSL.

Le principe en est simple : le répartiteur agit comme un reverse proxy et sert de terminaison SSL. Il détient les certificats des serveurs, déchiffre la requête, accède au contenu et la redirige vers les serveurs (soit en clair ou en chiffrement lèger).

Ceci soulagera alors les serveurs d'application du traitement SSL et leur apportera un gain de performance.

Quoi qu'il en soit, le répartiteur devient alors le point de congestion : un simple répartiteur logiciel ne sera pas plus rapide pour traiter le SSL qu'une ferme de huit serveurs. A cause de cette erreur d'architecture, le répartiteur arrivera à saturation avant les serveurs d'application, et le seul remède à cela sera d'ajouter un autre étage de répartiteurs en frontal et ainsi de suite jusqu'à traiter toute la charge SSL.

Manifestement, cette méthode n'est pas bonne. Maintenir une ferme de répartiteurs est difficile et les « health checks » réalisés par tous ces systèmes risquent de provoquer une charge notable sur les serveurs.

**La bonne solution consiste donc à disposer d'une ferme dédiée à la gestion du SSL.**

### UNE FERME SSL DEDIEE

Lorsque les applications nécessitent l'emploi du SSL, la solution la plus évolutive consiste à dédier une ferme de reverse proxies SSL. Cette solution ne présente que des avantages :

1. Les reverse proxies SSL sont économiques mais puissants. Tout le monde est en mesure de configurer un tel système sur base Apache pour moins de 1000€. Ce coût doit être comparé à celui d'un répartiteur embarquant la gestion de SSL (plus de 10000€), mais aussi à celui d'un serveur multi-processeur utilisé pour l'application, dont la puissance ne sera plus gaspillée par le traitement SSL.
2. Les reverse proxies SSL fournissent presque toujours la fonction de cache, et parfois même, de la compression. La plupart des applications de e-business présentent un taux de cache de l'ordre de 80 à 90%, ce qui signifie que les reverse proxy caches déchargeront les serveurs d'au moins 80% des requêtes.
3. Ajoutant à cela le fait que les proxies SSL peuvent être partagés entre plusieurs applications, le gain global réduit ainsi le besoin de nombreux et puissants serveurs d'applications, ce qui aura un impact positif sur les coûts de maintenance et de licences.
4. Une ferme SSL peut grossir en fonction de l'augmentation du trafic, sans nécessiter d'évolution ou de remplacement des répartiteurs. Si l'architecture est bien pensée, il faut savoir que les capacités des répartiteurs sont généralement largement au dessus des besoins de la grande majorité des sites.
5. Le premier niveau de reverse proxies fournit un très bon niveau de sécurité en permettant le filtrage des requêtes invalides et bien souvent, en autorisant le filtrage d'URL pour les attaques les plus connues.
6. Il est généralement facile de remplacer un composant d'une ferme SSL lorsque des besoins spécifiques se font jour (ex: authentification forte). A l'opposé, remplacer un répartiteur qui gère également le SSL peut fortement impacter le comportement de l'application en raison des différences de méthodes de « health check », d'algorithmes de répartition ou des moyens de persistance.
7. Généralement, le choix d'un répartiteur se fera sur la base de ses performances de répartition plutôt que sur ses performances de traitement SSL. Un simple répartiteur sera toujours mieux adpaté et moins couteux qu'une solution « tout-en-un ».

Une ferme de proxies-cache-SSL est composée d'un ensemble de serveurs identiques, s'appuyant presque toujours sur le couple logiciel Apache+modSSL (même dans de nombreuses solutions commerciales), dont la charge est répartie par un répartiteur réseau externe, ou bien par un logiciel embarqué tel que LVS sous Linux.

Ces serveurs ne nécessitent que très peu de maintenance. Leurs seules tâches consistent à transformer le trafic HTTPS en HTTP, vérifier le cache, et transmettre les requêtes non mémorisées vers la ferme de serveurs d'applications, constituée du répartiteur et des serveurs d'applications. En termes d'architecture, il est d'ailleurs intéressant de noter que le même répartiteur de charge peut être partagé entre les reverse proxies SSL et les serveurs d'applications.

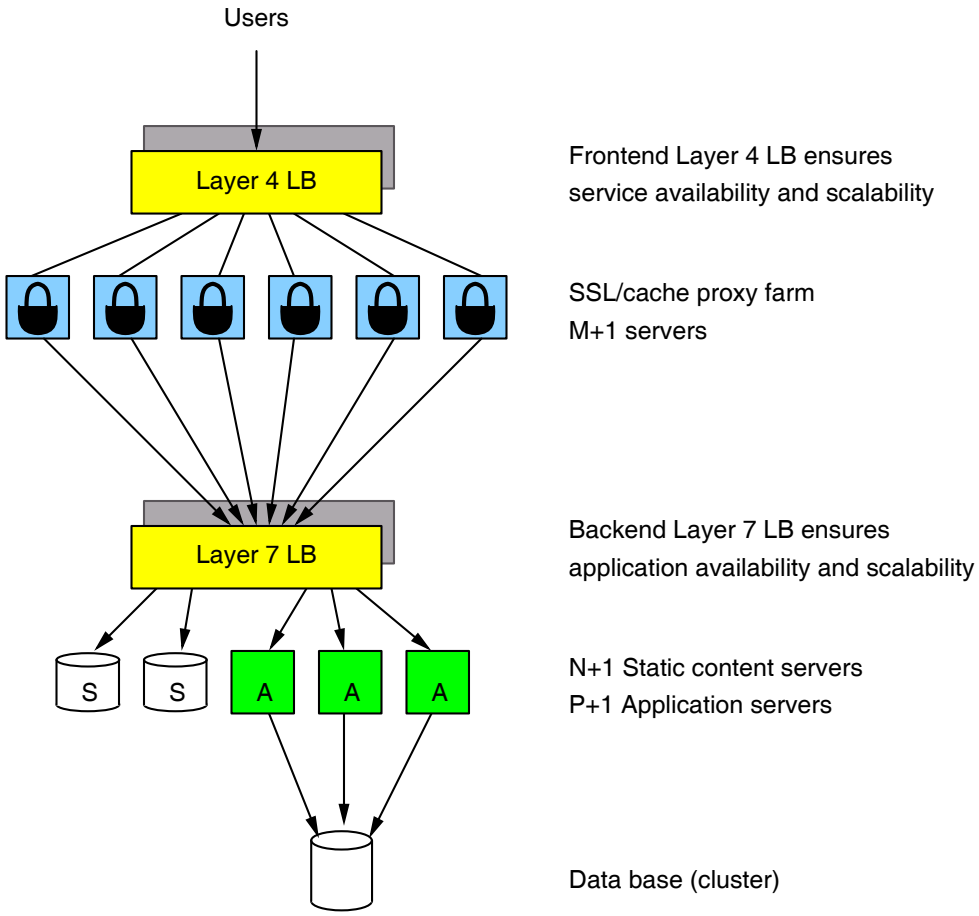


**Fig.2** : ferme SSL devant le répartiteur de niveau 7

## Comment choisir entre un répartiteur matériel ou logiciel ?

Bien que toutes les solutions semblent pouvoir répondre à tous les besoins, ce n'est pas le cas. Il y a principalement 3 aspects importants qui doivent permettre le choix d'un répartiteur :

- performance
- fiabilité
- fonctionnalités

La performance est le point critique, car lorsque le répartiteur devient le point de congestion, il devient impossible d'y remédier. La fiabilité est également très importante parce que le répartiteur supporte l'ensemble du trafic, sa fiabilité doit donc être bien supérieure à celle des serveurs, aussi bien en termes de disponibilité qu'en terme de qualité de traitement. Les fonctionnalités proposés sont déterminantes pour choisir une solution mais ne sont pas critiques. Le choix dépendra principalement des compromis acceptables selon les modifications qui seront nécessaires sur les serveurs d'application.

**NOMBRE DE SESSIONS**

L'une des questions les plus fréquentes lorsque l'on compare des répartiteurs matériels ou logiciels est du pourquoi existe-t-il une telle différence entre leur nombre de sessions supportées. Les proxies logiciels annoncent généralement quelques milliers de sessions concurrentes là ou les répartiteurs matériels parlent de millions.

De toute évidence, très peu de personnes disposent de 20000 serveurs Apache pour gérer 4 millions de sessions simultanées !

En fait, cela va dépendre si le répartiteur doit ou non gérer le protocole TCP. Quand une session TCP se termine, elle reste dans la table de sessions en état TIME_WAIT suffisamment longtemps pour pouvoir détecter des retransmissions tardives, qui peuvent arriver plusieurs minutes après que la fin de la session. Au-delà de ce délai, la session est automatiquement supprimée.

En pratique, en fonction des configurations, des délais variant entre 15 et 60 secondes sont usuels. Ceci pose un réel problème sur les systèmes qui supportent un fort taux de renouvellement de sessions parce que toutes les sessions terminées doivent être conservées dans la table. Un délai d'expiration de 60 secondes peut générer pas moins de 1.5 millions d'entrées pour un taux de 25000 sessions par seconde.

Heureusement, les sessions stockées dans cette état ne portent aucune donnée et sont donc très peu coûteuses. Comme elles sont gérées de manière transparente par le système d'exploitation, le répartiteur logiciel ne les voit jamais et annonce donc seulement le nombre de sessions actives qu'il peut supporter. Mais quand le répartiteur doit gérer le protocole TCP, il doit être capable de stocker toutes ces sessions et annonce donc une limite globale beaucoup plus élevée.

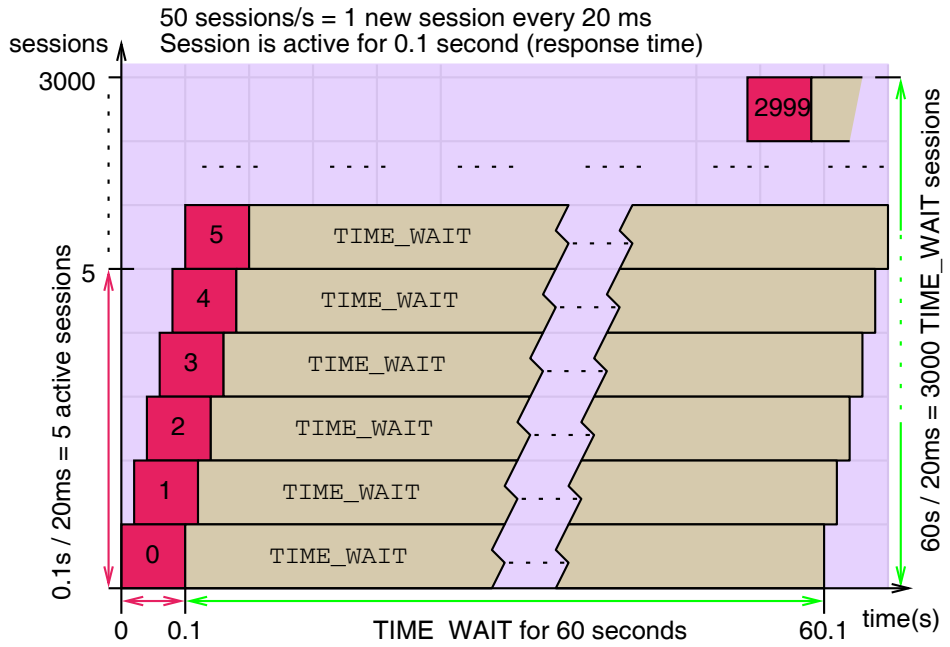


**Fig.3** : remplissage de la table des sessions selon le trafic

**REPARTITION DE CHARGE DE NIVEAU 3/4**

Naturellement, la meilleure performance peut être atteinte en ajoutant la plus faible surcharge, ce qui signifie traiter les paquets au niveau réseau. Pour cette raison, les répartiteurs matériels peuvent atteindre de très hauts débits quand il s'agit de répartir du trafic au niveau 3 (ex : routage direct avec un algorithme de hash). Comme le niveau 3 ne suffit généralement pas, un répartiteur réseau doit également s'occuper du niveau 4. La majorité de la charge sera générée par la gestion des sessions qui pourra néanmoins être réalisée à fort débit compte tenu du fait que cela reste une opération peu coûteuse en ressources.

Des opérations plus complexes telles que la translation d'adresses demandent plus de puissance de traitement pour recalculer les « checksums » et effectuer des recherches dans les tables. Nous voyons partout des solutions d'accélération matérielle qui permettent de forts gains de performance, et dans ce domaine, la répartition de charge ne fait pas exception.

Un ASIC dédié peut router des paquets à la vitesse du fil, gérer des sessions à très haute vitesse en s'affranchissant de la surcharge qu'impose généralement un système d'exploitation générique.

Réciproquement, l'utilisation d'un proxy logiciel pour le traitement de niveau 4 limite fortement les performances par le simple fait que pour réaliser de simples tâches au niveau des paquets, le système doit les décoder, décortiquer les niveaux pour trouver les données, allouer de l'espace mémoire pour les gérer, les transférer au processus, établir les connexions vers les serveurs distants, gérer les ressources systèmes, … Pour cette raison, un répartiteur proxy de niveau 4 sera généralement 5 à 10 fois plus lent que son équivalent réseau sur le même matériel.

La mémoire est également une ressource limitante : le répartiteur de niveau réseau requiert de la mémoire pour stocker les paquets à la volée ainsi que les sessions (composées des adresses, des protocoles, des ports et de quelques autres paramètres qui globalement n'occupent que quelques centaines d'octets par session). Le répartiteur proxy quant à lui à besoin de tampons mémoire pour communiquer avec le système. Les tampons mémoire par session sont mesurés en kilo octets, ce qui implique des limites pratiques aux alentours des quelques dizaines de milliers de sessions actives.

Les répartiteurs réseau permettent également d'utiliser de nombreuses fantaisies utiles telles que la gestion des adresses MAC virtuelles, le détournement d'adresses de machines ou de réseaux, … qui ne sont pas possibles en standard avec les proxies qui s'appuient sur un système d'exploitation classique.

Une note complémentaire sur la fiabilité. Les répartiteurs de niveau réseau peuvent également parfois agir « salement » en transmettant des paquets invalides tels quels, ou en mélangeant les sessions (particulièrement en mode NAT). Nous avons déjà évoqué le phénomène de « ACK storms » qui surviennent parfois en mode NAT avec des délais d'expiration de session trop faibles. Ils sont provoqués par la réutilisation trop précoce de sessions récemment terminées, et causent des saturations du réseau entre un utilisateur donné et un serveur. Ceci ne peut pas se produire avec les proxies car ils reposent sur des piles TCP totalement standards et qu'ils coupent complètement la session entre le client et le serveur.

Dans l'ensemble, un répartiteur de niveau réseau bien configuré sera généralement la meilleure des solutions de niveau 3/4 en termes de performances et de fonctionnalités.

### REPARTITION DE CHARGE DE NIVEAU 7

La répartition de charge de niveau 7 implique la persistance basée sur les cookies, la transcription d'URL et quelques fonctionnalités complémentaires fort utiles. Tandis qu'un proxy qui s'appuie sur une pile TCP/IP classique peut facilement s'acquitter de sa tâche, les répartiteurs de niveau réseau doivent faire appel à de nombreuses astuces qui ne s'accommodent pas toujours bien avec les standards et causent souvent de nombreux problèmes.

Les problèmes les plus complexes à résoudre sont :

1. **les entêtes multi-paquets :** le répartiteur recherche des chaînes de caractère dans les paquets**.** Lorsque les données attendues ne se trouvent pas dans le premier paquet, celles-ci doivent être mémorisées et consomment de la mémoire. Quand la chaîne de caractères débute à la fin d'un paquet et se poursuit dans le suivant, elle devient difficile à extraire. Ceci est pourtant le mode de fonctionnement standard des flux TCP. Pour référence, une requête d'une taille de 8 Ko telle que supportée par un serveur Apache se répartira communément sur 6 paquets.

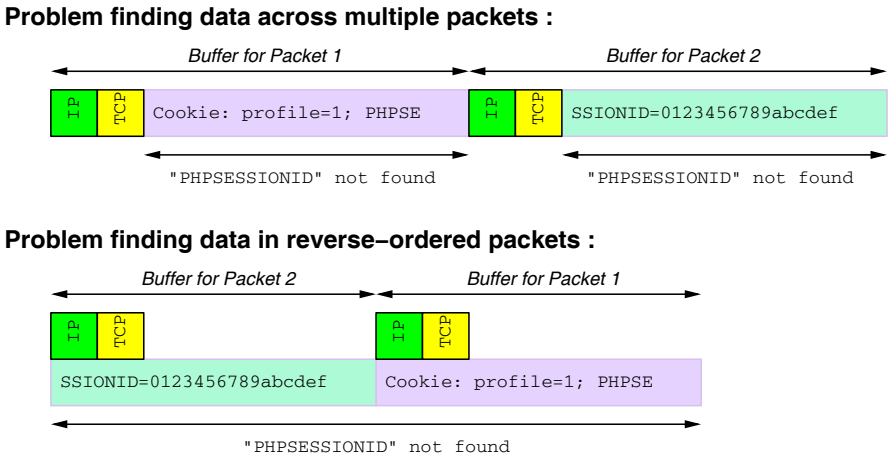


**Fig.4** : Une requête HTTP peut s'étendre sur plusieurs paquets

2. **les paquets en ordre inversé** : lorsqu'un gros et un petit paquet sont envoyés sur Internet, il est très fréquent que le plus petit atteigne sa destination avant le gros. Le répartiteur doit alors gérer cela avec précaution.

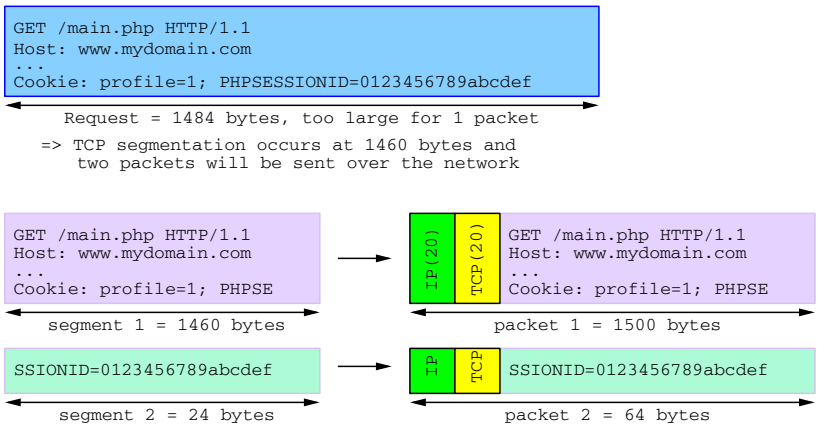


**Fig.5** : La difficulté de gérer HTTP avec certains répartiteurs matériels

3. **la fragmentation** : Quand un paquet est trop gros pour être routé sur un media, un routeur intermédiaire est autorisé à le fragmenter. Ceci devient rare sur Internet, mais c'est de plus en plus fréquent sur les réseaux internes du fait des VPN qui limitent la taille de la charge utile des paquets. Les fragments sont difficiles à traiter tenant au fait qu'ils arrivent dans le désordre, doivent donc être mémorisés pour reconstruire le paquet alors que seul le premier paquet contient les informations de session. Les répartiteurs de niveau réseau ne gèrent que difficilement les fragments qu'ils qualifient parfois d' « attaques » comme une excuse au fait qu'ils ne peuvent pas les traiter.
4. **La perte de paquets et retransmissions** : A travers Internet, il y a une importante perte de paquets entre un utilisateur et un serveur. Ils sont généralement automatiquement retransmis après un court délai, mais le processus de traitement des paquets doit alors être recommencé. Le répartiteur ne doit donc pas considérer que ce qui a été réalisé ne devra pas l'être à nouveau (typiquement quant la fin d'un entête est atteinte).
5. **La différenciation des entêtes et des données** : En HTTP, l'information de protocole, appelée « entête » se situe au début des échanges, suivi par les données qui débutent à la suite de la première ligne vide. La lecture approximative des paquets dans les répartiteurs de niveau réseau fait parfois correspondre des cookies ou bien modifie des sections de données ce qui engendre une corruption des informations. Un problème similaire apparaît parfois à la suite d'une fin de session : si l'utilisateur réutilise trop rapidement le même port, le répartiteur peut l'assimiler à des données et n'effectuera pas le traitement de niveau 7.
6. **Insertion de données** : quand une donnée (ex: un cookie) est inséré par le répartiteur, les numéros de séquence TCP tout comme les "checksums" doivent être recalculés pour tous les paquets qui passent, générant un impact notable sur les performances voire des comportements erronés.

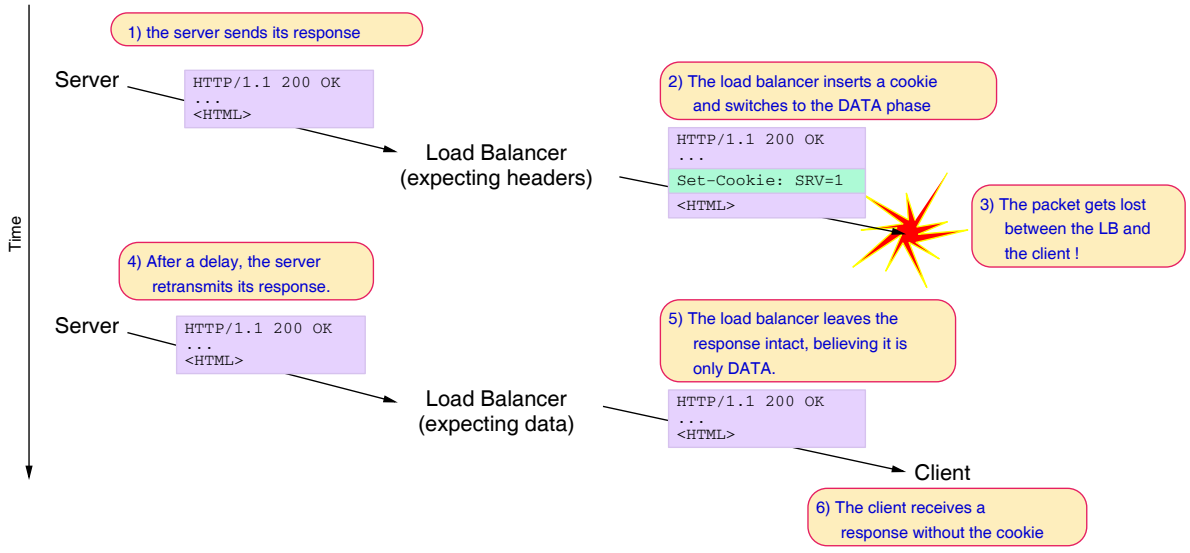


**Fig.6** : Problèmes fréquents dans les piles TCP simplifiées

Comme on le voit, il est très difficile de traiter le niveau 7 au niveau des paquets. Le contenu sera parfois mal identifié, les requêtes transmises au mauvais serveur, et parfois, un réponse ne sera pas traitée comme il se doit avant d'être transmise à l'utilisateur. En fait, les répartiteurs les plus complets s'appuient souvent sur des proxies embarqués pour traiter les tâches les plus complexes. Certains répartiteurs de niveau réseau transfèrent les traitements de niveau 7 à leur processeur de « management » qui est souvent peu puissant, et d'autres, possèdent des processeurs dédiés mais qui ne sont généralement pas aussi puissants que ce ceux que l'on trouve dans les serveurs matériels les plus récents. De plus, ils sont généralement touchés par le problème de la consommation mémoire due à la gestion des tampons mémoire.

Généralement, ces produits n'annoncent pas le nombre de sessions qu'ils sont capables de gérer au niveau 7, ou se contentent d'affirmer que cette gestion n'a aucun impact sur les performances ce qui est bien entendu totalement faux comme le démontre ce qui suit : pour se situer au même niveau de performance qu'un serveur Web basé sur Apache, un répartiteur de charge devrait supporter 8Ko par requête, ce qui signifie qu'il devrait conserver en mémoire jusqu'à 8Ko de données par session établie à tout instant. Ainsi, pour supporter 4 millions de sessions simultanées de 8Ko chacune, ceci nécessiterait **32**Go de mémoire, ce dont bien entendu, ils ne disposent jamais.

Avant de choisir un répartiteur, des tests doivent être réalisés par le client afin d'en observer le comportement face aux situations les moins triviales, car des dénis de service faciles à mettre en oeuvre sont monnaie courante.

Au contraire, le répartiteur de charge de niveau 7 ne supporte pratiquement aucune surcharge due au traitement de niveau 7 principalement du fait que les problèmes évoqués sont naturellement résolus par le système d'exploitation sous jacent. Le répartiteur n'a donc plus qu'à se concentrer sur le contenu et sur ses tâches. Très souvent, ces répartiteurs seront en mesure de fournir des logs très détaillés sans surcoût de ressources, ce qui non seulement permettra de décharger les serveurs de cette tâche mais fournira également de précieuses informations pour optimiser la gestion de la capacité.

Enfin, le niveau 7 évolue rapidement, principalement du fait de nouvelles méthodes de persistance ou d'analyse de contenu requises par les évolutions des applications. Alors que les mises à jour logicielles sont très faciles à fournir pour les éditeurs de logiciels, et faciles à installer par les clients, la mise à jour des images de firmware pour les répartiteurs matériels requièrent un plus haut niveau de validation de la part des constructeurs, ne leur permettant pas d'offrir un même niveau de réactivité.

#### LA MEILLEURE COMBINAISON, POUR CEUX QUI PEUVENT SE L'OFFRIR

La meilleure combinaison est donc d'utiliser un premier niveau de répartiteurs réseau (éventuellement avec accélération matérielle) pour assurer la répartition de niveau 4 entre les reverse proxies SSL et les répartiteurs de charge de niveau 7. Tout d'abord, les tests réalisés par les répartiteurs réseau sur les répartiteurs de niveau 7 seront toujours plus fiables que la connaissance qu'un proxy a de son propre état. Ensuite, cette architecture fournit la meilleure évolutivité générale pour la simple raison que lorsque les reverse proxies satureront, il sera toujours possible et facile d'en rajouter jusqu'à atteindre la limite de saturation des répartiteurs de niveau 4. A ce stade, nous parlons déjà de saturer des réseaux multi-gigabits. Enfin, il sera toujours temps de mettre en place la méthode du DNS cyclique (round roubin) pour adresser de multiples répartiteurs de niveau 4, de préférence répartis sur plusieurs sites.

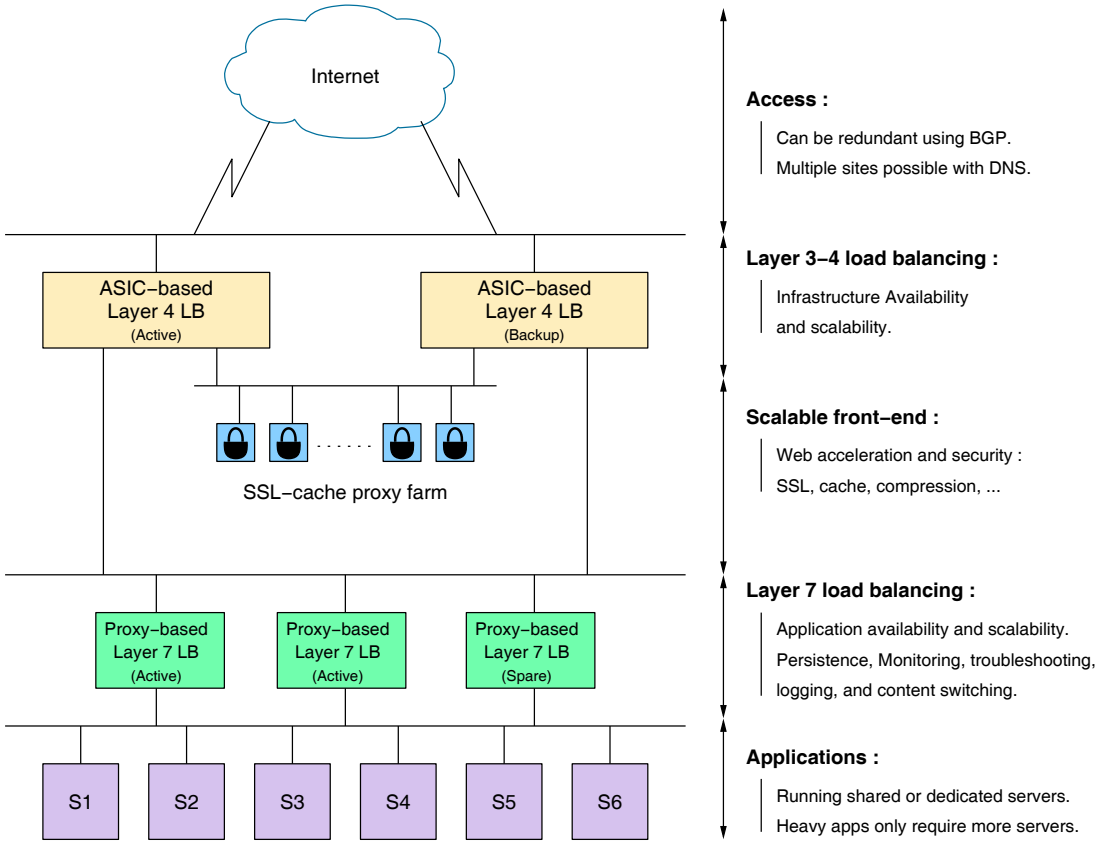


**Fig.7** : Exemple type d'un architecture évolutive

# Optimisation applicative indépendamment du répartiteur

Mettre en place un répartiteur est très bien pour améliorer l'évolutivité, mais ne doit pas permettre de s'affranchir de l'optimisation des applications ou des serveurs.

#### SEPARER LES CONTENUS STATIQUES ET DYNAMIQUES

La recette est connue mais rarement mise en oeuvre. Dans de nombreuses applications, environ 25% des requêtes concernent des objets dynamiques, les 75% restants concernant du contenu statique. Chaque processus Apache+PHP sur le serveur d'application consommera de 15 à 50 Mo de mémoire en fonction de l'application. Il est donc totalement anormal de dédier un tel monstre pour transférer une petite icône vers un utilisateur au travers d'Internet, tenant compte également de toute la latence et des pertes de paquets qui sont engendrées. Et c'est même pire lorsque un processus entier est monopolisé pendant plusieurs minutes parce que l'utilisateur télécharge un fichier volumineux tel qu'un PDF !

La solution la plus simple consiste donc à s'appuyer sur un cache reverse proxy en frontal de la ferme de serveurs qui renverra directement à l'utilisateur le contenu mis en cache sans avoir besoin d'interroger les serveurs d'application. La solution la plus propre consiste à dédier à cette tâche un serveur HTTP léger. Il peut être installé sur les mêmes serveurs et utiliser par exemple un autre port TCP. De préférence, on utilisera un serveur mono-thread tel que Lighttpd ou Thttpd dont la charge par session reste très faible. L'application devra juste être capable de classer les contenus statiques dans un répertoire tel que « /static » de façon à ce que le répartiteur frontal puisse diriger le trafic vers le serveur dédié. Il est également possible d'utiliser des noms de serveurs totalement différents pour les serveurs statiques, ce qui permettra éventuellement de les installer sur d'autres sites, parfois plus proches des utilisateurs.

#### CE QUI PEUT ETRE OPTIMISE COTE SERVEUR - EXEMPLE AVEC APACHE

Il y a souvent quelques astuces simples qui peuvent être appliquées sur les serveurs et qui permettront d'augmenter drastiquement leur capacité. Avec l'astuce décrite ci-dessous, il est très fréquent de pouvoir augmenter par un facteur de deux à trois le nombre d'utilisateurs simultanés sur un serveur Apache+PHP sans devoir toucher au matériel.

Premièrement, il convient de désactiver le "keep alive". C'est la pire des choses qui puisse impacter les performances. Il a été conçu à un époque ou les sites utilisaient le serveur NCSA httpd qui créer un processus (fork) à chaque nouvelle requête. Tous ces processus finissaient par tuer le serveur et « keep alive » était un bon paliatif. De nos jours, les choses ont changé. Les serveurs ne créent plus de nouveau processus à chaque connexion et leur coût en ressource est minimal. Les serveurs d'application gèrent un nombre limité de threads ou de processus, souvent à cause de contraintes mémoire, de limites liées aux descripteurs de fichiers, ou à la gestion des verrous. De ce fait, avoir un utilisateur qui monopolise un thread pendants des secondes ou même des minutes à ne rien faire est une pure perte.

Le serveur n'utilisera pas toute sa puissance processeur, il consommera un volume mémoire effarant et les utilisateurs attendront qu'une connexion se libère. De plus, si le délai de « keep alive » est trop court pour maintenir une session, il est alors totalement inutile. Si il est assez long, cela signifie alors que le serveur aura besoin d'environ un processus par utilisateur simultané, sans compter le fait que la plupart des navigateurs établissent 4 sessions simultanées !

Pour être clair, un site qui utilise "keep alive" avec un serveur de type Apache n'a aucune chance de pouvoir desservir plus de quelques centaines d'utilisateurs simultanés.

Deuxièmement, il convient d'observer la consommation mémoire moyenne par processus. Il faut ensuite jouer avec le paramètre « MaxClient » pour ajuster le nombre maximum de processus simultanés afin que le serveur ne « swappe » pas.

Si il existe de très nettes différences entre les processus, cela signifie que certaines requêtes produisent des résultats lourds, qui sont une pure perte lorsqu'ils sont conservés en mémoire.

Pour résoudre ce problème, vous devrez indiquer au serveur Apache de tuer plus rapidement ses processus en jouant avec le paramètre 'MaxRequestsPerChild'. Plus la valeur est haute, plus la consommation mémoire est importante. Plus elle est basse, plus la consommation du processeur sera importante. Généralement, des valeurs comprises entre 30 et 300 donnent les meilleurs résultats. Ensuite, ajuster la valeur des paramètres 'MinSpareServers' et 'MaxSpareServers' proches de la valeur du paramètre 'MaxClient' de façon à ce que le serveur ne perde pas trop de temps à créer de nouveaux processus quand la charge survient.

Avec ces astuces, un serveur récent avec 2Go de RAM n'aura aucun problème à gérer plusieurs centaines d'utilisateurs. Le reste est typiquement le travail du répartiteur de charge.

## Plus de lecture sur le sujet de la répartition de charge

Deux analyses concurrentes mais néanmoins très intéressantes sur le sujet (en anglais):

http://www.zeus.com/news/pdf/white_papers/7_myths.pdf

http://www.foundrynet.com/pdf/wp-server-load-bal-web-enterprise.pdf