Apache/PHP et IIS/ASP méritent-ils la place qu'ils occupent ?

vendredi 28 juin 2002, **par aegir**

Entre Microsoft Internet Information Server, et Apache avec son module PHP, la guerre fait rage. En terme de parts de marché, Apache l'emporte haut la main avec plus de 60%, tandis que MS-IIS se contente des miettes avec un peu plus de 20% de part de marché. Pourtant, aucune des deux solutions n'est entièrement convaincante.

Besoins et contraintes techniques

Pour commencer rappelons quelques aspects techniques. Un serveur WEB « basique » est quelque chose de très simple qui peut s'écrire en quelques lignes de C++ ou autres langages. Le serveur WEB ne fait que « écouter » les connexions qui arrivent sur le port 80 (en général), et lorsqu'une requête du type « GET http://www.monsite.com/mapage.html » arrive, le serveur va chercher le fichier « mapage.html » et en envoie le contenu sur cette même connexion, puis ferme la connexion.

Bon, la réalité est un peu plus complexe dans la mesure où le protocole HTTP ne se limite pas à la requête « GET », mais fondamentalement voici ce qu'étaient à l'origine les serveurs WEB.

Il ne vous échappe donc pas qu'un serveur WEB basique tel que je viens de le décrire ne sait faire qu'une seule chose : transmettre le contenu d'un fichier. C'est pauvre, pas très utile, mais cela correspondait très exactement au besoin qu'il y avait en 1991 pour exploiter le HTML et ses hyper liens, qui avaient pour but de permettre de consulter simplement de la documentation.

Seulement, en 1994, l'Internet explose, s'ouvre au grand public, des sites apparaissent un peu partout, mais ces sites gardaient le concept « documentation à consulter ». Bientôt certains voulurent proposer des sites interactifs aux visiteurs, et là les choses devinrent plus ardues puisque le WEB avec son protocole HTTP et son « langage » HTML n'était pas du tout fait pour cela. Alors il a d'abord été imaginé un artifice : le « CGI ».

Le CGI correspond à une commande qui est exécutée, et dont le résultat correspond à un morceau de la page WEB. Un exemple typique est le compteur de visiteurs. Le fichier HTML qui est stocké sur le disque dur du serveur va contenir quelque chose du genre (la syntaxe n'est qu'un simple exemple) :

Nombre d'internautes ayant visité cette page : <!--CGI moncompteur.exe -->

Ainsi, les serveurs WEB consultaient le contenu du fichier HTML avant d'en envoyer le contenu sur la connexion TCP/IP d'où vient la requête GET. Lorsque le serveur rencontre la balise spéciale < !-- xxxxxxx -->, il va la remplacer par le résultat d'une opération. Dans notre exemple, l'opération consiste à exécuter le programme monprogramme.exe. Si l'exécution de ce programme affiche la valeur « 10 », alors le contenu du fichier transmis sera :

Nombre d'internautes ayant visité cette page : 10

C'était déjà mieux que de simples fichiers html, mais c'était vraiment primitif, et passablement inadapté. D'abord parce que si la page comporte plusieurs appels CGI, il s'agit d'autant de programmes exécutés séparément. Puisque l'exécution est séparée, les différents programmes ne peuvent pas communiquer entre eux (ou du moins pas simplement). Ensuite, parce qu'en terme de performances, ce système est désastreux. Si la page comporte quatre appels de CGI, le serveur devra exécuter quatre programmes différents à chaque fois qu'un internaute veut consulter cette page ! Et l'exécution d'un programme, même très simple, est toujours une opération coûteuse sur un système : il faut initialiser un contexte d'exécution, réserver une zone mémoire, charger dans cette zone le programme, l'exécuter, et lorsque le programme est terminé faire l'opération inverse... Enfin, l'interactivité sur le WEB se heurtait à un dernier problème que le CGI n'aidait pas à résoudre : le contexte de connexion. En effet, comme je l'ai décrit plus haut, lorsqu'un fichier HTML est demandé, le serveur ferme la connexion après en avoir envoyé le contenu. C'est le protocole HTTP qui est fait comme cela. C'est tout à fait adapté à la consultation de documentation mais certainement pas à l'interactivité. En effet, si la connexion reste ouverte le serveur peut mémoriser les opérations déjà effectuées sur cette connexion. Représentez-vous la connexion comme un tuyau, vous pouvez vous souvenir de ce qui est déjà arrivé par ce tuyau, et ce que vous y avez envoyé. La caractéristique d'un tuyau étant d'avoir deux extrémités fixes, vous savez que deux informations qui vous parviennent successivement par un même tuyau ont la même provenance. Si la connexion est fermée systématiquement, les deux informations envoyées par la même personne vous parviendront par deux tuyaux différents, comment faire le lien entre ces deux informations ? C'est impossible avec le protocole HTTP, il faut chercher des astuces ailleurs, c'est pourquoi dans le jargon on dit que le HTTP est un protocole « stateless » (sans contexte).

Alors une idée a germé : puisqu'on demandait déjà au serveur web de consulter le contenu des fichiers HTML pour remplacer les balises d'appels aux CGI par le résultat de l'exécution d'un programme externe, pourquoi ne pas permettre de mettre directement dans ces balises des instructions écrites dans un langage spécifique qui serait interprété par le serveur ?

Le « scripting »

C'est ainsi qu'ont été conçues les pages PHP ou ASP (ou ColdFusion etc.). C'est ce qu'on appelle des langages de scripting. Concrètement le contenu d'une page va désormais être de la forme :

```
<body>

<P>La page a été appelée avec le paramètre :
<? printf $PARAMETRE ?>
<P>
<? printf $PARAMETRE ?>
multiplié par deux égale
<? printf $PARAMETRE*2 ?>
</body>
```

Si vous appelez mapage.php ?PARAMETRE=5, votre navigateur va afficher :

La page a été appelée avec le paramètre : 5

mutiplié par deux égale 10

Cela permet d'avoir plusieurs parties de la page générées dynamiquement par le serveur, et pouvant partager les mêmes variables ce qui n'était pas le cas des CGI. Cela permet aussi avec plus ou moins de bonheur de résoudre le problème de contexte de la connexion, par exemple en ajoutant systématiquement au bout de chaque URL un paramètre ID=xxxxx.

Les langages utilisés dans les pages PHP ou ASP sont très complets, et assez semblables. Il existe d'ailleurs même des utilitaires permettant de convertir des pages ASP en PHP. On peut dans le script d'une page consulter une base de données, la mettre à jour, générer des images, bref tout ce dont on peut rêver. Un site comme LinuxFrench est par exemple entièrement écrit en PHP. Le bricolage du scripting est certes très astucieux, mais cela ne reste qu'un vulgaire bricolage, et je n'aime pas beaucoup ces solutions pour mener un projet d'envergure. Voici pourquoi :

Défauts inhérents au scripting

Une première contrainte, c'est que le code source des pages du site doit se trouver sur le serveur de production. En effet le serveur interprète lui-même ce code source. Ce n'est pas que ça m'ennuie de publier du code source (sur linuxfrench ce serait un comble !), mais cela peut poser des problèmes. Par exemple le code source peut contenir des informations qu'on ne souhaite pas rendre publiques auprès des administrateurs du site WEB. Ainsi, si j'ai en toutes lettres dans un fichier DBCONNECT("dbproduction","utilweb","tarzan"), cela va prendre quelques secondes à l'administrateur du site WEB (qui a accès au code source des pages WEB) d'avoir également l'accès à la base de données de production. Bien sûr, dans une petite PME où l'administrateur du site WEB est aussi l'administrateur des bases de données, le mainteneur du parc informatique, le responsable de la formation utilisateurs, celui qui sait changer le toner de la photocopieuse etc., cela ne pose pas de problème. Mais dans une grosse structure (bancaire par exemple ou bien des sociétés du genre Alcatel Thomson etc.), il est rare que ceux qui administrent les serveurs WEB (il y a d'ailleurs au moins douze administrateurs, deux stagiaires et trois prestataires de services) aient également accès aux bases de données (ils sont au moins autant à administrer les bases de données). D'une manière générale dans ces grosses structures, celui qui administre un outil n'administre surtout pas l'information circulant par cet outil (ce serait là créer une zone d'incertitude dangereuse pour l'organigramme hiérarchique).

Le scripting a également l'inconvénient d'être interprété. Ça peut paraître bizarre au XXIème siècle de rappeler cette évidence, mais l'interprétation est moins performante (en terme de temps d'exécution) que l'exécution d'un binaire compilé. Avec la compilation l'analyse lexicographique, l'analyse syntaxique, et la résolution des symboles sont déjà effectuées. (C'est d'ailleurs pour cela que ceux qui prétendent que le JAVA c'est « de l'interprété » n'ont pas compris grand chose à l'informatique, et encore moins à l'algèbre, mais c'est là de la petite histoire.)

Continuons notre démarche masochiste de démolition ;) Les langages PHP et autres VBScripts utilisés dans les ASP, ont un défaut fondamental : ils ne sont pas « typés ». Un langage « fortement typé » est un langage « déclaratif » dans lequel vous indiquez que le symbole "age" contient une valeur numérique entière. Vous pouvez donc écrire dans votre programme "age<=22", mais en aucun cas "age<=22,5" ou bien "age<='douze'". Ça c'est le « typage ». L'aspect déclaratif c'est que vous avez le droit d'utiliser le symbole "age" dans votre programme parce que vous avez au préalable déclaré que vous alliez utiliser ce symbole pour une valeur numérique entière. C'est contraignant pour le programmeur (il doit déclarer au préalable tous les symboles qu'il va utiliser), mais cela lui évite de devenir chauve prématurément s'il fait une faute de frappe. En effet, si vous calculez "année_de_naissance <= année_courante - aeg" (notez la faute de frappe "aeg" au lieu de "age") dans un langage déclaratif fortement typé, le programmeur verra immédiatement s'afficher une erreur du genre « Symbole 'aeg' non défini, ou bien erreur de type 'aeg' / 'année_de_naissance' ». Il n'y aura même pas eu besoin de tester le programme pour déceler l'erreur.

Dans un langage non typé, mais déclaratif (comme le C) le programmeur aurait eu également une erreur « symbole 'aeg' non défini ». C'est déjà pas mal. Mais si par malheur le programmeur avait aussi déclaré le symbole 'aeg' comme étant une chaîne de caractère contenant « aegir@linuxfrench.net » aucune erreur ne serait signalée et vous auriez un PDG d'une société qui se gratterait la tête en se demandant comment il se fait qu'il a dû payer son service informatique pendant deux semaines uniquement pour résoudre le problème : « pourquoi le système informatique m'affirme que j'ai fondé l'entreprise à l'ère des dinosaures ? » (oui, je le sais, il y a parfois un peu de malice dans mes exemples). Il y a des langages « plus ou moins fortement typés », Par exemple ajouter l'entier "1 " au décimal "1,5" donnera "2,5" en PASCAL, mais génèrera une erreur lors de la compilation en ADA avec un message qui dira en substance "éh mon gars, dis-moi clairement si tu veux additionner des entiers ou des décimaux". Ça peut paraître fasciste comme attitude, mais quand on sait qu'un ordinateur est incapable de se représenter la valeur réelle de "2,1", cela évite des mois d'incantations ésotériques dans le but de comprendre pourquoi ce fichu ordinateur déclare que 2,1 + 1,0 n'est pas la même chose que 2 + 1 + 0,1 pas plus que la même chose que 2,0 + 1 + 0,1. (et c'est là une réalité vécue !).

Une fois encore, je trouve étrange d'écrire cela au 21ème siècle alors que je croyais que ce débat avait été largement exploré dans le milieu des années 70 par les génies que sont Wirth, Dijkstra ou Kleene. Mais j'ai l'impression qu'il faut le rappeler.

Les données, leur traitement, et leur représentation

Continuons avec un aspect plus visuel. Un problème qui se pose en informatique depuis une dizaine d'années, est de séparer trois choses :

- Les données ;
- le traitement des données (aussi appelées règles de gestion) ;
- La représentation des données.

Le problème est le suivant : vous êtes informaticien, mais j'ai aussi embauché un graphiste webdesigner. Le genre de gars qui est capable de vous transformer un simple briquet en véritable oeuvre d'art. Comment peut-on faire pour que vous, informaticien, vous conceviez le site et que l'artiste en fasse le design ? Lui il ne touchera pas au code source, d'ailleurs œ n'est pas son boulot, c'est un graphiste. De plus, comment faire pour changer le graphisme de votre site sans en toucher le code source ? Parce que si le site n'évolue pas fonctionnellement, pourquoi diable faudrait-il toucher au code source afin de mettre les pages sur fond bleu, et certains textes en caractères gras ? Avec le scripting c'est impossible, les données et leur affichage sont imbriqués dans le code source.

Enfin il y a le traitement des données, et surtout la factorisation de ces règles de gestion. Si demain vous décidez de changer les conditions générales de vente, il y a fort à parier qu'il faille en tenir compte dans plusieurs pages (au risque d'en oublier une), et de faire autant de fois le travail. Bien sûr, les jeunes débutants qui sortent de l'université vont s'écrier « c'est faux, il suffit de faire des includes !) Ils n'ont pas tout à fait tort, mais « s'il suffit » d'en faire, cela n'a rien de trivial avec ces langages là, et surtout ils oublieront de

vous dire que l'include est encore plus primitif que le CLAUSE COPY du COBOL. Enfin, c'est pas qu'ils l'oublient, mais c'est qu'ils ne le savent pas, tout comme ils ne connaissent pas les travaux du mathématicien Kleene au sujet des langages. Ne leur en voulons pas, c'est juste un problème de mode dans l'enseignement...

... et nous nous retrouvons face au défi de réinventer des solutions aux problèmes résolus depuis 25 ans. La seule nouveauté c'est que désormais un Bill Gates a le droit de breveter la solution d'il y a 25 ans.

Alors que faut-il utiliser ?

J'ai pas envie de répondre à cette question, mais maintenant que j'ai écrit cet article, il me faut bien assumer...

Le scripting, c'est pour moi bon pour un bricolage du genre de LinuxFrench. C'est valable pour un projet qui va occuper deux stagiaires l'été, sans plus. Bien sûr, pour nous qui maintenons LinuxFrench sans le moindre centime de budget, le scripting PHP est intéressant dans la mesure où il nous permet de maintenir un site en état de marche. Mais une entreprise n'a pas la contrainte de « zéro-budget ». Par là même, je veux signaler que du scripting avec Microsoft-IIS/ASP n'a pas le moindre intérêt si ce n'est que donner l'illusion à du personnel non-formé à ces technologies de maîtriser ladite technologie.

Mon choix se porterait plutôt sur les serveurs applicatifs (serveurs d'applications comme ils disent dans les mauvaises traductions), j'ai plutôt un faible pour SilverStream, mais comme il n'est (hélas) pas opensource, je ne m'étendrai pas sur ce sujet. Évoquons aussi ZOPE, qui lui est un logiciel libre, c'est pas mal du tout, mais personnellement je n'aime pas beaucoup. Enfin relativisons, si on me donne le choix entre un projet PHP et ZOPE, je n'hésiterais pas. Mais entre moi et Zope, ce n'est pas le coup de foudre, c'est personnel c'est tout. Quelques sites connus sous Zope : portalux ou dot.kde.

J'évite comme la peste les choses qui prétendent au titre de « application server » qui sont basées sur des architectures Java/EJB. C'est pas que java/EJB soit mauvais, c'est seulement que les « application serveur » auxquels je pense n'apportent pas grand chose de plus que des problèmes et des factures. Coder en L3G c'était bien il y a 15 ans, mais maintenant on veut quand même du RAD [1], et lorsqu'il faut multiplier les temps de développement par 4, finalement autant choisir du PHP ou payer une licence SilverStream.

Bref, j'attends toujours un serveur applicatif, basé sur XML, qui permette de séparer données, présentation, traitement des données, et dont le code est à la fois compilé, déclaratif et fortement typé (Java est parfait pour cela). Un L4G où l'on n'est pas obligé de coder des boucles pour afficher un tableau. Enfin je veux dire que c'est un outil qui finira bien par arriver. Espérons seulement qu'il sera libre.

[1] Rapid Application Development