Haute disponibilité et équilibrage de charge des CMS en PHP / MySQL et Zope / Plone : des architectures pour supporter des pics de charge

Philippe Daubias
Université de Lyon
ENS de Lyon / DSI
15 parvis René Descartes, BP 7000, 69342 Lyon cedex 07

Nicolas Carel
Université de Lyon
ENS de Lyon / DSI
15 parvis René Descartes, BP 7000, 69342 Lyon cedex 07

Résumé

Cet article propose un retour d'expérience sur l'utilisation d'architectures logicielle, système et matérielle permettant la répartition et l'absorption de charge pour des serveurs Web sujets à d'importants pics de charge saisonniers ou médiatiques. Plus précisément, nous présentons des architectures entièrement fondées sur des solutions logicielles libres ainsi que la méthodologie que nous avons utilisée pour les choisir. Nous montrons par des exemples concrets tirés des logs de connexion de ces sites, les comportements constatés, en tentant d'analyser précisément les pics de charge observés. Enfin, nous proposons une discussion sur les avantages et inconvénients d'une telle architecture complexe avec répartition de charge.

Mots clefs

Pic de charge, répartition de charge, HAProxy, Heartbeat, Varnish, CMS, Zope, Zeo, Plone, ZoDB, Ldap

1 Introduction

L'utilisation du Web comme vecteur privilégié de communication expose les serveurs Web potentiellement à de fortes sollicitations. De plus, le déploiement généralisé de gestionnaires de contenus (CMS), pour simplifier la mise à jour des contenus, pose la question de leur performance et de leur résistance face aux sollicitations. Les CMS, conçus pour permettre l'interaction directe ou le travail collaboratif, peuvent se révéler peu performants, que ce soit en raison d'un recours trop important à des requêtes SQL lourdes comme c'est le cas pour certains modules de Drupal [1] ou pour des questions de performance intrinsèque de l'outil, comme pour Plone. Ainsi, pour des raisons qui seront discutées dans la section 3, nous pouvons être amenés dans nos métiers à gérer des sites Web dont au moins une partie des contenus sont portés par un CMS peu performant. Si, de plus, ces sites sont susceptibles d'être fortement consultés, il faut mettre en place des infrastructures pour rendre plus robuste et plus performante la partie CMS. Selon le site, l'affluence sera plus ou moins régulière ou saisonnière, mais sera généralement prévisible et les infrastructures pourront être adaptées, y compris pour répondre à des pics de charge, tout en limitant les ressources allouées quand elles ne sont plus nécessaires.

Dans cet article, nous présentons les architectures simples que nous avons déployées pour rendre Plone robuste à des pics de consultation. Nous exposons ensuite la méthodologie de test que nous avons adoptée pour prévoir le dimensionnement des serveurs à mettre en place. Puis nous décrivons les pics de charge que nous avons observés en production. Enfin, nous discutons des avantages et inconvénients de ces architectures complexes vis-à-vis d'autres plus simples mais nécessitant en revanche des serveurs très puissants pour absorber la charge.

2 Contexte

Nous proposons ici un retour d'expérience sur deux sites sujets à des variations de charge sur des modes très différents (voir section 5). Le premier de ces sites (neo.inrp.fr) est destiné à l'ensemble des enseignants entrant en poste chaque année et à leurs formateurs. Il fournit des vidéos de situations de classe devant les aider à gérer leurs propres classes. Les vidéos sont des situations réelles dont la diffusion doit être limitée aux enseignants pour des raisons juridiques. De plus, pour des raisons administratives et techniques, il n'est possible de proposer un accès à la plateforme aux nouveaux enseignants (de l'ordre de 10 000 par an) qu'à partir de fin août, sachant que de nombreux contenus leur sont utiles pour leur rentrée. L'affluence sur le site a ainsi tendance à se concentrer sur la fin août et les quelques premiers jours de septembre. La charge est d'une part due au nombre important de visiteurs potentiels et aux contenus vidéos qui représentent une importante quantité de données à transférer. Par ailleurs, ce site est exposé médiatiquement en raison des récentes et profondes modifications des IUFM : des reportages y sont consacrés dans les médias et chaque diffusion provoque un surcroît temporaire, mais très important de la consultation du site (voir Figure 3). Le second site (interens.ens-lyon.fr) présente quant à lui les annales et les résultats des concours des ENS. Il subit une augmentation d'affluence à la date de parution des convocations pour les oraux du concours d'entrée et un pic de consultations (voir Figure 4) au moment de la promulgation des résultats.

Ces sites devant être pour partie modifiables simplement par des utilisateurs non informaticiens, une partie de leurs contenus est stockée sur le CMS Plone. Par ailleurs, les contenus des sites nécessitent une authentification des utilisateurs. Cela a dû être pris en compte dans les choix d'architectures pour éviter que des utilisateurs n'aient sans arrêt à se ré-authentifier en raison de l'utilisation d'un répartiteur de charge frontal.

3 Architectures déployées

Les deux cas que nous avons étudiés sont très différents mais partagent la même faiblesse : le CMS Plone. Plone est complexe à maîtriser, mais offre en retour un bon niveau de protection (voir le pied de première page de [5]). En effet, les attaques par injection SQL ne sont pas possibles avec Plone, car la base est celle de Zope, pas MySQL. Par ailleurs, un système en couches avec des vérifications multiples de droits rend peu utilisable une éventuelle faille sur ce CMS. Une fois sa maîtrise acquise, Plone permet de rapidement mettre en place des espaces de travail collaboratifs avec une gestion de droits répondant à la plupart des besoins (voir [2] ou [3] pour des comparatifs de CMS), mais Plone supporte mal la charge. Sur un CMS, les pages et certaines de leurs composantes doivent être générées et pas seulement servies comme c'est le cas pour un site statique sur Apache par exemple. Les pages de Plone comportent souvent une vingtaine d'éléments (entre les images de la page, les CSS, les images nécessaires pour appliquer la CSS) et Plone se révèle lent dans la construction d'une page et son envoi au client.

Cependant, principalement pour sa robustesse et pour la maîtrise que nous avions de ce CMS, il a été choisi pour les contenus à modifier par les webmasters. Par ailleurs, nous utilisons Xen pour la virtualisation, nous avons recouru massivement au clonage pour les déploiements et nous nous sommes appuyés sur l'infrastructure que nous avions mise en place pour la gestion des serveurs d'application Zope et des sites Plone associés [5].

3.1 Neo

Pour la plateforme Neo, il faut d'une part pouvoir fournir rapidement aux utilisateurs les contenus vidéo et d'autre part permettre aux responsables de créer et mettre à jour des pages explicatives associées. Le service ayant été jugé critique par l'établissement, nous avons cherché à ce que l'architecture choisie permette une haute disponibilité.

L'architecture choisie se compose d'un répartiteur de charge frontal HAProxy et de plusieurs serveurs Apache en seconde ligne qui servent les vidéos, interprètent le code PHP pour générer des pages, authentifient les utilisateurs et agissent en tant que proxy pour des serveurs Zope/Plone en backoffice (voir Figure 1). Plus précisément en frontal, nous utilisons deux répartiteurs de charge avec Heartbeat, placés sur deux serveurs physiques distincts qui travaillent en binôme maître/esclave. Les deux répartiteurs partagent une adresse IP qui « flotte » entre les deux serveurs : si le maître ne répond plus, l'esclave prend immédiatement la main. L'esclave ne prend la main qu'en l'absence du maître et la lui rend immédiatement dès qu'il revient. Il n'y a ainsi pas de point de cassure supplémentaire du fait de la présence du répartiteur de charge redondé. Nous avons choisi d'utiliser la répartition au niveau du protocole HTTP, ce qui permet grâce au suivi du cookie, de maintenir un client sur le même serveur tant que ce dernier répond (voir [4] pour plus de détails).

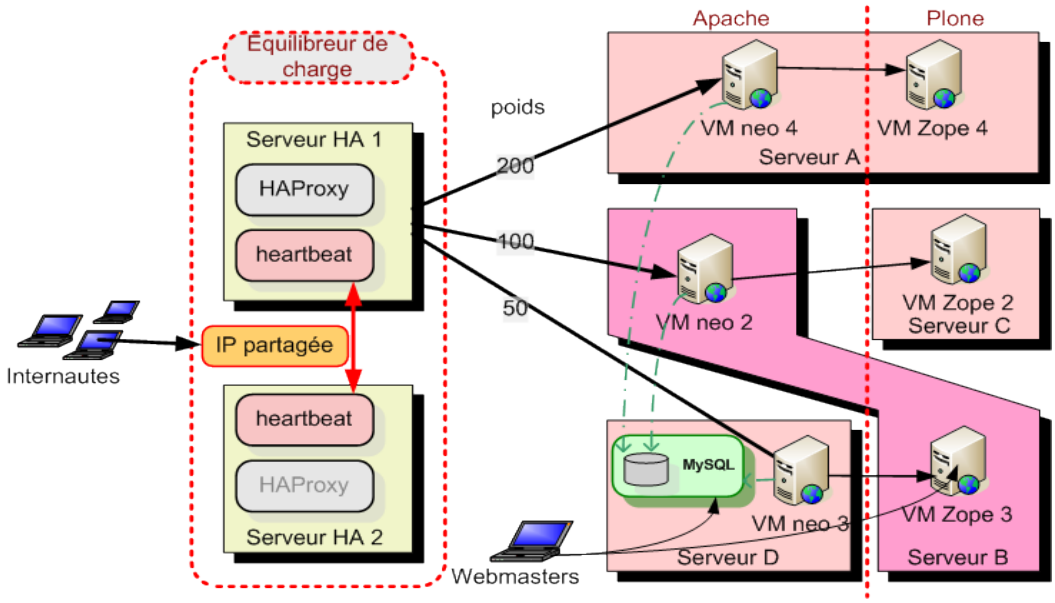

Figure 1 - *Architecture du site neo.inrp.fr*

Dans un premier temps, nous avions configuré HAProxy avec deux types de pondérations différents : 250-1 avec 2 serveurs, ce qui permettait que le second ne soit sollicité qu'en cas de défaillance du premier et correspondait à une absorption de charge et non une répartition. Puis nous avons pondéré en 200-100-50 avec trois serveurs pour répartir la charge principalement sur un serveur très puissant, pour un tiers environ sur un autre serveur moins puissant, mais totalement dédié au projet et très peu vers un dernier serveur récent pour le préserver, car il porte la base de données MySQL.

Au niveau du CMS, 3 machines virtuelles (VM) séparées tant au niveau du frontal Zope que de la base de données répondent aux requêtes envoyées par les serveurs Apache en amont (voir droite Figure 1). Les mises à jour de contenus sur Plone étant sporadiques, elles sont toujours effectuées sur le serveur Zope le moins exposé grâce à un accès Web spécifique fourni aux webmasters. Les modifications sont reportées sur les autres serveurs à la demande, lors de périodes creuses par des scripts automatisant la succession de tâches à effectuer : dé-validation des branches 2 et 4, snapshot LVM du disque du serveur 3, arrêt des instances Zope 2 et 4, synchronisation par rsync, redémarrage des Zope, re-validation des branches 2 et 4.

3.2 InterENS

Pour InterENS, la problématique est très différente de celle du site précédent : il n'y a ni contenus vidéos, ni mise en lumière médiatique. La charge est dans l'ensemble peu importante, mais augmente très fortement à l'annonce des résultats du concours.

Ce site est un cas d'école de dérive de projet auquel nous pouvons être confrontés : par manque de temps et de disponibilité des équipes de développement de plateformes Web à cette période, l'école a eu recourt à une prestation extérieure. Faute d'un cahier des charges suffisamment précis et d'un suivi suffisamment attentif de la prestation, la solution qui a été livrée par le prestataire était certes fonctionnelle[1], mais complexe et lourde. De plus, elle comportait des erreurs de configuration. La plateforme ne supportait pas la charge en raison de choix technologiques inadéquats (Plone). Le prestataire avait effectué quelques tests unitaires (avec Apache Bench), mais aucun test de charge représentatif. Pour résister à la charge, le prestataire préconisait d'utiliser un cache (Varnish), un répartiteur de charge frontal (Pound), quatre instances Zeo client avec cache, ainsi qu'une base de données centralisée Zeo serveur. Le tout devait fonctionner sur une seule et même VM trop peu dimensionnée tant en mémoire (2 Go RAM) qu'en CPU (2 cœurs) pour faire fonctionner une telle quantité de composants. La plateforme devait être remise en production sans temps disponible pour la re-développer avec des technologies adaptées. Nous avons donc étudié attentivement le système pour déduire les besoins de chacun des composants et déterminer le provisionning, c'est-à-dire les moyens nécessaires au bon fonctionnement de l'application (pour plus de détails, voir section 4). Nous avons choisi de répartir les différents éléments de l'architecture sur plusieurs VM en les clonant et en paramétrant les machines clonées pour arriver rapidement à la solution souhaitée.

[1]Les fonctions essentielles étaient opérationnelles, mais le retour arrière en cas d'échec ou d'erreur des utilisateurs notamment n'était pas prévu.

L'architecture était donc la suivante : un premier serveur Apache principal portant tous les éléments nécessaires au fonctionnement du site, mais incapable de supporter une charge importante et des serveurs puissants en backoffice fournissant l'essentiel du travail. Ainsi, la charge était principalement dirigée vers les 3 VM d'un serveur physique puissant avec tolérance à la panne grâce à une 4ème VM portée par un autre serveur physique. Dans cette architecture, il était particulièrement important de soulager au maximum le frontal. Le répartiteur de charge HAProxy a donc été configuré avec une pondération de 100 pour les 14 instances (Zeo client) portées par les serveurs backoffice et de 10 pour les instances du frontal (qui ne reçoivent qu'une requête sur 142). En revanche, contrairement à Neo, les instances partageaient la même base de données (Zeo serveur) et HAProxy n'a pas été configuré pour maintenir les clients sur une instance particulière : la totalité de la charge était répartie entre toutes les instances car la base centralisée permettait d'assurer le suivi de session.

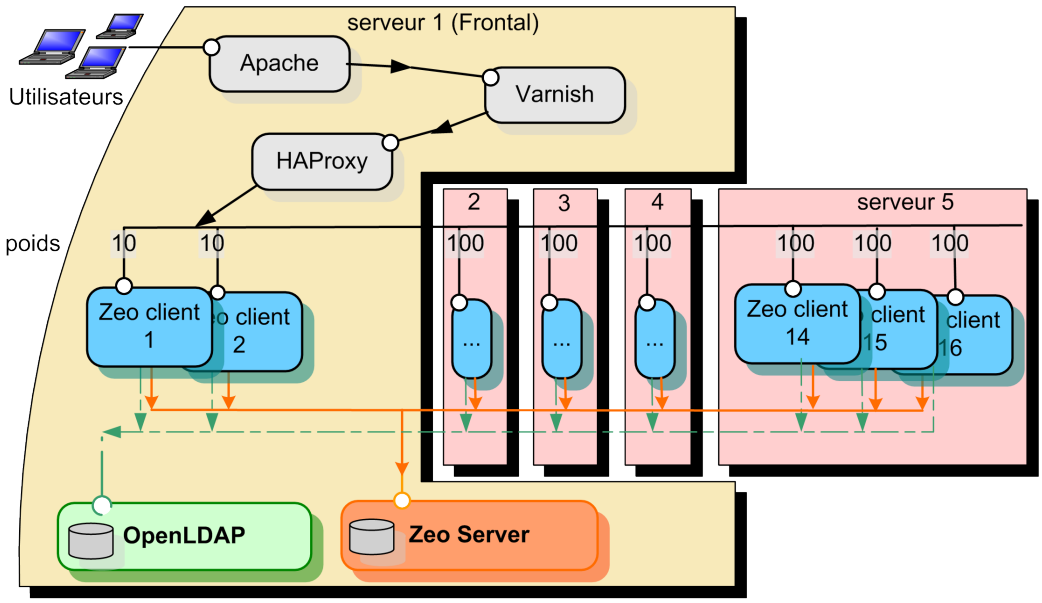

Figure 2 - *Architecture du site interens.ens-lyon.fr*

Pour le déploiement, nous avons cloné quatre fois la première VM, supprimé du démarrage les composants non souhaités et configuré le répartiteur de charge. Nous avons également adapté le nombre d'instances sur les VM clonées en fonction du nombre de cœurs CPU disponibles.

4 Tests de charge et provisionning

Pour vérifier la résistance à la charge de différentes architectures et établir le provisionning, nous avons réalisé des tests de charge. Nous avons sélectionné comme outil Apache Jmeter pour ses fonctionnalités : multi-plateformes, gestion de scénarios complexes, commande d'injecteurs répartis et fichiers statistiques.

4.1 Scénarios

Pour que le test soit le plus proche possible de la réalité, nous élaborons pour chaque site un ou plusieurs scénarios de test avec les responsables du site Web concerné. Le test de charge inclus les tests fonctionnels afin de mettre en évidence la faiblesse éventuelle d'un composant. Nous cherchons également à limiter les biais sur l'injection unique et le temps de latence. Pour cela nous utilisons une batterie d'injecteurs situés sur différents postes internes et externes afin d'être le plus proche possible de l'utilisation réelle. L'utilisation d'un réseau haut débit donnant un temps de latence de 1 ms, nous simulons l'ADSL en insérant de vieux équipements réseau (10 Mbps half duplex). Nous poursuivons les tests en augmentant le nombre d'utilisateurs par paliers jusqu'à obtenir l'indisponibilité du système testé. Nous faisons aussi varier les ressources serveur : mémoire et nombre de CPU.

Pour le site InterENS nous avons utilisé le scénario suivant : consultation de la page de garde du site (publique), puis d'une page d'explication (publique) ; accès au formulaire de connexion ; connexion ; accès à l'espace personnel ; récupération du PDF de convocation ; déconnexion. Nous avons aussi souhaité que chaque test unitaire soit différent ; pour cela nous avons utilisé des tables associées au scénario contenant le login, le mot de passe et le nom du document à récupérer.

4.2 Observation et analyse

Pendant le test de charge, nous vérifions d'une part la réactivité ressentie du site en y navigant. D'autre part, nous surveillons la charge au niveau de la machine avec les outils suivants : iostat (pour les accès disques), ps, top (pour la mémoire et la consommation CPU), netstat (accès réseau) et le tableau de statistiques fourni par HAProxy.

L'analyse se base sur les statistiques de Jmeter, sur les observations et sur l'ensemble des logs (apache, système et équipements réseau traversés). Les éléments analysés permettent d'une part de mesurer le nombre de clients à partir duquel les temps de réponse du serveur ne sont plus acceptables pour les utilisateurs. D'autre part, ils permettent de connaître les besoins des différents composants de l'architecture et d'en déduire le provisionning en fonction de la charge attendue.

4.3 Provisionning des ressources

Lorsque l'on met en place un nouveau service, il est toujours difficile d'imaginer l'intérêt qu'il suscitera et en conséquence le trafic qu'il engendrera. En revanche, en analysant le comportement du trafic sur d'autres sites, on peut émettre des hypothèses sur le trafic attendu et ses variations. Si l'on connaît la consommation de ressources des composants que l'on met en œuvre, on peut la croiser avec le trafic attendu pour déterminer les besoins en termes ressources.

Dans le cas d'InterENS, nous avons mis en évidence que tous les threads d'une instance Zope restent attachés au même cœur : augmenter le nombre de CPU est donc inutile dans ce cas car ils ne seront pas utilisés. Nous avons ensuite estimé les besoins en mémoire des différents composants en utilisant l'observation des ressources avant et après le test de charge. Le résultat de l'étude sur le serveur n°1 avec 4 instances donne : 4 Go de RAM / 6 cœurs en période creuse ; 7 Go de RAM en période de charge.

5 Pics de charge observés

On définit généralement la charge sur un système par le taux d'occupation du processeur et des entrées / sorties. Pour un serveur Web, la charge sera fonction du nombre, de la taille et de la complexité des pages à servir aux clients : la charge augmentera avec la complexité des pages et le nombre de clients. On peut considérer que les logs de connexion du serveur fournissent des indications sur l'affluence et par extension la charge du serveur. On pourra parler d'un « pic de charge » si la charge atteint ponctuellement des valeurs plus élevées que celles généralement constatées pour le même service. Le pic peut être le fruit de l'action d'une multitude d'individus, de quelques individus très consommateurs, ou encore de robots d'indexation.

Que l'on utilise le mode par défaut « common » ou le mode « combined » pour les informations supplémentaires qu'il apporte sur les clients, dans le fichier de log d'Apache figurent notamment pour chaque requête l'adresse IP du client, la page demandée, l'heure d'arrivée de la requête. Pour les graphiques présentés par la suite, nous avons calculé le nombre de « hits » par seconde sur le serveur grâce à l'information d'heure d'arrivée des requêtes. Comme ceci est difficilement exploitable, en fonction des besoins, nous avons calculé des cumuls sur 10 secondes, 1 minute, 10 minutes. Ces valeurs permettent de bien suivre l'évolution de l'affluence sur un site, mais ne permettent pas nécessairement de comparer entre eux des sites différents. A titre d'exemple, un CMS peut générer 20 hits pour une page sans contenu. D'autres indicateurs comme le nombre de pages, celui d'IP distinctes (ce qui correspond approximativement aux clients différents) ou la bande passante employée peuvent parfois être plus judicieux. Enfin, avant de décrire les pics de charge que nous avons observés, nous attirons l'attention du lecteur sur le fait que l'architecture peut modifier l'observation : par exemple, en cas de saturation complète, on n'aura plus de connexion, pourtant il y aura peut être quand même des clients souhaitant voir le site.

5.1 Pic de charge sur médiatisation

L'application Neo a fait l'objet d'une douzaine d'expositions médiatiques dans la presse écrite et audiovisuelle que nous avons suivies dans les logs Apache. Pour le passage en presse écrite, on constate une augmentation du trafic visible sur la volumétrie de la journée, mais répartie tout au long de la journée, sans pic. Pour les cinq diffusions au journal télévisé, nous avons en revanche observé des pics de charge très similaires dont seule l'amplitude variait (voir Figure 3) : on constate une augmentation très brusque de la charge pendant la diffusion du reportage et une très forte diminution immédiatement après. Les utilisateurs supplémentaires sont très volatiles et ne restent sur le site que quelques secondes. L'impact est tel qu'il est même visible sur le trafic réseau : pour le quatrième exemple, il y avait un pic de 25 Mbps sur le routeur d'entrée du site.

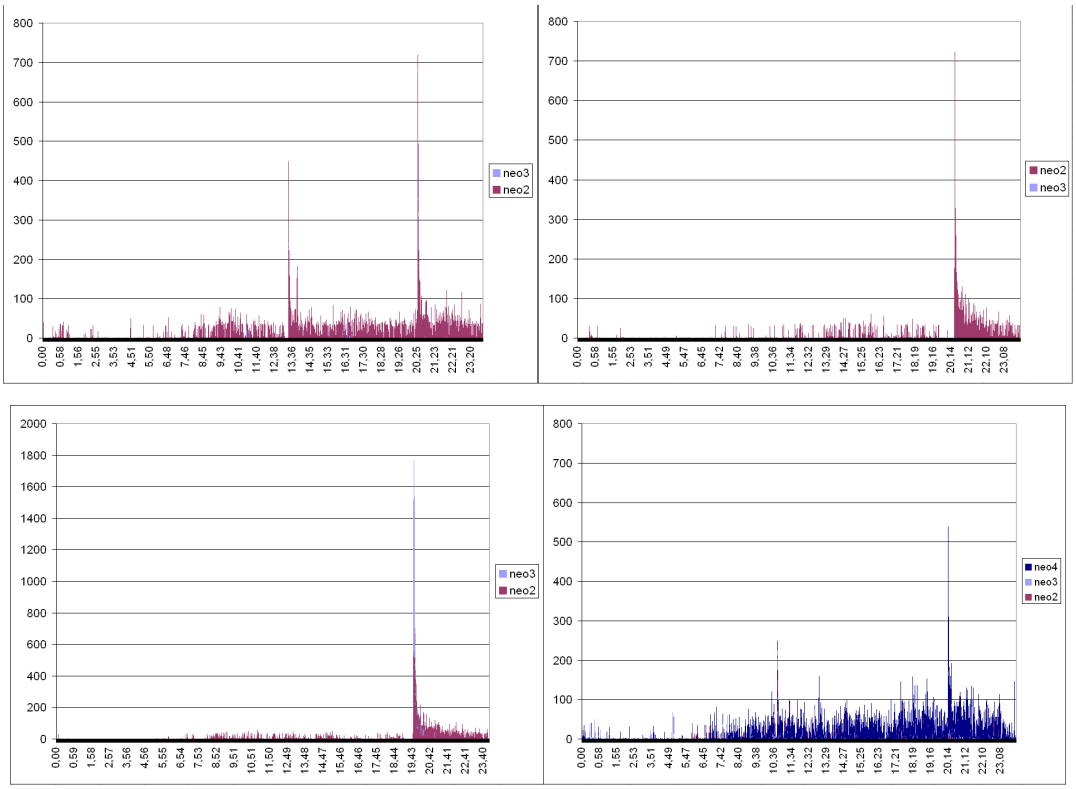

Figure 3 - *Pics de charge observés lors des journaux télévisés nationaux (cumuls sur 10 secondes)*

5.2 Pic de charge résultat de concours

Pour ce cas de figure, nous n'avons eu la possibilité d'observer qu'une seule occurrence (voir Figure 4). La charge a légèrement augmenté avant l'annonce des résultats du concours, puis on constate un pic avec une augmentation brutale de l'affluence. Contrairement aux pics médiatiques, la charge reste ensuite soutenue et décroit lentement sur une période de plusieurs heures.

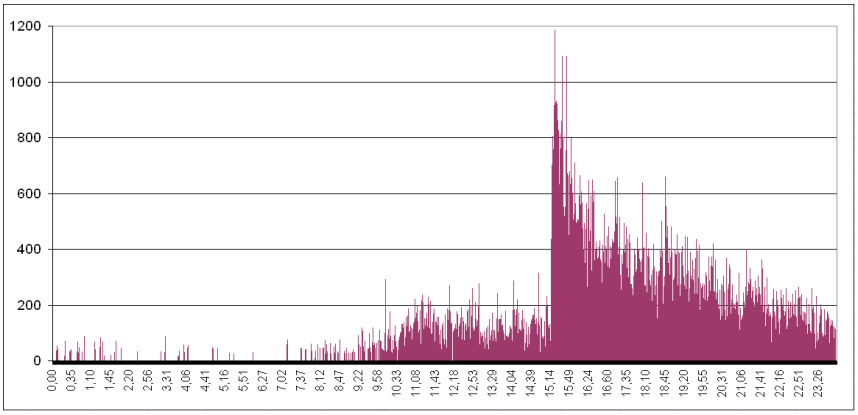

Figure 4 - *Hits cumulés sur des plages de 10 secondes le jour des résultats du concours*

6 Atouts et limites de la répartition de charge

Face à un problème de performances, les axes de résolution qui s'offrent sont : l'augmentation des performances par l'achat d'un nouveau serveur plus puissant; ou la distribution de la charge en la répartissant entre plusieurs serveurs standards. Si de plus on veut assurer une haute disponibilité de service, le premier axe impliquera l'achat d'un serveur non seulement puissant, mais aussi très fiable et s'accompagnera d'un bon contrat de maintenance. Dans notre cas, nous avons exploré le second axe qui nécessite davantage de temps de déploiement et l'acquisition de la maitrise de la répartition de charge, mais présente l'avantage d'assurer la

disponibilité en se rendant tolérant aux pannes via la duplication des composants. Par ailleurs, la virtualisation simplifie le clonage des serveurs et le déploiement d'une architecture avec des machines répliquées.

Avec une architecture répartie, le clonage simple suffit pour les sites statiques, mais le problème se complique pour les sites dynamiques. En effet, dans ce cas les données présentes sur le site peuvent être modifiées et il faut s'assurer de leur cohérence entre les différents serveurs. Ces données seront stockées soit sur un système de fichiers, soit dans une base de données, principalement sur le SGBD MySQL, mais aussi sur OpenLDAP ou ZoDB et différentes approches sont possibles pour gérer la répartition pour les deux types de stockage de données.

6.1 Bases de données

MySQL propose deux options pour répartir les données : la réplication et le cluster. Le cluster MySQL réduit les fonctionnalités SQL, ce qui peut se révéler incompatible avec l'une des applications qu'on souhaite répartir (par exemple si l'on y utilise des clés étrangères). Le mode cluster n'est d'ailleurs pas proposé par Debian. Si l'on souhaite rester dans des outils standards, il faudra s'orienter vers la réplication de type maître/maître ou maître/esclave. Dans les deux exemples d'architectures que nous avons présentés, des bases de données sont utilisées : ZoDB et OpenLDAP pour InterENS, ZoDB et MySQL pour Neo. Les besoins étant différents, nous avons adapté nos choix, mais dans aucun cas nous n'avons pour l'instant eu recours en production à des bases de données distribuées : pour InterENS, il y a une base ZoDB centralisée fournie par Zeo serveur à tous les Zeo clients et un OpenLDAP centralisé également. Pour Neo, la base MySQL est centralisée, nous l'avons mise sur un serveur peu sollicité, les bases ZoDB sont indépendantes et sont synchronisées à la demande quand il y a eu une modification.

6.2 Système de fichiers

Pour les données stockées sur le système de fichier, on peut aborder le problème de différentes manières. Si les données changent fréquemment et qu'il est indispensable d'avoir en permanence des données à jour, il faudra s'orienter vers des solutions de type système de fichiers répartis (par exemple GFS [6]). Ceci a déjà été étudiée dans la communauté des JRES (voir [7] et [8] entre autres). Ces solutions sont délicates à mettre en place et doivent être testées en profondeur, en particulier pour savoir comment assurer la reprise en cas de problème quel que soit le cas de figure. Si la mise à jour est moins critique et peut attendre quelques minutes, des solution beaucoup plus simples peuvent être mises en place en utilisant par exemple Rsync lancé automatiquement à intervalles réguliers.

6.3 Pour l'exploitation des systèmes

Une architecture avec répartition de charge implique d'avoir beaucoup plus de machines que si l'on utilise un seul calculateur puissant et donc beaucoup plus de risques de pannes, même si chaque panne aura en principe un impact très réduit. De plus, dans le cas d'architectures avec système de fichiers et bases de données répartis tels que celles que nous avons testées, le risque de désynchronisation de données est fort et nécessite une bonne maîtrise de l'architecture pour l'exploiter.

La répartition engendre aussi beaucoup d'administration système pour maintenir les systèmes à jour. La principale difficulté se produit lors des mises à jour de noyau qui sont finalement assez fréquentes et nécessitent pour être appliquées de redémarrer le serveur. Dans les architectures que nous avons exposées, nous n'avons pas été jusqu'à tout répartir pour limiter la complexité de la solution mise en œuvre, pourtant, la quasi totalité des composants est redondée. Pour une mise à jour de noyau, nous pouvons interrompre les serveurs sans que cela ne soit visible sur la production (à condition de ne pas le faire simultanément). Il reste toutefois nécessaire de bien maîtriser l'architecture pour savoir ce qui ne doit pas être arrêté simultanément.

Un autre aspect intéressant de la répartition avec virtualisation pour des serveurs sujets à des pics de charge saisonniers est qu'il est possible d'arrêter des VM en dehors de la période de charge, en utilisant la puissance de calcul du serveur pour une autre application et de les redémarrer en cas de besoin.

7 Conclusions et perspectives

Au regard de notre expérience, mettre en place une répartition de charge « modeste », sans chercher à avoir une architecture symétrique entièrement redondée, nous semble être la meilleure solution. Plus la complexité de l'architecture augmente plus elle est difficile à maintenir et il faut évaluer le gain apporté par une solution plus complexe par rapport à l'effort de déploiement et d'exploitation de la solution. Le surcoût budgétaire en temps d'ingénieur pour la mise en place des solutions que nous avons utilisées est très réduit par rapport à celui correspondant à l'achat des calculateurs puissants capables d'absorber à eux seul la

charge et des contrats de maintenance associés pour assurer une haute disponibilité. De plus, une fois la maitrise acquise elle est réutilisable et c'est même le cas d'une partie des infrastructures. En revanche, permettre une haute-disponibilité est aussi une question d'organisation des équipes et l'infrastructure technique seule n'en est qu'une partie.

Les solutions que nous avons mises en place se sont révélées performantes et robustes : le site interENS a supporté le pic de charge lié à l'annonce des résultats du concours et les autres montées de charge. En observant le trafic entrant au niveau de l'Apache frontal et celui vu au niveau d'HAProxy, nous avons constaté une diminution conséquente (de l'ordre de 30%) liée au travail du cache Varnish qui s'avère être un composant particulièrement important de l'architecture car il permet d'en partie pallier le manque de performance de Plone. Le site Neo n'a connu aucune interruption autre que celles dues à des pannes électrique et/ou réseau depuis son lancement en Aout 2010. Pour l'exemple, à la date de rédaction de cet article, l'HAProxy principal tourne depuis 49 jours, a servi 363 Go de données, et indique 0h31, 3h39 et 3h12 de maintenance respectivement pour les branches 2, 3 et 4, et un total de 0 seconde d'interruption pour les clients, les opérations de maintenance n'ayant pas été simultanées sur les différents serveurs. Il est difficile d'avoir des données précises sur un long terme car HAProxy est réinitialisé à chaque maintenance même si le service n'est pas interrompu pour les utilisateurs car l'esclave prend alors la main pour maintenir le service.

La plateforme InterENS est en cours de redéveloppement en PHP/Mysql pour la partie liée à la diffusion des résultats et la plateforme Neo a également évolué en proposant maintenant du HTML5 (vidéos lisibles directement par les navigateurs, sans recourir à Flash). Les activités présentées sont également en cours d'enrichissement.

8 Bibliographie

[1] Pierre-Jean Duvivier, Les pics de charge avec Drupal. En ligne. http://www.media-business.biz/content/traitement-des-pics-de-charge-dans-un-environnement-complexe-drupal

[2] Jean Xech, Albertine Rabat, Aline Savarèse, Comparatif de serveurs de contenus (CMS – Content Management Server). Dans *Actes du congrès JRES2007*, pages 455-464, Strasbourg, 20-23 Novembre 2007, http://2007.jres.org/planning/pdf/104.pdf

[3] Frédéric Saint-Marcel et Philippe Lecler, Plone, un outil de gestion de contenu Web. Dans *Actes du congrès JRES2005*, pages 131-139, Marseille, Décembre 2005. http://2005.jres.org/paper/46.pdf

[4] Frédéric Soulier, Haute disponibilité : HaProxy et Heartbeat pour la sécurisation d'un portail Web. TutoJres 21/04/2009, http://www.jres.org/_media/tuto/tuto10/8-hd_securisation_portail_Web.pdf

[5] Philippe Daubias, Rationalisation de l'exploitation de serveurs Zope/Plone avec Xen, Dans *Actes en ligne du congrès JRES2009*, Nantes, 1-4 Décembre 2009. https://2009.jres.org/planning_files/article/pdf/43.pdf

[6] Red Hat Cluster Service 2 Tutorial, https://alteeve.com/w/Red_Hat_Cluster_Service_2_Tutorial

[7] Yann Dupont et Yoann Juet, Mécanismes de haute-disponibilité côté système : retours d'expérience sous Gnu/Linux, *TutoJRES*, 2009, http://www.jres.org/_media/tuto/tuto10/3-hd_systeme_linux.pdf

[8] Nicolas Schmitz, Solution Haute Disponibilité pour Linux : Un cluster avec Heartbeat et Drbd. Dans *Actes du congrès JRES2005*, pages 241-250, Marseille, Décembre 2005, http://2005.jres.org/paper/91.pdf