Optimisation Moodle

à l'Université de Rouen Normandie en 2020

Version 2

I. Introduction

Le présent document tente de retranscrire les travaux d'optimisation réalisés fin avril et début mai sur la plateforme moodle de l'Université de Rouen Normandie nommée universitice.

Ces optimisations ont été réalisées pour faire face techniquement à un usage accru de la plateforme lors du confinement (crise du covid-19) et des examens en distanciel qui ont suivi.

Ce document se veut un retour objectif de ce qui a été réalisé et des résultats obtenus.

D'autres solutions auraient sans doute permis d'obtenir les mêmes résultats, voire des résultats meilleurs.

Certains établissements partenaires qui ont opté pour d'autres stratégies ont également amélioré sensiblement leur plateforme.

Ce document est avant tout technique. Nous n'aborderons donc pas les mesures fonctionnelles qui permettent également (voire plus encore) une bonne tenue de montée en charge des différents outils numériques dont la plateforme pédagogique moodle (étalement des examens en ligne, préférence pour les activités asynchrones, …).

En avril 2020, le Système d'Information de l'Université de Rouen Normandie recense 33902 comptes étudiants activés.

En comptant les personnels (dont les enseignants), le nombre d'usagers potentiels dépasse donc les 35000.

II. Contexte

L'Université de Rouen Normandie utilise moodle comme outil de plateforme pédagogique.

En avril 2020, cette plateforme s'appelle Universitice, elle est accessible depuis https://universitice.univ-rouen.fr/, elle correspond à une version 3.5 de moodle.

La base de données utilisée est PostgreSQL.

Sa taille (database_size) est de 80GB, elle occupe sur disque 40GB (répertoire data), son dump compressé est de 6.7GB.

Moodle est servi par un apache en MPM prefork (mod_php).

Toute l'application (frontal apache, base de données) est sur un seul et même serveur physique doté de 24 CPUS [Intel(R) Xeon(R) Gold 6128 CPU @ 3.40GHz], 64 GB de RAM, disques montés en ZFS. La distribution linux utilisée est la CentOs 7.8. Version de PHP 7.0.27 et Apache 2.4.34.

Lors de sa mise en place, ce serveur n'a pas nécessité de paramétrages spécifiques/techniques au niveau du frontal apache et de la base de données ; les configurations par défaut ont permis une exploitation satisfaisante de la plateforme.

Des configurations spécifiques (optimisations) avaient cependant été réalisées au niveau moodle (cache moodle, opcache) et système couche basse (ZFS).

III. Problème

Le confinement, puis la perspective des examens en distanciel utilisant principalement comme outil cette plateforme pédagogique, a modifié profondément l'usage de moodle au sein de notre université : l'usage s'est fortement accentué. On estime qu'au 17 mars les usages avaient déjà pratiquement doublé.

Le nombre de connexions a augmenté. La qualité de service s'est détériorée.

Pour l'usager final, cette qualité de service s'est d'autant plus détriorée que lui aussi a pu voir ses conditions d'utilisation être dégradées, ce par le fait d'une connexion internet lente par exemple (alors que pour certains, et depuis les résidences universitaires par exemple ou l'enceinte même de l'université, on peut estimer que la connexion était locale au serveur). Du point de vue du serveur, les clients se sont donc en moyenne détériorés ; cela a un impact côt serveur dans le sens où un client lent augmente également le nombre de clients à traiter à un instant t.

Si on compare cela à une autoroute, nous sommes passés d'un trafic peu chargé avec des véhicules rapides à un trafic très chargé avec des véhicules plus lents ; le service moodle étant l'autoroute.

Les outils de mesure en place montraient des pics de charge sur les différentes ressources : pics de connexions apache et PostgreSQL notamment.

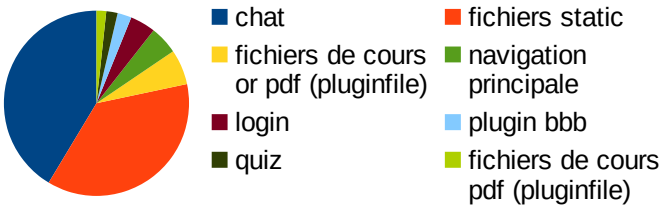
La première réponse dans l'urgence a été de monter les différents curseurs : nombre de connexions maximal côté apache, nombre de connexions maximal côté PostgreSQL (~800 pour chaque, là où on avait comme maximum 200 environ). Une baisse du keeepalive apache a également été effectuée.

Cela a permis de répondre à l'urgence.

La situation n'était pas idéale, des lenteurs étaient constatées, le service restait dégradé.

Des interruptions de service étaient à craindre.

On constatait que les requêtes effectuées étaient réparties ainsi :



IV. Solutions

La consultation de collègues d'autres établissements au travers du Groupe de Travail Développement EsupPortail (qui a organisé une réunion de travail à distance le 27 Avril 2020) nous a permis de prendre connaissance des solutions et expériences de l'Université Paris1, l'Université de Lorraine, Aix Marseille Université, GIP RECIA, Université de Lille, entre autres.

Ces échanges nous ont permis d'amorcer ces travaux avec en tête une idée assez claire de ce que nous devions réaliser.

1 Moodle

1 Chat

Le module chat de moodle est particulièrement gourmand dans sa configuration Ajax (~ 2 requêtes par seconde pour chaque client).

Ces requêtes opérées par moodle sont coûteuses car monopolisant un thread apache, ce même si le temps de réponse est très faible (< 30ms).

Ici on a opté (comme les autres établissements) pour la diminution du nombre de requêtes en modifiant la configuration du module : nous sommes passés de la méthode 'Ajax' à la méthode 'Normal' avec une actualisation toutes les 5 secondes (qui est la valeur par défaut pour ce mode). Cela a entraîné 2 régressions sur le plan fonctionnel : interface moins ergonomique et réactive avec perte de la possibilité de 'chater' en privé avec un usager.

Aussi dans les améliorations à envisager, utiliser un outil de chat dédié comme peut l'être rocket.chat permettrait de décharger moodle de cette activité. Ce d'autant qu'un plugin moodle existerait (non testé) : https://github.com/adpe/moodle-local_rocketchat

2 OPCache

Cet outil compile et met en cache les fichiers php. Déjà configuré, on a modifié la fréquence de revalidation (opcache.revalidate_freq).

On l'a passé de 2sec à 60sec comme préconisé dans la documentation moodle.

Même si notre cache de 128MB (opcache.memory_consumption) correspondait à la taille de notre moodle et modules associées (la somme des tailles des fichiers php fait 117MB), le apache fourni est également utilisé pour servir un autre site web et certains contenus de cours sont eux-mêmes en php (quelques dizaines de MB).

Nous avons donc passé cette valeur à 256MB (on constate un usage d'environ 140MB).

3 Sessions moodle

Notre moodle mettait les sessions en base de données.

Ce paramétrage, conseillé et donné par défaut auparavant, est maintenant déconseillé.

Des erreurs apache apparaissaient ainsi :

- Cannot obtain session lock for sid: ...

Cela accompagné de locks persistants au niveau de la base de données qui avaient eux-mêmes des répercussions sur la bonne marche de PostgreSQL : on trouvait ainsi dans les logs PostgreSQL des erreurs type :

- ERROR: canceling statement due to statement timeout STATEMENT: SELECT pg_advisory_lock(56701458)

On a modifié cela pour gérer les sessions en fichier, comme proposé maintenant par défaut sur moodle.

4 Cache moodle

Le cache moodle (répertoires cache, mcu, localcache [et sessions]) est configuré sur universitice pour utiliser le système de fichiers en ZFS.

A vu des retours des établissements ainsi que des forums et documentations moodle, nous sommes en train de passer ces répertoires en tmpfs (RAM).

2 NGINX en proxy

Le Apache en prefork proposé directement au client pose un certain nombre de problèmes.

À la fois les connexions sont coûteuses et ne peuvent de fait pas être augmentées de beaucoup :

- une connexion prefork monopolise un thread prefork qui peut prendre 100MB de RAM ;
- le MPM prefork n'est pas connu pour ses performances, lui sont préférés worker et event, mais seul prefork serait compatible avec mod_php.

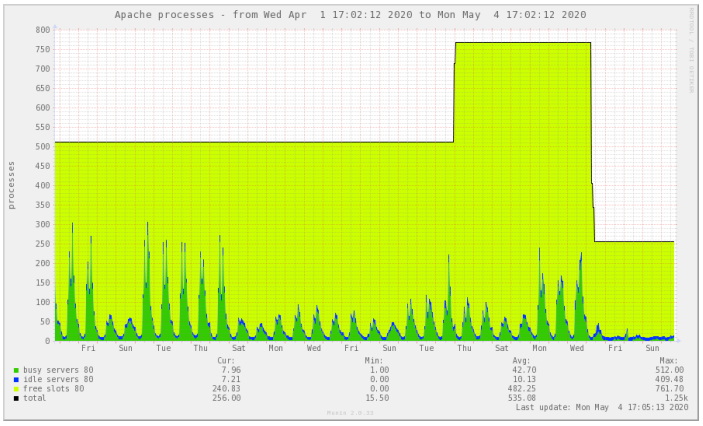
Et à la fois, c'est le client qui détermine en grande partie l'usage de la connexion Apache, encore plus si on souhaite laisser le keepalive actif pour minimiser le temps de connexion et fluidifier la navigation de l'utilisateur.

- ainsi une connexion lente pour (télé)charger un fichier monopolisera un thread pendant un temps parfois conséquent.
- de même ces connexions coûteuses sont utilisées en grande partie pour servir de simples fichiers statiques (images, css, js) … la mise en cache côté navigateur permet d'éviter que cela corresponde à 90 % des requêtes … mais 40% (cf graphe ci-dessus) des requêtes utilisées à cette tâche sur le apache reste dommageable.

La mise en place d'un proxy tel que NGINX sur un serveur dédié apporte donc un certain nombre d'avantages :

- bénéficier des performances de NGINX vis à vis du client :
  - gestion de milliers de connexion,
  - possibilité d'un keepalive élevé (65 secondes) permettant une connexion plus fluide pour le client
- proposer à apache prefork un client exemplaire :
  - en bufferisant les requêtes/réponses (dont download/upload de fichiers) du client navigateur, les connexions apache sont utilisées de manière optimale : suppression du temps de connexion utilisée pour simplement transférer à un client (lent) les requêtes et réponses.
  - pas de pré-connexion abusive
  - connexions apache uniquement pour les pages dynamiques, NGINX mettant en cache l'ensemble des pages statiques (en les mettant à jour simplement suivant les en-têtes cache-control positionnées par moodle).
- déporter un certain nombre de tâches 'élémentaires' à NGINX :
  - cache
  - ssl
  - gzip
- enfin séparer la gestion des appels clients des appels au serveur moodle lui-même permet d'y voir beaucoup plus clair sur les points de vigilance à observer, voire les défaillances.
  - on peut notamment distinguer les latences dues au moodle, des latences dues aux problèmes de connexion de l'usager : on peut considérer qu'en loguant/mesurant les temps de réponse apache, on mesure les temps de traitement de la requête par le apache uniquement.
  - les appels étant très élagués côté apache/moodle, il est également plus facile de comprendre et résoudre les points de blocage éventuels.

Le graphe suivant montre la baisse significative du nombre de connexions occupées sur le apache prefork suite à la mise en place du proxy NGINX:



La mise en place d'un proxy est quelque chose de classique dans certains établissements.

Il est parfois utilisé en reverse-proxy de manière généralisée dans des établissements partenaires.

Cette mise en place a été déterminante. C'est la première chose que nous avons décidé de faire sur notre infrastructure, et c'est celle qui nous a permis par la suite d'y voir clair dans les optimisations à réaliser, notamment côté PostgreSQL.

La question du choix de la solution de proxy peut se poser. Nous avons choisi NGINX car il nous a semblé le plus à même de répondre aux besoins décrits ci-dessus, ce notamment vis à vis d'autres solutions libres telles que Apache (avec MPM worker ou event) ou HAProxy : bufferisation des requêtes/réponses, cache, keepealive ...

3 Apache

Avec un NGINX en frontal, le Apache se voit finalement peu sollicité dans le sens où à un instant t :

- presque la moitié des requêtes est traitée directement par NGINX (fichiers statiques)
- « peu » de requêtes sont en cours car rapidement traitées, les réponses sont transférées très rapidement au NGINX directement sur le réseau local.

Avec un relevé matomo de 2000 sessions sur les 30 dernières minutes, 400 visites sur les 3 dernières minutes (chiffres de fait sous-estimés puisque les navigateurs incluant des bloqueurs type ublock ne sont pas comptabilisés ici) ou encore 800 utilisateurs en ligne sur les 5 dernières minutes (tableau de bord moodle), on reste en moyenne autour de 15 requêtes en parallèle sur le /server-status apache (mod_status) à un instant t et une moyenne de plus de 30 requêtes par seconde.

Le fait que moodle subisse des usages non prévus, non encouragés mais pratiqués (vidéo dans moodle au lieu de passer par le serveur vidéo) implique le fait que certaines connexions restent persistantes et augmentent le nombre de connexions à un instant t (temps du streaming).

Les indications en direct du /server-status apache s'avèrent ainsi très précieuses pour comprendre quelles requêtes restent effectivement persistantes, correspondent à des requêtes sql problématiques ...

Finalement, grâce au proxy NGINX et aux optimisations de configuration côté PostgreSQL pour fluidifier au maximum le requêtage en base, le MaxClients a pu être descendu.

On l'a laissé cependant assez haut par rapport à l'usage (200) pour prendre en compte les requêtes de chargement vidéo.

Le keepalive a été remis par défaut … et on a désactivé les tâches que l'on a pu reporter sur NGINX (ssl, cache, gzip).

4 PostgreSQL

Il y a quelques années, le choix a été fait d'utiliser PostgreSQL pour le serveur moodle. Sur des grosses installations, cette base de données est remarquablement performante, et au moins une partie de la communauté moodle estime PostgreSQL comme plus performante que MySQL[1]. Moodle étant cependant développé dans une logique technologique 'LAMP', il est fort probable qu'il puisse y avoir ici débat !

Le fait est que le moodle de l'Université de Rouen Normandie utilise PostgreSQL. Suite au confinement, son usage

1https://docs.moodle.org/38/en/Arguments_in_favour_of_PostgreSQL

intensif avec des requêtes parfois coûteuses de la part du module quiz nous a conduits à modifier les paramètres donnés par défaut.

PostgreSQL peut par défaut tourner sur des matériels très variés.

Aussi sur une installation serveur fortement sollicitée, il est probable de devoir modifier quelques paramètres.

Sur le serveur moodle, seul le shared_buffers avait été modifié jusque là et l'usage de ZFS permettait à PostgreSQL de tenir la charge usuelle avec ses configurations par défaut, de manière honorable.

Suite à la sollicitation massive du moodle, les graphes montraient un PostgreSQL en souffrance au niveau des vacuum ainsi que des requêtes longues notamment. Les logs PostgreSQL rapportaient effectivement des problèmes de vacumm.

Entre le vacuum à optimiser (notamment vérifier que le pg_stat_tmp est monté en RAM ; cela est fait par défaut sous debian mais pas sous centos ; monter également maintenance_work_mem) et les requêtes longues à prendre en compte (work_mem), un bon nombre de paramètres peuvent être étudiés et discutés.

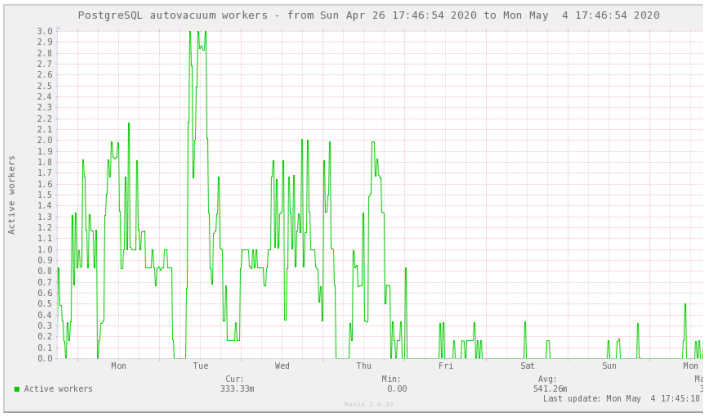
La documentation ne manque pas à ce sujet et s'y plonger est réellement instructif.

Un outil comme https://pgtune.leopard.in.ua peut aussi donner des paramétrages pertinents de manière rapide et efficace.

Sur les 64 GB de RAM, nous estimons la moitié à dédier au ZFS.

Nous avons actuellement positionné les configurations suivantes sur notre PostgreSQL 9.6 (les autres sont donc celles données par défaut ; attention également que d'une version à une autre, les paramètres évoluent, leurs valeurs par défaut également) :

- max_connections = 300
- shared_buffers = 6GB
- effective_cache_size = 20GB
- maintenance_work_mem = 1.5GB
- checkpoint_completion_target = 0.7
- random_page_cost = 2
- effective_io_concurrency = 50
- work_mem = 16MB
- min_wal_size=160MB
- max_wal_size=2GB



Notons que l'on a mis un max_connections plus élevé que le MaxClients apache, pour 2 raisons :

- postgresql réserve des connexions pour les super-utilisateurs, aussi l'ensemble des connexions ne peuvent pas être utilisées par Apache, de plus les tâches moodle externes (cron.php, synchronisation avec le SI) utilisent également des connexions PostgreSQL ;
- avec un ServerLimit plus haut que le MaxClients, on peut augmenter le MaxClients à chaud côté Apache (reload), mais la modification du max_connections de PostgreSQL nécessite un redémarrage du PostgreSQL (moodle n'utilise pas de pool de connexions interne aussi le nombre de connexions apache et le nombre de connexions PostgreSQL fluctuent de concert).

5 Autre

Lors des périodes de forte charge, nous avons pu constater que les tâches cron de synchronisation avec le Système d'Information pouvaient potentiellement déstabiliser le système.

Même si nous ne pouvons pas certifier que cela a effectivement pu engendrer des perturbations pour l'utilisateur, il nous est apparu plus sage de diminuer les fréquences de synchronisation avec le Système d'Information et de les décaler sur des périodes 'creuses'.

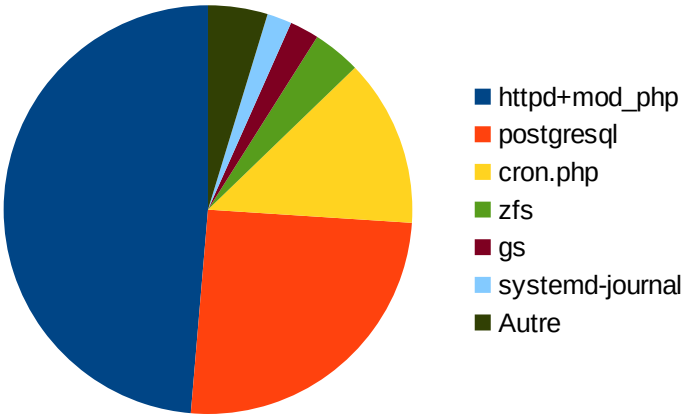
V. Mesures

1 Côté Serveur

Le serveur NGINX très sollicité n'utilise que très peu de ressources. La solution est stable et fiable.

Concernant le serveur moodle lui-même, serveur physique monté en zfs sur lequel nous avons le apache+mod_php+PostgreSQL, et après les optimisations données ci-dessus effectuées, le système semble être exploité de manière efficace. On observe moins de latence ou de locks au niveau de la base de données notamment.

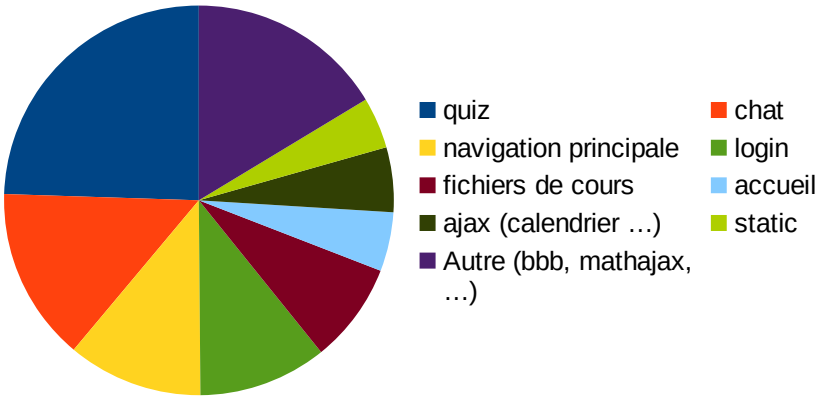
Sur quelques minutes de temps voici comment est répartie la consommation cpu[2] (cela donne simplement un ordre d'idée ici, il est possible notamment que la part de PostgreSQL soit sous-estimée par exemple, l'échantillonage a été fait sur une courte durée sur un moment donné de la journée [relativement chargé]).



Les sollicitations sont maintenant consacrées avant tout à des requêtes dynamiques.

Les mesures suivantes ont été prises lors de la période d'examen le 6 mai 2020 au matin, elles correspondent donc à une analyse des requêtes qui arrivent sur le serveur apache.

L'activité quiz prédomine, l'activité chat continue d'être sollicitée.



Sur la journée du jeudi 7 mai, avec déjà quelques examens à distances opérés, voici les statistiques relevées entre 8H00 et 18H00 sur la plateforme moodle.

Rappel : sur des requêtes 'usuelles' (cad hors téléchargement de 'très gros' fichiers que l'on ne bufferise pas par NGINX) on peut estimer le temps de réponse côté Apache comme étant le temps d'exécution/calcul de la requête/réponse, alors que le temps de réponse NGINX dépend également du client (correspond donc à un temps de requête opéré sur le apache [sauf cache pour les fichiers statiques] ajouté d'un temps de latence).

2 LC_ALL=C top -b | perl -lne 'if (/^\s*PID\s/ ... /^$/) { @l = split; $s{$l[11]} += $l[8] } if (/^$/) { print "$_ $s{$_}" foreach sort { $s{$a} <=> $s{$b} } keys %s; print "" }'
(merci à Pascal à nouveau pour le partage de cette commande)

Type de requête (avec réponse HTTP 200 uniquement)	Nombre de requêtes Apache	Temps de réponse Apache moyen	Nombre de requêtes NGINX	Temps de réponse NGINX moyen
Total	906 612	236.72ms	1 847 760	276.61ms
Quiz (/mod/quiz)	121 800	132.36ms	226 361	153.66ms
Chat (/mod/chat)	222 837	25.68ms	222 837	26.87ms
Login (/login)	22 954	122.15ms	22 954	123.99ms
Navigation principale (/course)	120 881	232.13ms	120 881	243.74ms
Rapports Quiz (/mod/quiz/report.php)	439	994ms	-	-
Fichiers cours (dont films, pdf, html ... /pluginfile.php)	99 388	1.19s	170 494	2.11s
Fichiers cours (pdf uniquement /pluginfile.php)	23 695	1.17s	23 698	3.71s
Fichiers cours (mp4 uniquement /pluginfile.php)	4 339	16.61s	4 371	36.61s

A noter que nous avons paramétré côté NGINX le proxy_max_temp_file_size à 100MB, aussi les fichiers en dessous de 100MB sont bufferisés côté NGINX mais pas au-dessus. Cela permet d'éviter de garder une connexion ouverte sur le apache pour la visualisation d'une vidéo de moins de 100MB donc notamment. Cependant, cela implique aussi le fait que le apache serve l'intégralité de la vidéo au NGINX même si finalement le client final stoppe au bout d'un moment son téléchargement.

Ainsi pour la dernière mesure indiquée dans ce tableau « Fichiers cours (mp4 uniquement /pluginfile.php) », on a relevé un transfert de 156GB côté apache mais « seulement » de 58GB côté NGINX.

On retrouve cette différence sur le total traité, alors même que NGINX sert 2 fois plus de requêtes que apache (fichiers statiques en cache) : 290GB pour apache vs 167GB pour NGINX. En plus de la différence constatée dû à l'avortement potentiel de téléchargement de certains fichiers, la différence est également due sur le total au fait que l'on a laissé le soin à NGINX de compresser les fichiers.

L'accès massif à des fichiers vidéo servis par le moodle peut mettre à mal le bon fonctionnement de la plateforme. Le module /pluginfile.php sert les fichiers au client (dont NGINX) en positionnant des entêtes HTTP interdisant la mise en cache. Du point de vue de moodle l'accès à ces fichiers est dynamique car dépendant de l'utilisateur connecté : cela permet notamment à moodle de restreindre l'accès en fonction de la bonne inscription de l'usager au cours par exemple. Sur des vidéos de plusieurs dizaines de MB accédées par 400 utilisateurs sur l'espace de quelques minutes, ce fonctionnement n'est pas efficace ; les connexions avec apache-prefork/mod_php sont couteûses notamment (sans parler également des accès disques effectués). Une solution (pas très élégante car mettant à mal le système de sécurité d'accès de la ressource) peut être de forcer les entêtes HTTP de cache côté apache pour permettre à NGINX de mettre en cache certaines vidéos de certains cours ne serait-ce que sur une période de quelques dizaines de minutes. Sinon la

mise en oeuvre de X-Sendfile[1] permettrait sans doute de répondre de manière élégante au moins en partie à ce problème. L'usage généralisé de la videothèque (en lieu et place du dépôt de vidéos dans moodle) permettrait enfin tout simplement ici de ne pas faire face à cette problématique côté moodle.

Un dernier point à souligner est le coût important de certaines requêtes côté enseignant. Dans le tableau ci-dessus, on illustre cette problématique avec les statistiques de temps de réponse de la consultation des rapports de quiz. Plus généralement, tous les accès en « report.php » ou « /report » sont relativement gourmands et peuvent prétendre à altérer la bonne réactivité du service moodle si ceux-ci sont demandés lors d'un usage marqué de la plateforme. Une demande auprès des enseignants peut permettre d'éviter que ces derniers sollicitent ces fonctionnalités lors des périodes de forte affluence. Nous avons de plus constaté que parfois, notamment lorsque les temps de réponse pour obtenir ces rapports sont longs, certains clients retentent l'opération alors même que la première requête n'a pas été traitée (pensant sans doute que cette première requête n'a pas été prise en compte par la plateforme). Ce comportement provoque alors une surconsommation des CPUs. Une solution éventuelle également (à l'étude pour notre plateforme moodle) serait de limiter l'accés à ces urls avec ngx_http_limit_conn_module[2] par exemple (1 seule requête en cours par IP pour le /report par exemple).

2 Côté client

Les mesures données ci-dessus n'indiquent malheureusement pas le temps effectif d'affichage d'une page côté utilisateur … et l'objectif est pourtant bien de proposer un service réactif à l'usager.

Le temps d'affichage comprend la récupération des ressources/media composants la page (html, css, js, images …) et leur interprétation pour affichage par le navigateur. L'interprétation dépend notamment du thème moodle choisi, partie qui ne fait pas partie de notre périmètre d'action ici (le thème d'UniversiTICE n'est d'ailleurs pas particulièrement gourmand).

Le temps de récupération des ressources dépend fortement de la connexion internet du client.

Côté client, le plus simple et rapide est de faire une mesure manuelle avec l'outil de développement intégré directement au navigateur (comme peuvent le fournir Mozilla Firefox ou Google Chrome).

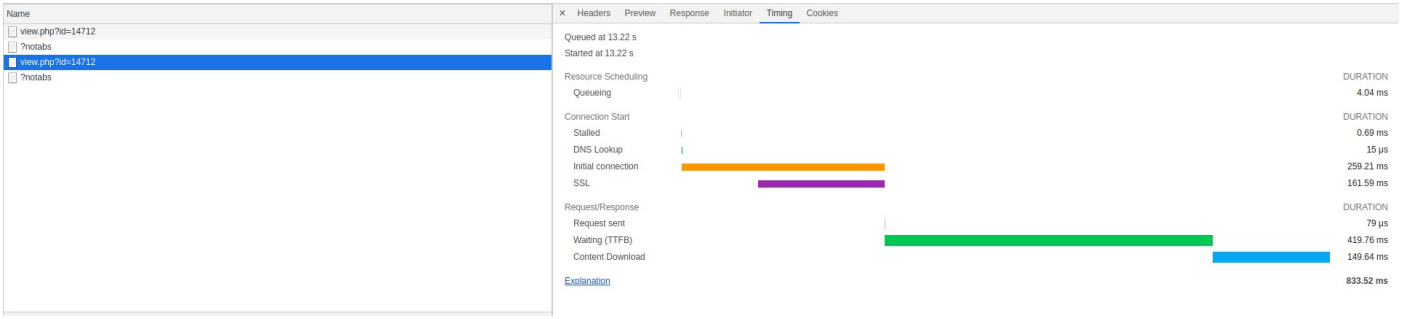
Avant les optimisations effectuées et décrites ici, nous avions fait une copie d'écran d'une mesure d'un accès sur une ressource donnée. Nous avons refait cette même mesure après optimisation. Nous sommes dans un environnement de connexion internet modeste mais relativement stable (0.8Mbit/s en débit montant, 2.1Mbit/s en débit descendant, ping de 30ms) ; lors de l'accès à la ressource, aucun autre accès internet n'est fait en parallèle, et nous souhaitons ici avoir une mesure reflétant au mieux le ressenti utilisateur lros de la navigation/utilisation du site : la mesure concerne donc la récupération du document html seulement correspondant à la page consultée, sachant qu'usuellement les medias js/css du thème sont en cache navigateur.

La page consultée est une page de cours où l'on est enseignant et qui comporte un nombre de blocs/sections relativement important dont le but est en fait d'illustrer la plupart des possibilités de UniversiTICE.

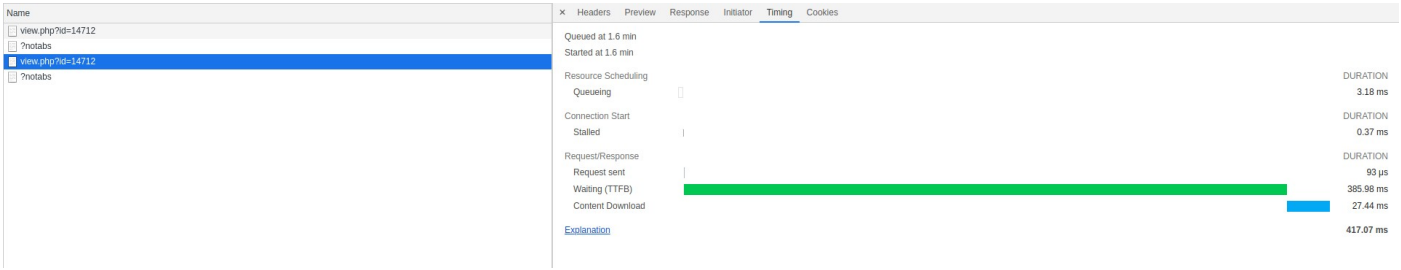
1 https://docs.moodle.org/38/en/Performance_recommendations#X-Sendfile

2 http://nginx.org/en/docs/http/ngx_http_limit_conn_module.html

Avant optimisation :



Après optimisation :



Lors du test de l'accès, après optimisation, on a également relevé les temps de réponse côté serveur ; pour le apache on relève 269ms et pour nginx on relève 270ms. Cela confirme que les mesures côté serveur utilisant uniquement les fichiers de logs d'accès ne sont pas toujours représentatives de l'expérience utilisateur.

Globalement, du point de vue du client on a ici divisé par 2 le temps de récupération de la ressource en passant de 833ms à 417ms.

Plus finement, il y a beaucoup plus à dire :

- keepalive

Avant optimisation, et avec apache-prefork/mod_php nous avions un keepalive de 2 secondes pour éviter d'avoir un nombre de connexions Apache réservées trop important. De fait entre 2 pages de navigation, il était rare que le client bénéficie d'un keepalive. Ici le keepalive (de 65 secondes sur NGINX) permet sur ce test de gagner près de 260ms au client ("initial connection" en orange). Si le gain est si important, c'est que la connexion internet est ici relativement lente.

- bufferisation

Le téléchargement est plus rapide après optimisation qu'avant (on passe de 150ms à 27ms), cela s'explique par le fait que lorsque NGINX commence à envoyer la réponse, la page est déjà entièrement calculée et prête. Il a en effet attendu qu'apache lui donne complètement la page avant de l'envoyer. De fait cependant, cela veut dire que ce temps de calcul complet est reporté dans le "waiting TTF" (Time To First Byte) qu'on peut considérer comme le temps de calcul de la page côté serveur.

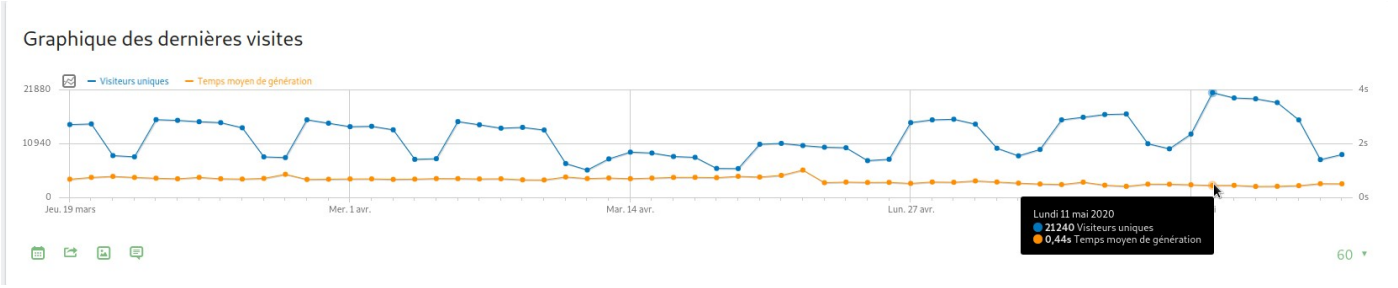
- temps de calcul

Alors même que l'on devrait reporter logiquement/théoriquement la différence de temps de transfert (150-27=123ms) dans le temps "waiting TTF", le "waiting TTF" passe de 420ms à 386ms … le temps de calcul côté serveur du point de vue du client passerait donc plutôt de 543ms à 386ms[3] soit 157ms de gain que l'on peut octroyer ici aux optimsiations réalisées côté serveur, notamment côté PostgreSQL ?

Ces accès ont été faits sur une période creuse … mais sur une période chargée (du point de vue global), et actuellement (après optimisation), les mesures restent stables. Le apache/mod_php/moodle n'étant à un instant t que peu sollicité : une dizaine de requêtes à traiter tout au plus en même temps alors qu'il dispose de 24 CPUs.

Les outils de statistiques web comme peut l'être Matomo (anciennement Piwik) intègrent le "temps de génération" de la page, ce temps correspond à peu de choses près au temps que nous venons de mesurer.

Il est intéressant de noter qu'en moyenne, ce temps n'a pas beaucoup varié, il est d'un peu moins de 0,5 seconde. Il a simplement augmenté vers la fin Avril avec un usage qui s'intensifiait sans qu'on ait encore eu le temps d'intervenir. Ce qui est à noter, c'est que malgré un usage important, il n'augmente pas.



VI. Conclusion

Disposer d'outils de métrique est indispensable. La configuration des logs pour y indiquer les temps de réponse est rapide et simple à mettre en œuvre, c'est la première chose que l'on a opérée.

Suite aux différentes optimisations décrites ici, les mesures indiquent une baisse de charge, de consommation de ressources, la disparition de pics, un nivellement.

Même côté utilisateur, une mesure avec les outils de développement montre un gain chiffré pour des connexions lentes : proposer un keepalive de 65 secondes peut tout à fait être perceptible.

Aussi on peut affirmer qu'après ces différentes optimisations, la plateforme répond mieux.

Cependant :

- l'application elle-même peut parfois être gourmande (quiz) ;
- l'usage est déterminant quand à la bonne tenue à la montée en charge ;
- les utilisateurs ont eux-mêmes potentiellement des difficultés techniques (accès internet, matériel informatique).

La prudence reste donc de mise.

La période intensive d'examens en ligne qui vient d'avoir lieu nous permet cependant maintenant d'être plus confiant quant au fonctionnement de la plateforme UniversiTICE en période chargée.

Ainsi durant la seule journée du lundi 11 mai, le serveur nginx a traité 3.560.000 et transféré 235GB de données ; plus de 15.000 utilisateurs authentifiés différents (données de notre serveur CAS) s'étant alors connectés/authentifiés pour accéder au service[4]. La moyenne du temps de réponse est restée à près de 260ms par requête côté serveur et 440ms côté client.

3 Pour cette requête avant optimisation, nous n'avons malheureusement pas le temps de calcul mesuré côté serveur ; on aurait sinon pu voir si on retrouvait une différence de temps d'exécution d'environ 157ms.

4 Les logs font ainsi état de 22600 adresses IP différentes pour 34000 visiteurs dits uniques (navigateur, péripéhrique, adresse).

Dire que ces optimisations techniques à elles seules ont permis aux étudiants de disposer d'une plateforme moodle fonctionnelle et réactive durant ces examens serait cependant faux. Le plus important a été réalisé au niveau fonctionnel. L'étalement des examens, l'attitude des enseignants (qui ont respecté les consignes d'éviter de trop solliciter la plateforme durant les examens … la consultation des résultats/rapports d'un quiz est particulièrement consommatrice de ressources par exemple) ainsi que des étudiants ont permis à la plateforme de tenir.

La mise en place d'un proxy NGINX nous a permis de palier les faiblesses de apache-prefork associé à mod_php, ce sans remettre en cause notre installation moodle alors que le service était devenu critique.

Les examens passés, on pense maintenant faire évoluer notre installation pour abandonner apache-prefork et mod_php en faveur de php-fpm comme recommandé dans la documentation moodle officielle (https://docs.moodle.org/38/en/Performance_recommendations). L'usage de X-Sendfile pourrait également permettre d'absorber de manière plus performante les accès aux fichiers lourds telles que les vidéos de cours (intégrées directement dans le moodle au lieu d'être proposées au travers de l'outil de videothèque ; cela démontre à nouveau ici que l'usage est déterminant).

VII. Remerciements

Ces travaux ont été réalisés conjointement par des membres du Service aux Usagers du Numérique et de la Direction des Systèmes d'Information de l'Université de Rouen Normandie.

Comme évoqué ci-dessus, le bon fonctionnement de la plateforme UniversiTICE durant cette période particulière est dû en partie aux optimisations techniques réalisées ; mais il est aussi dû en grande partie au bon usage qui en a été fait. On remercie donc ici l'ensemble de nos collègues personnels, enseignants, étudiants, et plus particluièrement nos gouvernances en matière de numérique (dont notre Vice Président du Numérique Vincent Roy).

Ce document est aussi l'occasion pour nous de remercier les collègues des autres établissements avec qui nous avons pu échanger.

Nous tenions tout particulièrement à remercier Pascal Rigaux de l'Université de Paris 1 (à qui nous avons repris toute la configuration NGINX) et l'ensemble du Groupe de Travail Développement EsupPortail que Pascal pilote.